Contradictory, My Dear Watson

Introduction

The task is to create an NLI model that assigns labels of 0, 1, or 2 (corresponding to entailment, neutral, and contradiction) to pairs of premises and hypotheses.

We will use traditional NLP approaches (TF-IDF vectorization, word embeddings, Bag-Of-Words), sentiment analysis and roBERTa using TPUs for the completion of the task.

Importing libraries

```
import os
os.system("pip3 install googletrans==3.1.0a0 > /dev/null 2>&1")
os.system("pip3 install seaborn > /dev/null 2>&1")
os.system("pip3 install transformers > /dev/null 2>&1")
os.system("pip3 install TextBlob > /dev/null 2>&1")
os.system("pip3 install spacy > /dev/null 2>&1")
os.system("pip3 install nltk > /dev/null 2>&1")
os.system("pip3 install scikit-learn > /dev/null 2>&1")
os.system("pip3 install lazypredict > /dev/null 2>&1")
os.system("pip3 install tensorflow > /dev/null 2>&1")
os.system("pip3 install sentencepiece> /dev/null 2>&1")
os.system("pip3 install datasets > /dev/null 2>&1")

for dirname, _, filenames in os.walk("/kaggle/input"):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
 /kaggle/input/contradictory-my-dear-watson/sample_submission.csv
 /kaggle/input/contradictory-my-dear-watson/train.csv
 /kaggle/input/contradictory-my-dear-watson/test.csv
```

```
#basics
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

# NLP specific libraries

from transformers import BertTokenizer, TFBertModel
from googletrans import Translator, LANGUAGES
from textblob import TextBlob
import spacy
import nltk
nltk.download('punkt')
nltk.download('stopwords')
from nltk.tokenize import word_tokenize
from nltk.stem import SnowballStemmer
from nltk.corpus import stopwords
from transformers import BertTokenizer, TFBertModel
import re
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

# Sklearn
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from sklearn.impute import KNNImputer
from sklearn.metrics.pairwise import cosine_similarity
```

```
# Various
import timeit
from lazypredict.Supervised import LazyClassifier
from datasets import load_dataset

# Tensorflow
import tensorflow as tf
from tensorflow.keras.wrappers.scikit_learn  import KerasClassifier
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping
import tensorflow_datasets as tfds
import tensorflow_addons as tfa
import tensorflow_probability as tfp
```

```
[nltk_data] Downloading package punkt to /usr/share/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

TPU distribution strategy

```
# Detect hardware, return appropriate distribution strategy
try:
    # TPU detection. No parameters necessary if TPU_NAME environment variable is
    # set: this is always the case on Kaggle.
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    print('Running on TPU ', tpu.master())
except ValueError:
    tpu = None

if tpu:
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
else:
    # Default distribution strategy in Tensorflow. Works on CPU and single GPU.
    strategy = tf.distribute.get_strategy()

print("REPLICAS: ", strategy.num_replicas_in_sync)
```

```
Running on TPU  grpc://10.0.0.2:8470
REPLICAS:  8
```

EDA

Loading data

```
train = pd.read_csv('/kaggle/input/contradictory-my-dear-watson/train.csv')
test = pd.read_csv('/kaggle/input/contradictory-my-dear-watson/test.csv')
print('Train set shape:', train.shape)
print('Test set shape:', test.shape)
train.head(10)
```

```
Train set shape: (12120, 6)
Test set set shape: (5195, 5)
```

	id	premise	hypothesis	lang_abv	language	label
0	5130fd2cb5	and these comments were considered in formulat...	The rules developed in the interim were put to...	en	English	0
1	5b72532a0b	These are issues that we wrestle with in	Practice groups are not permitted to work	en	English	2

```
print(train['premise'][3], '\n', train['hypothesis'][3])
```

```
you know they can't really defend themselves like somebody grown uh say my age you know yeah
 They can't defend themselves because of their age.
```

Descriptions of features

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12120 entries, 0 to 12119
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   id          12120 non-null  object
 1   premise     12120 non-null  object
 2   hypothesis  12120 non-null  object
 3   lang_abv    12120 non-null  object
 4   language    12120 non-null  object
 5   label       12120 non-null  int64 
dtypes: int64(1), object(5)
memory usage: 568.2+ KB
```

Distribution of the target variable

```
label_distribution = train["label"].value_counts(normalize = True)
print(label_distribution)
```

```
0   0.34
2   0.34
1   0.32
Name: label, dtype: float64
```

