Intoroduction

Natural Language Inferencing (NLI) task is one of the most important subsets of Natural Language Processing (NLP). NLI is also known as Recognizing Textual Entailment (RTE). NLI is a task of determining whether the given “hypothesis” and “premise” logically follow (entailment) or unfollow (contradiction) or are undetermined (neutral) to each other. In other words, the task of NLI is also defined as determining whether a "hypothesis" is true (entailment), false (contradiction), or undetermined (neutral) given a "premise". Table 1 gives some examples of these three cases.

Text (Premise)	Hypothesis	Judgement (Label)
A soccer game with multiple males playing.	Some men are playing a sport.	entailment
---	---	---
A black race car starts up in front of a crowd of people.	A man is driving down a lonely road.	contradiction
---	---	---
An older and a younger man smiling.	Two men are smiling and laughing at the cats playing on the floor.	neutral

Type *Markdown* and LaTeX: α^2

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import re
from nltk.corpus import stopwords
import pickle
from tqdm import tqdm
from wordcloud import WordCloud
from sklearn.feature_extraction.text import TfidfVectorizer
import tensorflow as tf
import random
import math
from textblob import TextBlob
from collections import Counter
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
import nltk, string
from collections import Counter
from nltk.stem import WordNetLemmatizer
import Levenshtein
%matplotlib inline
```

In [2]:
```
1 nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\anike\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[2]: True

In [2]:
```
1 # Setting seed
2 global_seed = 42
3 np.random.seed(42)
4 random.seed(42)
5 tf.random.set_seed(42)
```

We have already extracted some features in the exploratory phase of the project. We will directly import data with those extracted features. We will start modelling with those features only. Then we will add features using word embeddings

Extracting word embedding features:average glove

In [11]:
```
1 embeddings_index = dict()
2 f = open('glove.6B.300d.txt', encoding="utf8")
3 for line in f:
4     values = line.split()
5     word = values[0]
6     coefs = np.asarray(values[1:], dtype='float32')
7     embeddings_index[word] = coefs
8 f.close()
```

In [12]:
```
1 glove_words=set(embeddings_index.keys())
```

train sentence1 sentence representations

In [13]:
```
from tqdm import tqdm
x_train_s1_tfidfw2v  = []
for sentence in tqdm(train_df['sentence1_preprocessed'].values):
    vector = np.zeros(300)
    cnt_words =0
    for word in sentence.split():
        if word in glove_words:
            vector += embeddings_index[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    x_train_s1_tfidfw2v.append(vector)

print(len(x_train_s1_tfidfw2v))
print(len(x_train_s1_tfidfw2v[0]))
```

```
100%|██████████| 5
49361/549361 [00:22<00:00, 23936.04it/s]

549361
300
```

train sentence2 sentence representations

In [14]:
```
from tqdm import tqdm
x_train_s2_tfidfw2v  = []
for sentence in tqdm(train_df['sentence2_preprocessed'].values):
    vector = np.zeros(300)
    cnt_words =0
    for word in sentence.split():
        if word in glove_words:
            vector += embeddings_index[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    x_train_s2_tfidfw2v.append(vector)

print(len(x_train_s2_tfidfw2v))
print(len(x_train_s2_tfidfw2v[0]))
```

```
100%|██████████| 5
49361/549361 [00:16<00:00, 33748.84it/s]

549361
300
```

validation sentence1 sentence representations

In [15]:
```
from tqdm import tqdm
x_val_s1_tfidfw2v  = [];
for sentence in tqdm(val_df['sentence1_preprocessed'].values):
    vector = np.zeros(300)
    cnt_words =0
    for word in sentence.split():
        if word in glove_words:
            vector += embeddings_index[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    x_val_s1_tfidfw2v.append(vector)

print(len(x_val_s1_tfidfw2v))
print(len(x_val_s1_tfidfw2v[0]))
```

```
100%|
| 9842/9842 [00:00<00:00, 27434.12it/s]

9842
300
```

validation sentence2 sentence representations

In [16]:
```
from tqdm import tqdm
x_val_s2_tfidfw2v  = []
for sentence in tqdm(val_df['sentence2_preprocessed'].values):
    vector = np.zeros(300)
    cnt_words =0
    for word in sentence.split():
        if word in glove_words:
            vector += embeddings_index[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    x_val_s2_tfidfw2v.append(vector)

print(len(x_val_s2_tfidfw2v))
print(len(x_val_s2_tfidfw2v[0]))
```

```
100%|
| 9842/9842 [00:00<00:00, 44841.91it/s]

9842
300
```

test sentence1 sentence representations

In [17]:
```
from tqdm import tqdm
x_test_s1_tfidfw2v  = [];
for sentence in tqdm(test_df['sentence1_preprocessed'].values):
    vector = np.zeros(300)
    cnt_words =0;
    for word in sentence.split():
        if word in glove_words:
            vector += embeddings_index[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    x_test_s1_tfidfw2v.append(vector)

print(len(x_test_s1_tfidfw2v))
print(len(x_test_s1_tfidfw2v[0]))
```

```
100%|
| 9824/9824 [00:00<00:00, 28765.11it/s]

9824
300
```

test sentence2 sentence representations

In [18]:
```
from tqdm import tqdm
x_test_s2_tfidfw2v  = [];
for sentence in tqdm(test_df['sentence2_preprocessed'].values):
    vector = np.zeros(300)
    cnt_words =0
    for word in sentence.split():
        if word in glove_words:
            vector += embeddings_index[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    x_test_s2_tfidfw2v.append(vector)

print(len(x_test_s2_tfidfw2v))
print(len(x_test_s2_tfidfw2v[0]))
```

```
100%|
| 9824/9824 [00:00<00:00, 40082.10it/s]

9824
300
```

In [19]:
```
print(type(x_train_s1_tfidfw2v))
print(type(x_train_s1_tfidfw2v[0]))
```

```
<class 'list'>
<class 'numpy.ndarray'>
```

In [20]:
```
print(type(X_train))
```

```
<class 'pandas.core.frame.DataFrame'>
```

In [21]:
```
x_train_s1_tfidfw2v = np.vstack(x_train_s1_tfidfw2v)
x_train_s2_tfidfw2v = np.vstack(x_train_s2_tfidfw2v)
x_val_s1_tfidfw2v = np.vstack(x_val_s1_tfidfw2v)
x_val_s2_tfidfw2v = np.vstack(x_val_s2_tfidfw2v)

```

In [22]:
```
x_test_s1_tfidfw2v = np.vstack(x_test_s1_tfidfw2v)
x_test_s2_tfidfw2v = np.vstack(x_test_s2_tfidfw2v)
```

In [23]:
```
np.save('x_train_s1_tfidfw2v.npy', x_train_s1_tfidfw2v)
np.save('x_train_s2_tfidfw2v.npy', x_train_s2_tfidfw2v)
np.save('x_val_s1_tfidfw2v.npy', x_val_s1_tfidfw2v)
np.save('x_val_s2_tfidfw2v.npy', x_val_s2_tfidfw2v)
np.save('x_test_s1_tfidfw2v.npy', x_test_s1_tfidfw2v)
np.save('x_test_s2_tfidfw2v.npy', x_test_s2_tfidfw2v)
```

In [13]:
```
x_train_s1_tfidfw2v = np.load('x_train_s1_tfidfw2v.npy')
x_train_s2_tfidfw2v = np.load('x_train_s2_tfidfw2v.npy')
x_val_s1_tfidfw2v = np.load('x_val_s1_tfidfw2v.npy')
x_val_s2_tfidfw2v = np.load('x_val_s2_tfidfw2v.npy')
x_test_s1_tfidfw2v = np.load('x_test_s1_tfidfw2v.npy')
x_test_s2_tfidfw2v = np.load('x_test_s2_tfidfw2v.npy')
```

cosine similarity between average glove

In [29]:
```
def tfidf_w2v_cosine(mat1, mat2):
    """
    Given two matrices, returns a list of cosine similarities between rows o
    """

    from scipy.spatial.distance import cosine
    sim = [1-cosine(i,j) for i,j in zip(mat1, mat2)]
    return sim


```

In [30]:
```
train_tfidf_w2v_cosine = tfidf_w2v_cosine(x_train_s1_tfidfw2v, x_train_s2_tf
```

```
C:\Users\anike\.conda\envs\tf_test\lib\site-packages\scipy\spatial\distance.py:
720: RuntimeWarning: invalid value encountered in double_scalars
  dist = 1.0 - uv / np.sqrt(uu * vv)
```