Missing data overview

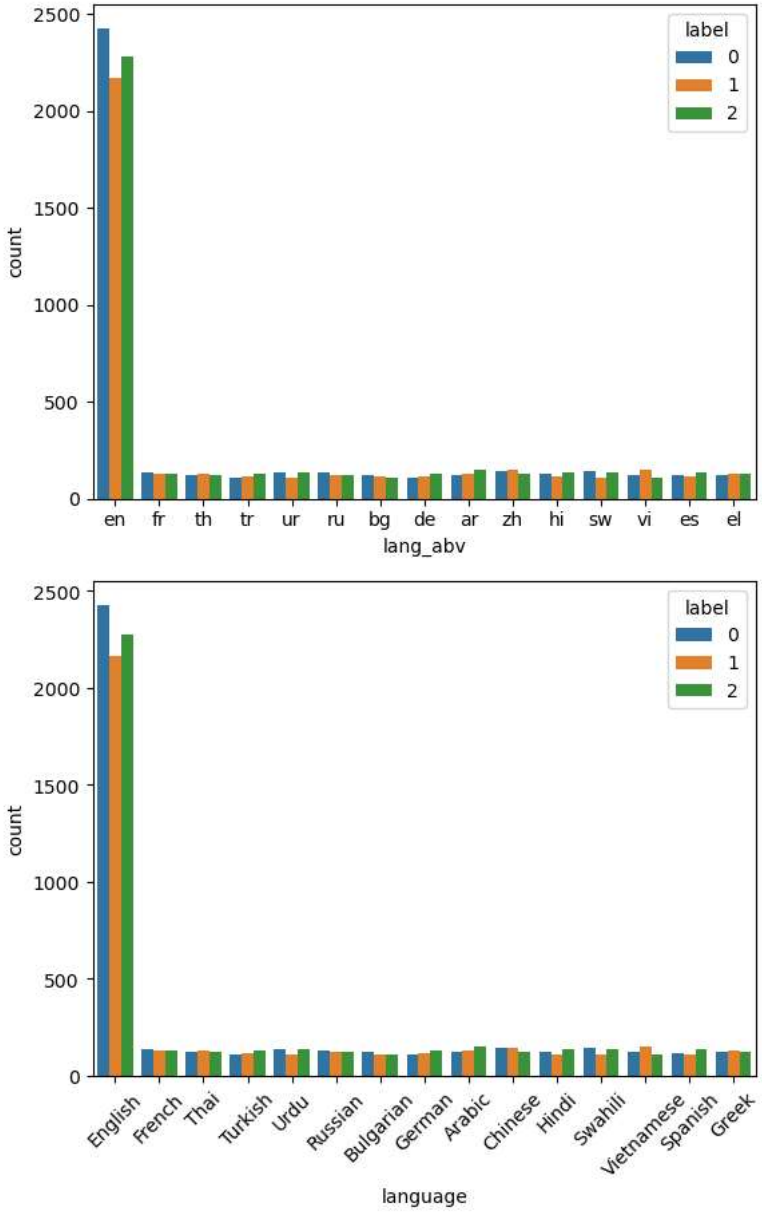
```
print("amount of missing values in train dataset:")
print(train.isna().sum())
print("amount of missing values in test dataset:")
print(test.isna().sum())
```

```
amount of missing values in train dataset:
id            0
premise       0
hypothesis    0
lang_abv      0
language      0
label         0
dtype: int64
amount of missing values in test dataset:
id            0
premise       0
hypothesis    0
lang_abv      0
language      0
dtype: int64
```

Distribution of categorical variables

```
# Plot distribution of languages for labels and comparison of two variables to check for difference between 2 features representing la
sns.countplot(data=train, x='lang_abv',  hue='label')
```

```
plt.show()
sns.countplot(data=train, x='language',  hue='label')
plt.xticks(rotation=45)
plt.show()
```



The variables may be considered identical.

```
language_distribution = train["language"].value_counts(normalize = True)
print(language_distribution)
```

```
English      0.57
Chinese      0.03
Arabic       0.03
French       0.03
Swahili      0.03
Urdu         0.03
Vietnamese   0.03
Russian      0.03
Hindi        0.03
Greek        0.03
Thai         0.03
Spanish      0.03
Turkish      0.03
German       0.03
Bulgarian    0.03
Name: language, dtype: float64
```

Translation of multilingual texts into english for standartization of modeling approach

```
train['language'] = train['language'].str.lower()
train['language'].head()
test['language'] = test['language'].str.lower()
```

```
# Get the language codes and names as a list of tuples
languages_list = [(code, name) for code, name in LANGUAGES.items()]

# Calculate the number of languages per column, rounded up
num_languages = len(languages_list)
num_columns = 4
languages_per_column = -(-num_languages // num_columns)

# Print the languages in 4 columns
for i in range(languages_per_column):
    for j in range(num_columns):
        index = i + j * languages_per_column
        if index < num_languages:
            code, name = languages_list[index]
            print(f"{code:<5} {name:<20}", end="")
    print()
```

```
af    afrikaans           fy    frisian             lo    lao                 st    sesotho
sq    albanian            gl    galician            la    latin               sn    shona
am    amharic             ka    georgian            lv    latvian             sd    sindhi
ar    arabic              de    german              lt    lithuanian          si    sinhala
hy    armenian            el    greek               lb    luxembourgish       sk    slovak
az    azerbaijani         gu    gujarati            mk    macedonian          sl    slovenian
eu    basque              ht    haitian creole      mg    malagasy            so    somali
be    belarusian          ha    hausa               ms    malay               es    spanish
bn    bengali             haw   hawaiian            ml    malayalam           su    sundanese
bs    bosnian             iw    hebrew              mt    maltese             sw    swahili
bg    bulgarian           he    hebrew              mi    maori               sv    swedish
ca    catalan             hi    hindi               mr    marathi             tg    tajik
ceb   cebuano             hmn   hmong               mn    mongolian           ta    tamil
ny    chichewa            hu    hungarian           my    myanmar (burmese)   te    telugu
zh-cn chinese (simplified)is    icelandic           ne    nepali              th    thai
zh-tw chinese (traditional)ig    igbo                no    norwegian           tr    turkish
co    corsican            id    indonesian          or    odia                uk    ukrainian
hr    croatian            ga    irish               ps    pashto              ur    urdu
cs    czech               it    italian             fa    persian             ug    uyghur
da    danish              ja    japanese            pl    polish              uz    uzbek
nl    dutch               jw    javanese            pt    portuguese          vi    vietnamese
en    english             kn    kannada             pa    punjabi             cy    welsh
eo    esperanto           kk    kazakh              ro    romanian            xh    xhosa
et    estonian            km    khmer               ru    russian             yi    yiddish
tl    filipino            ko    korean              sm    samoan              yo    yoruba
fi    finnish             ku    kurdish (kurmanji)  gd    scots gaelic        zu    zulu
fr    french              ky    kyrgyz              sr    serbian
```

```
train.loc[train['lang_abv'] == 'zh', 'lang_abv'] = 'zh-cn'
test.loc[test['lang_abv'] == 'zh', 'lang_abv'] = 'zh-cn'
```

```
def translate_text(text, lang, dest='en'):
    if lang != 'en':
        translator = Translator()
        text = translator.translate(text, src=lang, dest='en').text
    return text
print(translate_text("der Translator funktioniert wie gewünscht.", lang = 'de'))
```

```
the translator works as expected.
```

```
start_time = timeit.default_timer()  # start the timer
train_english_subset = train[:100].apply(lambda x: (translate_text(x['premise'], str(x['lang_abv']), 'en'), translate_text(x['hypothes
train_english_subset['premise_english'] = train_english_subset.apply(lambda x: x[0])
train_english_subset['hypothesis_english'] = train_english_subset.apply(lambda x: x[1])
end_time = timeit.default_timer()
elapsed_time = end_time - start_time  # calculate elapsed time
```

```
print(f"Total time taken: {elapsed_time:.2f} seconds")
```

```
start_time1 = timeit.default_timer()  # start the timer
premise_texts = train[:100]['premise']
hypothesis_texts = train[:100]['hypothesis']
langs = train[:100]['lang_abv']

translated_premise = [translate_text(premise, lang) for premise, lang in zip(premise_texts, langs)]
translated_hypothesis = [translate_text(hypothesis, lang) for hypothesis, lang in zip(hypothesis_texts, langs)]
end_time1 = timeit.default_timer()  # end the timer
elapsed_time1 = end_time1 - start_time1  # calculate elapsed time
print(f"Total time taken: {elapsed_time1:.2f} seconds")
```

```
Total time taken: 8.34 seconds
Total time taken: 6.17 seconds
```

```
train = train[train['lang_abv'] == 'en']
```

```
train.head()
```

	id	premise	hypothesis	lang_abv	language	label
0	5130fd2cb5	and these comments were considered in formulat...	The rules developed in the interim were put to...	en	english	0
1	5b72532a0b	These are issues that we wrestle with in pract...	Practice groups are not permitted to work on t...	en	english	2
3	5622f0c60b	you know they can't really defend	They can't defend themselves because	en	english	0

Sentiment analysis using TextBlob

```
# Perform sentiment analysis using TextBlob
sentiment_scores_premise = []
for text in train['premise']:
    blob = TextBlob(text)
    sentiment_scores_premise.append(blob.sentiment.polarity)
print(sentiment_scores_premise[:8])

sentiment_scores_hypothesis = []
for text in train['hypothesis']:
    blob1 = TextBlob(text)
    sentiment_scores_hypothesis.append(blob1.sentiment.polarity)
print(sentiment_scores_hypothesis[:8])
```

```
[0.0, 0.0, 0.2, 0.0, -0.05, 0.21666666666666667, 0.20625, 0.4]
[0.1, 0.0, 0.0, 0.0, 0.0, 0.5, 0.0, 0.0]
```

We will extract the difference in sentiments between premise and hypothesis

```
diff_in_sentiments = np.array(sentiment_scores_premise) - np.array(sentiment_scores_hypothesis)
diff_in_sentiments_labels = pd.DataFrame({'label': np.array(train['label']),
                                          'diff_in_sentiments': diff_in_sentiments,
                                          'language': np.array(train['language']),
                                           'sentiment_scores_premise': np.array(sentiment_scores_premise),
                                           'sentiment_scores_hypothesis': np.array(sentiment_scores_hypothesis)})
diff_in_sentiments_labels.head(3)
```