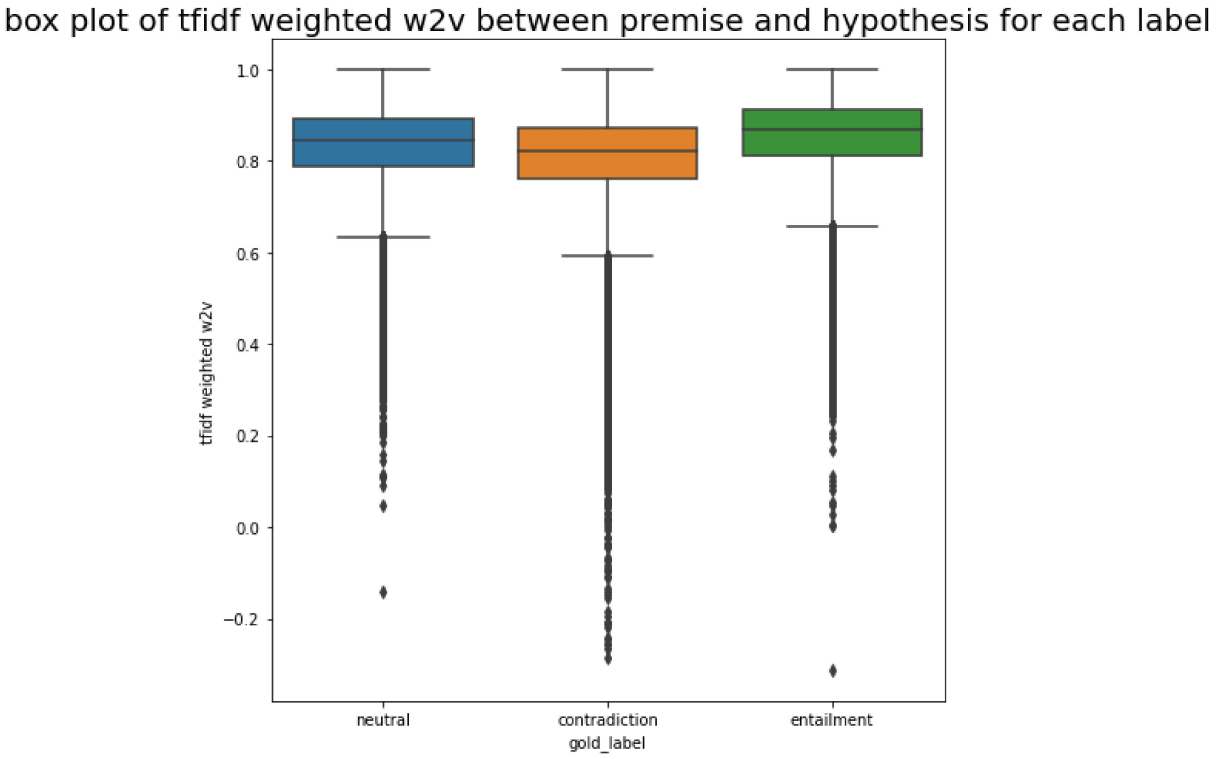
In [36]:
```
val_tfidf_w2v_cosine = tfidf_w2v_cosine(x_val_s1_tfidfw2v, x_val_s2_tfidfw2v
```

In [41]:
```
test_tfidf_w2v_cosine = tfidf_w2v_cosine(x_test_s1_tfidfw2v, x_test_s2_tfidf
```

In [42]:
```
train_df['tfidf_w2v_cosine'] = train_tfidf_w2v_cosine
val_df['tfidf_w2v_cosine'] = val_tfidf_w2v_cosine
test_df['tfidf_w2v_cosine'] = test_tfidf_w2v_cosine
```

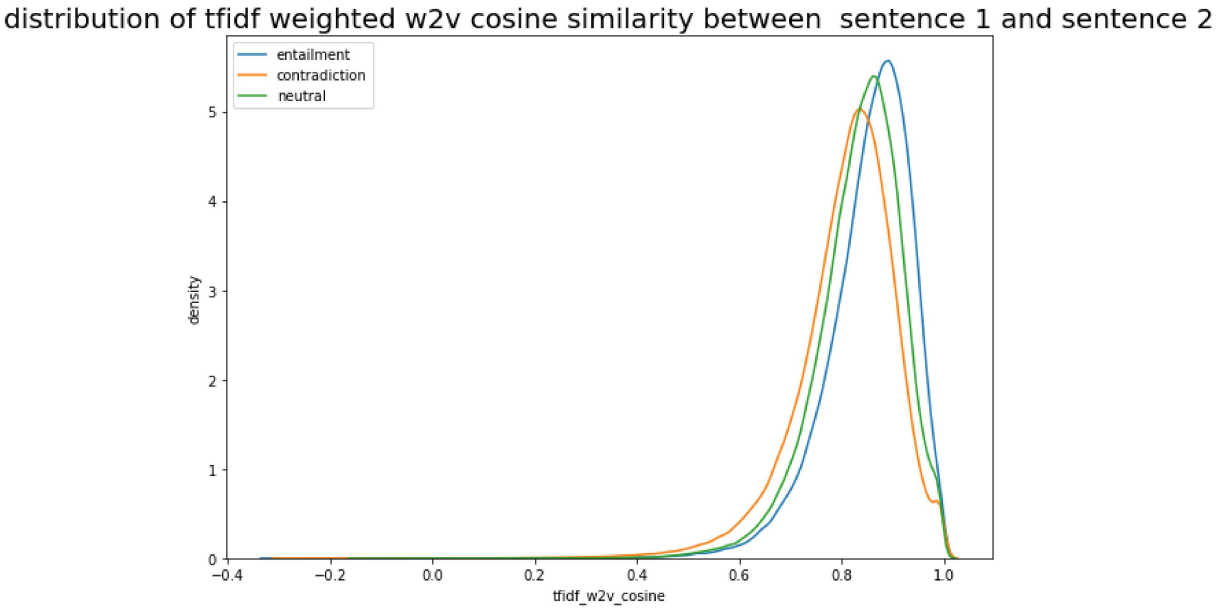
In [69]:
```
train_df = train_df.fillna(0)
```

In [43]:
```
fig,ax = plt.subplots(1,1, figsize=(7,7))
sns.boxplot(x = train_df['gold_label'], y = train_df['tfidf_w2v_cosine'], ax
ax.set_title('box plot of tfidf weighted w2v between premise and hypothesis
ax.set_ylabel('tfidf weighted w2v')
plt.tight_layout()
plt.show()
```



As expected, the tfidf weighted w2v similarity between two sentences is more for entailment sentences. It is least for contradictory sentences

In [44]:
```
fig,ax = plt.subplots(1,1, figsize=(10,7))
sns.kdeplot(x = train_df[train_df['gold_label'] == 'entailment']['tfidf_w2v_
sns.kdeplot(x = train_df[train_df['gold_label'] == 'contradiction']['tfidf_w
sns.kdeplot(x = train_df[train_df['gold_label'] == 'neutral']['tfidf_w2v_cos
ax.set_xlabel('tfidf_w2v_cosine')
ax.set_ylabel('density')
ax.set_title('distribution of tfidf weighted w2v cosine similarity between
plt.legend()
plt.show()
```



In [70]:
```
train_df.to_csv('train_df.csv')
val_df.to_csv('val_df.csv')
test_df.to_csv('test_df.csv')
```

In [71]:
```
train_df = pd.read_csv('train_df.csv', index_col = 0)
val_df = pd.read_csv('val_df.csv', index_col = 0)
test_df = pd.read_csv('test_df.csv', index_col = 0)
```

In [72]:
```
to_keep_x = ['sentence1_num_words', 'sentence2_num_words', 'log_bleu_score',
       'cosine_similarity', 'tfidf_cosine_sim', 'diff_num_words', 's1_CC',
       's1_IN', 's1_JJ', 's1_JJR', 's1_JJS','s1_PRP', 's1_PRP$', 's1_RB', 's
       's2_JJ', 's2_JJR', 's2_JJS', 's2_NN', 's2_NNP', 's2_NNS', 's2_PRP',
       's2_PRP$', 's2_RB', 's2_RBR', 's2_RBS', 's2_UH', 's2_VB', 's2_VBD',
       's2_VBG', 's2_VBN', 's2_VBP', 's2_VBZ', 'ratio_noun', 'ratio_verb', '

X_train = train_df[to_keep_x]
y_train = train_df['gold_label']
X_val = val_df[to_keep_x]
y_val = val_df['gold_label']
```

Training random forest model without average

glove vectors

In [73]:
```
from sklearn.ensemble import RandomForestClassifier
model2 = RandomForestClassifier(max_depth=80, min_samples_leaf=2, min_sample
                        n_estimators=800)
model2.fit(X_train, y_train)
```