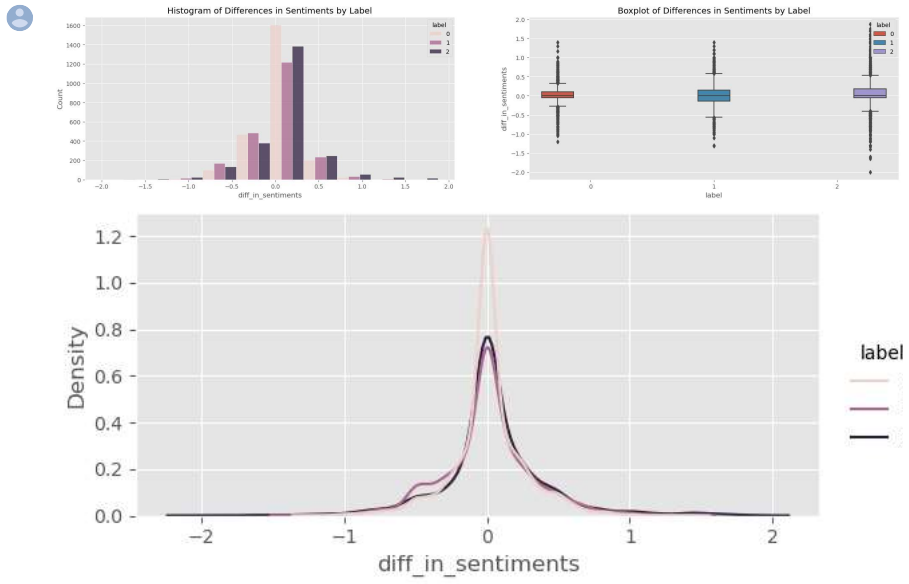
	label	diff_in_sentiments	language	sentiment_scores_premise	sentiment_scores
0	0	-0.10	english	0.00	
1	2	0.00	english	0.00	
2	0	0.20	english	0.20	

```
mean_sentiments_diff = diff_in_sentiments_labels[diff_in_sentiments_labels['language'] == 'english'].groupby('label').\
```

```
mean(["diff_in_sentiments", 'sentiment_scores_hypothesis', 'sentiment_scores_premise'])
mean_sentiments_diff
```

label	diff_in_sentiments	sentiment_scores_premise	sentiment_scores_hypothesis
0	0.02	0.09	0.07
1	0.01	0.09	0.08
2	0.05	0.08	0.03

```
plt.style.use('ggplot')
fig, axs = plt.subplots(ncols=2, figsize=(25, 5))

sns.histplot(data=diff_in_sentiments_labels,
             x='diff_in_sentiments',
             bins=10,  hue='label',  multiple = 'dodge', ax=axs[0])
axs[0].set(title='Histogram of Differences in Sentiments by Label')

sns.boxplot(data=diff_in_sentiments_labels,
            y='diff_in_sentiments',
            x='label', hue = 'label', ax=axs[1])
axs[1].set(title='Boxplot of Differences in Sentiments by Label')

sns.displot(data=diff_in_sentiments_labels,
            x='diff_in_sentiments', hue = 'label',  kind="kde",
            height=3, aspect=2)
plt.show()
```



The conclusions we might draw:

- the difference is very sparse, there are a lot of outliers by all labels.
- the absolute difference above 1 occurs more often by label 2.

- the text is more likely to belong to label 0 in case the difference in polarity is near zero.
- density plot shows a slight peak for label 1 around the values -0.6 to -0.3 for the difference in polarity.
- the polarity sentiment analysis is applicable only to the english texts, i.e. to 57% of our dataset.
- we might consider creating categorical variable for sentiments, that will split the data as follows:
    1. difference from -0.2 to 0.2
    2. difference from -0.6 to -0.3
    3. difference lower than -1.0 and higher than 1.0.
    4. texts on other languages together with the texts that do not belong to any of the above mentioned groups.

```
train['diff_in_sentiments'] = diff_in_sentiments
train['sentiment_group'] = np.nan
train.loc[(train['diff_in_sentiments'] >= -0.2) &
                            (train['diff_in_sentiments'] <= 0.2), 'sentiment_group'] = 'GROUP1'
train.loc[(train['diff_in_sentiments'] >= -0.6) &
                            (train['diff_in_sentiments'] <= -0.3), 'sentiment_group'] = 'GROUP2'
train.loc[(train['diff_in_sentiments'] >= 1.0) |
                            (train['diff_in_sentiments'] <= -1.0), 'sentiment_group'] = 'GROUP3'
train.loc[train['sentiment_group'].isna(), 'sentiment_group'] = 'GROUP4'

print(train['sentiment_group'].value_counts())
```

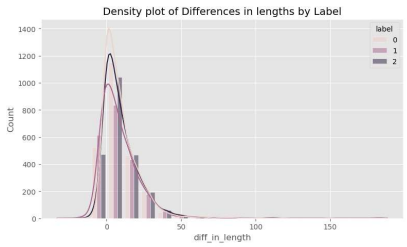
```
GROUP1    4444
GROUP4    1685
GROUP2     627
GROUP3     114
Name: sentiment_group, dtype: int64
```

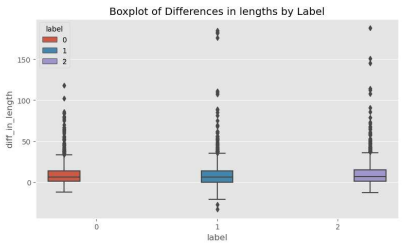
Number of words

```
text_len_premise=train['premise'].str.split().map(lambda x: len(x))
text_len_hypothesis=train['hypothesis'].str.split().map(lambda x: len(x))
train['diff_in_length'] = text_len_premise - text_len_hypothesis

train.head(2)
```

	id	premise	hypothesis	lang_abv	language	label	diff_in_sentiment
		and these comments were	The rules developed				

```
plt.style.use('ggplot')
fig, axs = plt.subplots(ncols=2, figsize=(20, 5))

sns.histplot(
    data=train,
    x="diff_in_length", hue="label", multiple = 'dodge',
    bins = 20, ax=axs[0], shrink=.8, kde =True)
axs[0].set(title='Density plot of Differences in lengths by Label')

sns.boxplot(
    data=train,
    y='diff_in_length',
    x='label', hue = 'label', ax=axs[1])
axs[1].set(title='Boxplot of Differences in lengths by Label')

plt.show()
```





We cannot draw many conclusions from the received distribution, but the difference below -5 indicates label 1, whereas the difference between 0 and 20 indicates that the likelyhood of label 1 is slighly lower than that of the other categories.

```
train['length_group'] = np.nan
train.loc[train['diff_in_length'] <= -5 , 'length_group'] = 'LENGTH_GROUP1'
train.loc[(train['diff_in_length'] >= 0) &
                            (train['diff_in_length'] <= 20), 'length_group'] = 'LENGTH_GROUP2'
train.loc[train['length_group'].isna(), 'length_group'] = 'LENGTH_GROUP3'
print(train['length_group'].value_counts())
```

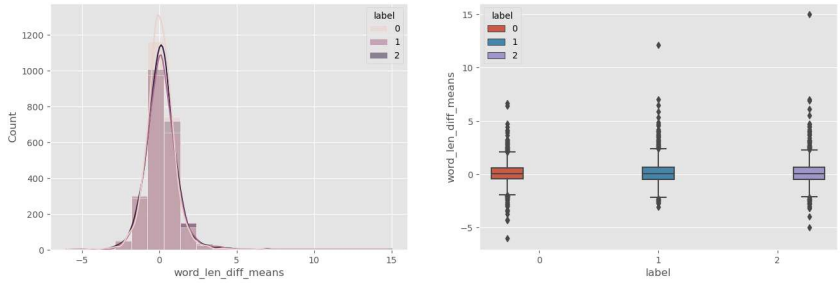
```
LENGTH_GROUP2    4682
LENGTH_GROUP3    1947
LENGTH_GROUP1     241
Name: length_group, dtype: int64
```

Average word length

```
word_len_premise=train['premise'].str.split().apply(lambda x : [len(i) for i in x])
word_len_hypothesis=train['hypothesis'].str.split().apply(lambda x : [len(i) for i in x])
from statistics import mean
word_len_premise_means = [mean(word_len) for word_len in word_len_premise]
word_len_hypothesis_means = [mean(word_len) for word_len in word_len_hypothesis]
train['word_len_diff_means'] = np.array(word_len_premise_means) - np.array(word_len_hypothesis_means)
print(train['word_len_diff_means'])
```

```
0        1.04
1       -0.50
3       -0.90
7       -0.14
8        0.21
         ...
12115    1.39
12116    0.54
12117    0.29
12118    0.00
12119   -0.25
Name: word_len_diff_means, Length: 6870, dtype: float64
```

```
fig, axs = plt.subplots(ncols=2, figsize=(16,5))

sns.histplot(
    data=train,
    x="word_len_diff_means", hue="label",
     ax=axs[0], kde = True, bins = 20)

sns.boxplot(
    data=train,
    y='word_len_diff_means',
    x='label', hue='label', ax=axs[1])

plt.show()
```



The distribution of differences in mean word lengthes seems nearly identical for target categories. We will not include this variable in our model.

Generating word embeddings via SpaCy

```
# Load the language model
!python -m spacy download en_core_web_sm --quiet
nlp = spacy.load("en_core_web_sm")

# Tokenize the sentence
spacy_premises_train =[nlp(sentence) for sentence in train["premise"]]
spacy_hypothesis_train =[nlp(sentence) for sentence in train["hypothesis"]]

word_embeddings_train_premises = []
for token in spacy_premises_train:
    word_embeddings_train_premises.append(token.vector)

word_embeddings_train_hypothesis = []
for token in spacy_hypothesis_train:
    word_embeddings_train_hypothesis.append(token.vector)
```

```
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manage
✓ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
```

Count Vectorizer + TF-IDF Vectorizer

```
def preprocess_text(text, lang='english'):
    # Tokenize the text
    tokens = word_tokenize(text)

    # Lowercase the tokens
    tokens = [token.lower() for token in tokens]

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in stop_words]

    # Stem the tokens
    stemmer = SnowballStemmer('english')
    tokens = [stemmer.stem(token) for token in tokens]
```

```
    return ' '.join(tokens)


def get_tokenized_data(data, lang_col, text_col):
    return [preprocess_text(row[text_col], row[lang_col]) for _, row in data.iterrows()]


train_tokenized_premises = get_tokenized_data(train, 'lang_abv', 'premise')
train_tokenized_hypothesis = get_tokenized_data(train, 'lang_abv', 'hypothesis')


# Create bag-of-words using CountVectorizer
count_vectorizer = CountVectorizer()
bow_train_premises = count_vectorizer.fit_transform(train_tokenized_premises)
bow_train_hypothesis = count_vectorizer.transform(train_tokenized_hypothesis)

# Create tf-idf vectors using TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()
tfidf_train_premises = tfidf_vectorizer.fit_transform(train_tokenized_premises)
tfidf_train_hypothesis = tfidf_vectorizer.transform(train_tokenized_hypothesis)
```