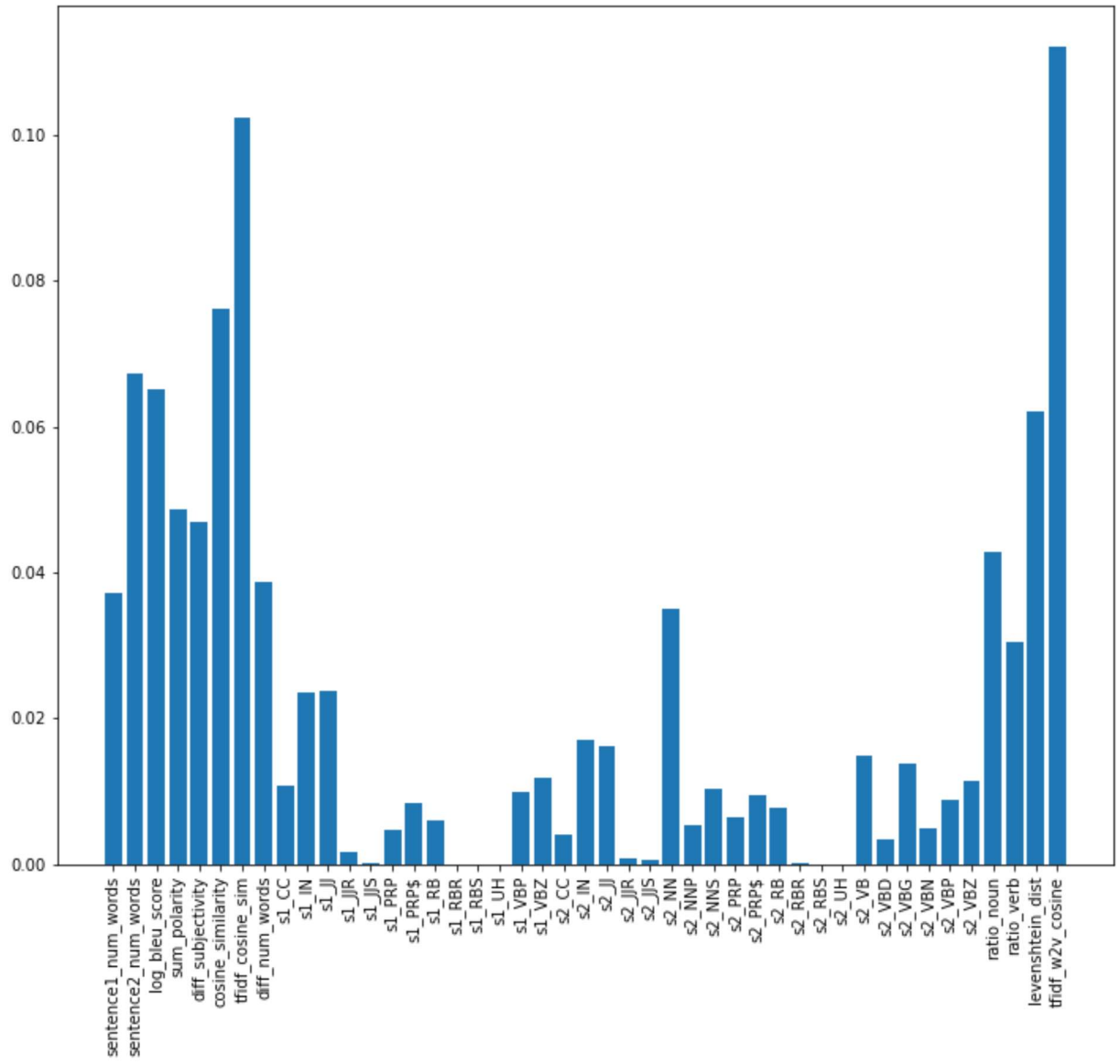
Out[73]:
```
RandomForestClassifier(max_depth=80, min_samples_leaf=2, min_samples_split=10,
                       n_estimators=800)
```

In [75]:
```
val_pred = model2.predict(X_val)

from sklearn.metrics import accuracy_score
acc = accuracy_score(y_val, val_pred)

acc
```

Out[75]:
```
0.5821987400934769
```

with only extracted features we are getting accuracy of 0.58

In [77]:
```
importance = model2.feature_importances_

# Plotting feature importance
columns = train_df[to_keep_x].columns
x_pos = np.arange(len(columns))
plt.figure(figsize = (12,10))
plt.bar(x_pos, importance)
plt.xticks(x_pos, columns, rotation='vertical')
plt.show()
#plt.plot(np.array(range(0,47)), importance, 'bo' )
```



Training model with extracted features + avg glove vectors of both sentences

In [84]:
```
x_train = np.hstack((X_train, x_train_s1_tfidfw2v, x_train_s2_tfidfw2v))
x_val = np.hstack((X_val, x_val_s1_tfidfw2v, x_val_s2_tfidfw2v))
```

In [86]:
```
from sklearn.ensemble import RandomForestClassifier
model3 = RandomForestClassifier(max_depth=80, min_samples_leaf=2, min_sample
                        n_estimators=800)
model3.fit(x_train, y_train)
```