Cosine Similarity calculation

We generate cosine similarity between vectors and word embeddings for premises and hypothesis

```
tfidf_similarities_train = np.diag(cosine_similarity(tfidf_train_premises, tfidf_train_hypothesis))
bow_similarities_train = np.diag(cosine_similarity(bow_train_premises, bow_train_hypothesis))
embeddings_similarities_train = np.diag(cosine_similarity(word_embeddings_train_premises, word_embeddings_train_hypothesis))
```

Combining features

```
encoder = OneHotEncoder(handle_unknown='ignore')
train_groups_encoded = pd.DataFrame(encoder.fit_transform(train[['sentiment_group', 'length_group']]).toarray())


train_groups_encoded.head()
```

	0	1	2	3	4	5	6
0	1.00	0.00	0.00	0.00	0.00	0.00	1.00
1	1.00	0.00	0.00	0.00	0.00	1.00	0.00
2	1.00	0.00	0.00	0.00	0.00	1.00	0.00
3	1.00	0.00	0.00	0.00	0.00	1.00	0.00
4	1.00	0.00	0.00	0.00	0.00	1.00	0.00

```
print(tfidf_similarities_train.shape,
     bow_similarities_train.shape,
     embeddings_similarities_train.shape,
     train_groups_encoded.shape)
train_english_only = np.concatenate((tfidf_similarities_train.reshape(-1,1),
                                     bow_similarities_train.reshape(-1,1),
                                     embeddings_similarities_train.reshape(-1,1),
                                     train_groups_encoded), axis=1)
print(train_english_only.shape)
train_english_only = pd.DataFrame(train_english_only)
```

```
(6870,) (6870,) (6870,) (6870, 7)
(6870, 10)
```

Prediction from LazyClassifier

```
y = train['label']
X_train, X_val_complete, y_train, y_val = train_test_split(train_english_only, y, test_size=0.2, random_state=42)
clf = LazyClassifier(predictions=True, verbose=0, random_state=42)
models, predictions = clf.fit(X_train, X_val_complete, y_train, y_val)
models.head(10)
```

```
100%|██████████| 29/29 [00:19<00:00,  1.45it/s]
```

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
AdaBoostClassifier	0.45	0.45	None	0.44	0.31
SVC	0.44	0.45	None	0.43	2.53
BernoulliNB	0.44	0.44	None	0.43	0.02
LogisticRegression	0.44	0.44	None	0.43	0.06
RidgeClassifier	0.44	0.44	None	0.42	0.03
LinearSVC	0.44	0.44	None	0.42	1.30
RidgeClassifierCV	0.44	0.44	None	0.42	0.05
LGBMClassifier	0.44	0.44	None	0.43	0.41
CalibratedClassifierCV	0.44	0.44	None	0.42	4.36
LinearDiscriminantAnalysis	0.43	0.44	None	0.42	0.06

Traditional NLP approaches (TF-IDF vectorization, word embeddings, Bag-Of-Words) have not brought much for the NLI task of differentiation between relation of 2 sentences. Sentiment analysis has been also less predictive then expected.

Natural Language Inference with roBERTa

The approach from BertExperiment II https://www.kaggle.com/code/nicknosorogov/bertexperiment-ii has been used as a basis. The main difference - training of the model on only English training samples. Prior to applying model to the test set we will translate it into english.

```
train = pd.read_csv('/kaggle/input/contradictory-my-dear-watson/train.csv')
train = train[['premise', 'hypothesis', 'lang_abv', 'label']]
train = train[train['lang_abv'] == 'en']


def load_mnli(use_validation=True):
    result=[]
    dataset=load_dataset('multi_nli')
    print(dataset)
    for record in dataset['train']:
        c1, c2, c3 = record['premise'],record['hypothesis'], record['label']
        if c1 and c2 and c3 in {0, 1, 2}:
            result.append((c1, c2, c3, 'en'))
    result=pd.DataFrame(result, columns=['premise', 'hypothesis', 'label', 'lang_abv'])
    return result

mnli=load_mnli()
mnli
```

```
Downloading builder script:   0%|          | 0.00/1.90k [00:00<?, ?B/s]
Downloading metadata:   0%|          | 0.00/1.19k [00:00<?, ?B/s]
Downloading and preparing dataset multi_nli/default (download: 216.34 MiB, generat
Downloading data:   0%|          | 0.00/227M [00:00<?, ?B/s]
Generating train split:   0%|          | 0/392702 [00:00<?, ? examples/s]
Generating validation_matched split:   0%|          | 0/9815 [00:00<?, ?
examples/s]
Generating validation_mismatched split:   0%|          | 0/9832 [00:00<?, ?
examples/s]
Dataset multi_nli downloaded and prepared to /root/.cache/huggingface/datasets/mul
  0%|          | 0/3 [00:00<?, ?it/s]
DatasetDict({
    train: Dataset({
        features: ['promptID', 'pairID', 'premise', 'premise_binary_parse', 'premi
        num_rows: 392702
    })
    validation_matched: Dataset({
        features: ['promptID', 'pairID', 'premise', 'premise_binary_parse', 'premi
        num_rows: 9815
    })
    validation_mismatched: Dataset({
        features: ['promptID', 'pairID', 'premise', 'premise_binary_parse', 'premi
        num_rows: 9832
    })
})
```