Out[86]:
```
RandomForestClassifier(max_depth=80, min_samples_leaf=2, min_samples_split=10,
                       n_estimators=800)
```

In [88]:
```
val_pred = model3.predict(x_val)

from sklearn.metrics import accuracy_score
acc = accuracy_score(y_val, val_pred)

acc
```

Out[88]:
```
0.6633814265393213
```

after including average word vectors accuracy increases to 0.66

Considering ngrams: tfidf cosine similarity with bigrams, trigrams and quadgrams

In [89]:
```
#https://stackoverflow.com/questions/8897593/how-to-compute-the-similarity-b
stemmer = nltk.stem.porter.PorterStemmer()
remove_punctuation_map = dict((ord(char), None) for char in string.punctuati

def stem_tokens(tokens):
    return [stemmer.stem(item) for item in tokens]

'''remove punctuation, lowercase, stem'''
def normalize(text):
    return stem_tokens(nltk.word_tokenize(text.lower().translate(remove_punc

vectorizer = TfidfVectorizer(tokenizer=normalize, ngram_range=(2, 4))#we gav

def cosine_sim(text1, text2):
    tfidf = vectorizer.fit_transform([text1, text2])
    return ((tfidf * tfidf.T).A)[0,1]
```

ngram tfidf cosine similarity for training data

In [90]:
```
tfidf_cos_sim_ngram = list()
for i,row in train_df.iterrows():
    cos_sim = cosine_sim(row['sentence1'], row['sentence2'])
    tfidf_cos_sim_ngram.append(cos_sim)

train_df['tfidf_cosine_sim_ngram'] = tfidf_cos_sim_ngram
```

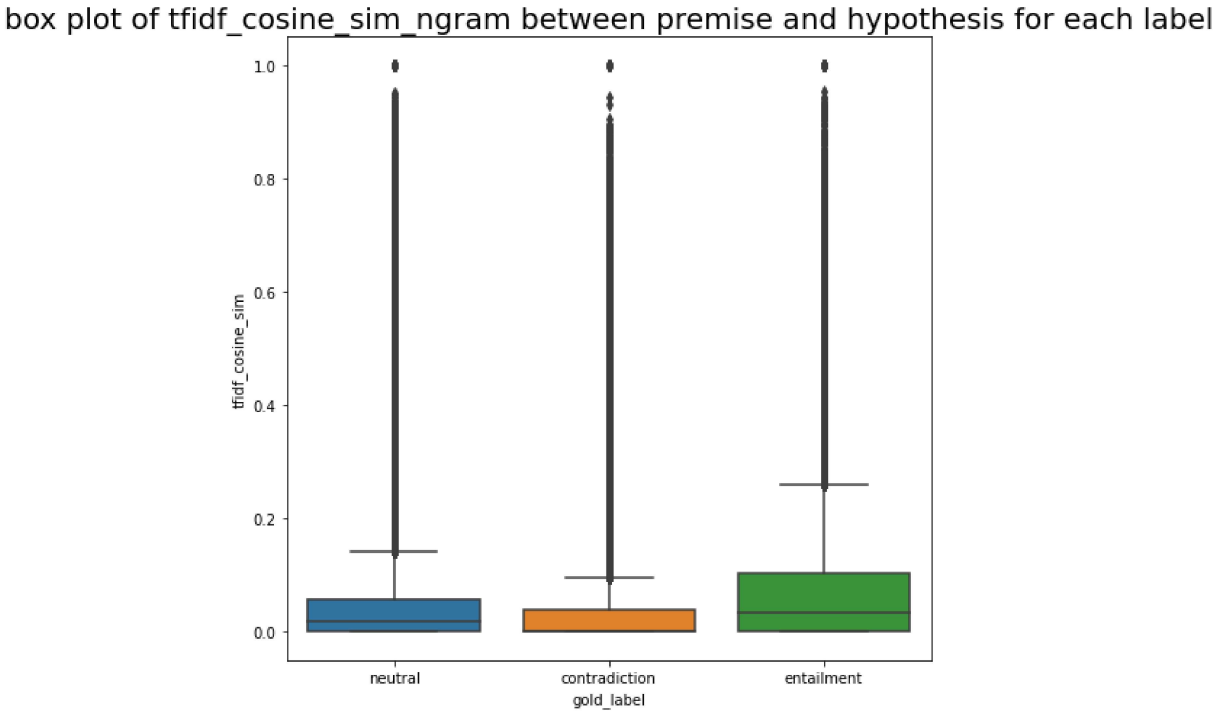
ngram tfidf cosine similarity for val data

In [91]:
```
tfidf_cos_sim_ngram = list()
for i,row in val_df.iterrows():
    cos_sim = cosine_sim(row['sentence1'], row['sentence2'])
    tfidf_cos_sim_ngram.append(cos_sim)

val_df['tfidf_cosine_sim_ngram'] = tfidf_cos_sim_ngram
```

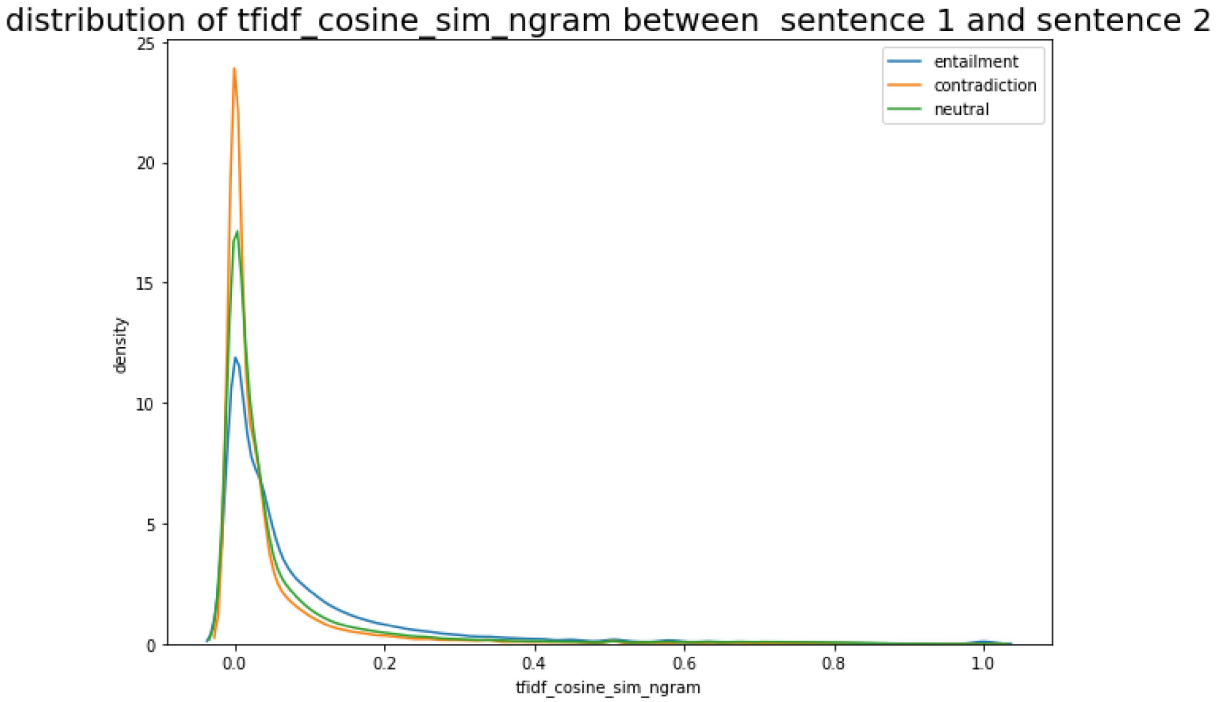
ngram tfidf cosine similarity for test data

In [92]:
```
tfidf_cos_sim_ngram = list()
for i,row in test_df.iterrows():
    cos_sim = cosine_sim(row['sentence1'], row['sentence2'])
    tfidf_cos_sim_ngram.append(cos_sim)

test_df['tfidf_cosine_sim_ngram'] = tfidf_cos_sim_ngram
```

In [94]:
```
fig,ax = plt.subplots(1,1, figsize=(7,7))
sns.boxplot(x = train_df['gold_label'], y = train_df['tfidf_cosine_sim_ngram
ax.set_title('box plot of tfidf_cosine_sim_ngram between premise and hypothe
ax.set_ylabel('tfidf_cosine_sim')
plt.tight_layout()
plt.show()
```



In [95]:
```
fig,ax = plt.subplots(1,1, figsize=(10,7))
sns.kdeplot(x = train_df[train_df['gold_label'] == 'entailment']['tfidf_cosi
sns.kdeplot(x = train_df[train_df['gold_label'] == 'contradiction']['tfidf_c
sns.kdeplot(x = train_df[train_df['gold_label'] == 'neutral']['tfidf_cosine_
ax.set_xlabel('tfidf_cosine_sim_ngram')
ax.set_ylabel('density')
ax.set_title('distribution of tfidf_cosine_sim_ngram between  sentence 1 and
plt.legend()
plt.show()
```