	premise	hypothesis	label	lang_abv
0	Conceptually cream skimming has two basic dime...	Product and geography are what make cream skim...	1	en
1	you know during the season and i guess at at y...	You lose the things to the following level if ...	0	en

```
from transformers import BertTokenizer, TFBertModel, AutoTokenizer,TFAutoModel
model_roBerta ='joeddav/xlm-roberta-large-xnli'
#model_Bert = 'bert-base-multilingual-cased'
tokenizer = AutoTokenizer.from_pretrained(model_roBerta)
model = TFAutoModel.from_pretrained(model_roBerta)
```

```
Downloading (…)okenizer_config.json:   0%|          | 0.00/25.0 [00:00<?, ?B/s]
Downloading (…)lve/main/config.json:   0%|          | 0.00/734 [00:00<?, ?B/s]
Downloading (…)tencepiece.bpe.model:   0%|          | 0.00/5.07M [00:00<?, ?B/s]
Downloading (…)cial_tokens_map.json:   0%|          | 0.00/150 [00:00<?, ?B/s]
Downloading (…)"tf_model.h5";:   0%|          | 0.00/2.24G [00:00<?, ?B/s]
Some layers from the model checkpoint at joeddav/xlm-roberta-large-xnli were not u
- This IS expected if you are initializing TFXLMRobertaModel from the checkpoint o
- This IS NOT expected if you are initializing TFXLMRobertaModel from the checkpoi
All the layers of TFXLMRobertaModel were initialized from the model checkpoint at
If your task is similar to the task the model of the checkpoint was trained on, yo
```

```
train=pd.concat([train, mnli.loc[:20000]], axis=0)
SEQ_LEN = 236

def bert_encode(df, tokenizer):
    batch_premises = df['premise'].tolist()
    batch_hypothesis = df['hypothesis'].tolist()

    tokens = tokenizer(batch_premises, batch_hypothesis, max_length = SEQ_LEN,
                   truncation=True, padding='max_length',
                   add_special_tokens=True, return_attention_mask=True,
                   return_tensors='tf')
    inputs = {
          'input_ids': tokens['input_ids'],
          'attention_mask': tokens['attention_mask'],
            }
    return inputs
train_input = bert_encode(train, tokenizer)


from tensorflow.keras import regularizers

def build_model():   # hp
    #FBertModel
    encoder = TFAutoModel.from_pretrained(model_roBerta)
    input_ids = tf.keras.Input(shape=(SEQ_LEN,), dtype=tf.int32, name="input_ids")
```

```
    attention_mask = tf.keras.Input(shape=(SEQ_LEN,), dtype=tf.int32, name="attention_mask")

    embedding = encoder([input_ids, attention_mask])[0]
    inputs=[input_ids, attention_mask]
    hp_units1 = 64
    hp_units2 = 32
    x = tf.keras.layers.Dense(units = hp_units1, activation=tf.nn.relu)(embedding[:,0,:])
    x = tf.keras.layers.Dense(units = hp_units2, activation=tf.nn.relu, kernel_regularizer=regularizers.l2(l2=1e-4))(x)
    output = tf.keras.layers.Dense(3, activation='softmax')(x)    #  (embedding[:,0,:])

    model = tf.keras.Model(inputs=inputs, outputs=output)
    hp_learning_rate = 1e-6
    model.compile(tf.keras.optimizers.Adam(learning_rate = hp_learning_rate), loss='sparse_categorical_crossentropy', metrics=['accura
    return model


with strategy.scope(): # defines the compute distribution policy for building the model. or in other words: makes sure that the model
    model = build_model() # our model is being built
    model.summary()       # let's look at some of its properties
```

```
Some layers from the model checkpoint at joeddav/xlm-roberta-large-xnli were not used when initializing TFXLMRobertaModel: ['clas
- This IS expected if you are initializing TFXLMRobertaModel from the checkpoint of a model trained on another task or with anoth
- This IS NOT expected if you are initializing TFXLMRobertaModel from the checkpoint of a model that you expect to be exactly ide
All the layers of TFXLMRobertaModel were initialized from the model checkpoint at joeddav/xlm-roberta-large-xnli.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFXLMRobertaModel for predict
Model: "model"
_________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_ids (InputLayer)         [(None, 236)]        0           []

 attention_mask (InputLayer)    [(None, 236)]        0           []

 tfxlm_roberta_model_1 (TFXLMRo  TFBaseModelOutputWi  559890432  ['input_ids[0][0]',
 bertaModel)                    thPoolingAndCrossAt               'attention_mask[0][0]']
                                tentions(last_hidde
                                n_state=(None, 236,
                                 1024),
                                 pooler_output=(Non
                                e, 1024),
                                 past_key_values=No
                                ne, hidden_states=N
                                one, attentions=Non
                                e, cross_attentions
                                =None)

 tf.__operators__.getitem (Slic  (None, 1024)        0           ['tfxlm_roberta_model_1[0][0]']
 ingOpLambda)

 dense (Dense)                  (None, 64)           65600       ['tf.__operators__.getitem[0][0]'
                                                                 ]

 dense_1 (Dense)                (None, 32)           2080        ['dense[0][0]']

 dense_2 (Dense)                (None, 3)            99          ['dense_1[0][0]']

==================================================================================================
Total params: 559,958,211
Trainable params: 559,958,211
Non-trainable params: 0
_________________________________________________________________________________________________
```

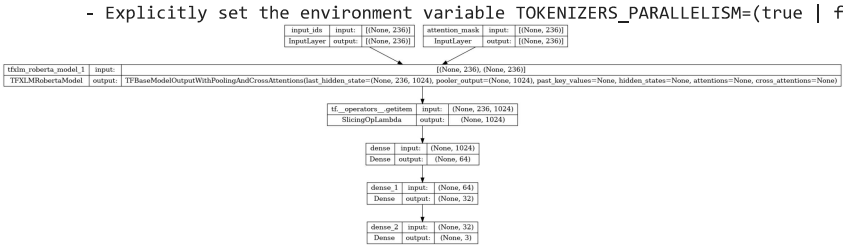
```
for key in train_input.keys():
    train_input[key] = train_input[key][:,:SEQ_LEN]


history = model.fit(train_input, train['label'], epochs = 10, batch_size=64,
                    validation_split = 0.2) #,callbacks=[hist]) verbose = 1,
```

```
Epoch 1/10
336/336 [==============================] - 369s 617ms/step - loss: 0.3577 - accuracy: 0.8864 - val_loss: 0.1850 - val_accuracy: 0
Epoch 2/10
336/336 [==============================] - 101s 301ms/step - loss: 0.2426 - accuracy: 0.9209 - val_loss: 0.1696 - val_accuracy: 0
Epoch 3/10
336/336 [==============================] - 101s 301ms/step - loss: 0.2198 - accuracy: 0.9272 - val_loss: 0.1638 - val_accuracy: 0
Epoch 4/10
336/336 [==============================] - 101s 301ms/step - loss: 0.2011 - accuracy: 0.9344 - val_loss: 0.1683 - val_accuracy: 0
```

```
Epoch 5/10
336/336 [==============================] - 101s 301ms/step - loss: 0.1872 - accuracy: 0.9389 - val_loss: 0.1651 - val_accuracy: 0
Epoch 6/10
336/336 [==============================] - 101s 300ms/step - loss: 0.1662 - accuracy: 0.9463 - val_loss: 0.1710 - val_accuracy: 0
Epoch 7/10
336/336 [==============================] - 101s 301ms/step - loss: 0.1585 - accuracy: 0.9499 - val_loss: 0.1679 - val_accuracy: 0
Epoch 8/10
336/336 [==============================] - 101s 301ms/step - loss: 0.1485 - accuracy: 0.9542 - val_loss: 0.1718 - val_accuracy: 0
Epoch 9/10
336/336 [==============================] - 101s 301ms/step - loss: 0.1360 - accuracy: 0.9584 - val_loss: 0.1778 - val_accuracy: 0
Epoch 10/10
336/336 [==============================] - 101s 301ms/step - loss: 0.1216 - accuracy: 0.9631 - val_loss: 0.1874 - val_accuracy: 0
```

```
tf.keras.utils.plot_model(model, show_shapes=True)
```

```
huggingface/tokenizers: The current process just got forked, after parallelism has
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | f
huggingface/tokenizers: The current process just got forked, after parallelism has
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | f
```



Google Translator

Since the model has learned only on English texts we will translate the test samples prior to predicting.

```
def translate(text):
    # Since not all of the languages from the dataset are represented in nltk modules we will remove
    # the stopwords and do stemming only for appicable languages only after translation.
    try:
       # if lang not in ['english', 'french', 'russian', 'german', 'arabic', 'spanish']:
        translator = Translator()
        return translator.translate(text, dest='en').text
    except:
        return text


test_translated = test.apply(lambda x: (translate_text(x['premise'], str(x['lang_abv']), 'en'),
                                        translate_text(x['hypothesis'], str(x['lang_abv']), 'en')), axis=1) ## applying translation to
test['premise'] = test_translated.apply(lambda x: x[0])
test['hypothesis'] = test_translated.apply(lambda x: x[1])
test['lang_abv'] = 'en'
```

```
test_input = bert_encode(test, tokenizer)
 #same for the test set we need to put it in the same size of the model
for key in test_input.keys():
    test_input[key] = test_input[key][:,:SEQ_LEN]
predictions = [np.argmax(i) for i in model.predict(test_input)]
```

```
163/163 [==============================] - 35s 169ms/step
```

Submission

```
submission = test.id.copy().to_frame()
submission['prediction'] = predictions
submission.to_csv("submission.csv", index = False)
```

Final thoughts

Traditional NLP approaches (TF-IDF vectorization, word embeddings, Bag-Of-Words) have not brought much for the NLI task of differentiation between relation of 2 sentences. Sentiment analysis has been also less predictive than expected.

The best prediction could be achieved with the help of multi_nli and RoBERTa. Applying it to only English samples did not approve the performance significantly. However applying prior translation might be a good step in the standartization pipelene for other NLP tasks.