In [99]:
```
train_df.to_csv('train_df.csv')
val_df.to_csv('val_df.csv')
test_df.to_csv('test_df.csv')
```

In [5]:
```
train_df = pd.read_csv('train_df.csv', index_col = 0)
val_df = pd.read_csv('val_df.csv', index_col = 0)
test_df = pd.read_csv('test_df.csv', index_col = 0)
```

In [3]:
```
to_keep = ['sentence1_num_words', 'sentence2_num_words',
       'log_bleu_score', 'sentence1_polarity',
       'sentence2_polarity', 'sentence1_subjectivity',
       'sentence2_subjectivity',
       'cosine_similarity', 'tfidf_cosine_sim', 's1_CC',
       's1_IN', 's1_JJ', 's1_JJR', 's1_JJS', 's1_NN', 's1_NNP', 's1_NNS',
       's1_PRP', 's1_PRP$', 's1_RB', 's1_RBR', 's1_RBS', 's1_UH', 's1_VB',
       's1_VBD', 's1_VBG', 's1_VBN', 's1_VBP', 's1_VBZ', 's2_CC', 's2_IN',
       's2_JJ', 's2_JJR', 's2_JJS', 's2_NN', 's2_NNP', 's2_NNS', 's2_PRP',
       's2_PRP$', 's2_RB', 's2_RBR', 's2_RBS', 's2_UH', 's2_VB', 's2_VBD',
       's2_VBG', 's2_VBN', 's2_VBP', 's2_VBZ', 'levenshtein_dist', 'tfidf_w2
       'tfidf_cosine_sim_ngram']
```

In [4]:
```
X_train = train_df[to_keep]
y_train = train_df['gold_label']
X_val = val_df[to_keep]
y_val = val_df['gold_label']
```

In [7]:
```
x_train = np.hstack((X_train, x_train_s1_tfidfw2v, x_train_s2_tfidfw2v))
x_val = np.hstack((X_val, x_val_s1_tfidfw2v, x_val_s2_tfidfw2v))
```

In [26]:
```
from sklearn.ensemble import RandomForestClassifier
model4 = RandomForestClassifier(max_depth=80, min_samples_leaf=2, min_sample
                        n_estimators=800)
model4.fit(x_train, y_train)


```

Out[26]:
```
RandomForestClassifier(max_depth=80, min_samples_leaf=2, min_samples_split=10,
                       n_estimators=800)
```

In [27]:
```
val_pred = model4.predict(x_val)
from sklearn.metrics import accuracy_score
acc = accuracy_score(y_val, val_pred)
acc
```

Out[27]:
```
0.662466978256452
```

random forest best score 0.66 on validation set

XGBoost

In [32]:
```
to_keep = ['log_bleu_score', 'sum_polarity', 'diff_subjectivity',
       'cosine_similarity', 'tfidf_cosine_sim', 'diff_num_words', 's1_CC',
       's1_IN','s1_PRP', 's1_PRP$', 's1_RB', 's1_RBR', 's1_RBS', 's1_UH', 's
       's2_PRP$', 's2_RB', 's2_RBR', 's2_RBS', 's2_UH', 'ratio_noun', 'ratio
       'diff_adjective', 'levenshtein_dist', 'tfidf_w2v_cosine',
       'tfidf_cosine_sim_ngram']
```

In [5]:
```
X_train = train_df[to_keep]
y_train = train_df['gold_label']
X_val = val_df[to_keep]
y_val = val_df['gold_label']

```

In [ ]:
```
x_train = np.hstack((X_train, x_train_s1_tfidfw2v, x_train_s2_tfidfw2v))
x_val = np.hstack((X_val, x_val_s1_tfidfw2v, x_val_s2_tfidfw2v))
```

In [7]:
```
#classes for grid search and cross-validation, function for splitting data a
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV,train_te
from sklearn.metrics import accuracy_score,f1_score,roc_auc_score,confusion_
X_tr, X_te, y_tr, y_te = train_test_split(X_train, y_train, test_size = 0.4,
```

In [10]:
```
# hyperparameter tuning
#XGBoost library
import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV
param_grid = {'gamma': [0,0.1,0.4,0.8,3.2,6.4,25.6,102.4, 200],
              'learning_rate': [0.01,0.001, 0.1, 0.5],
              'max_depth': [3,4,5,9,12],
              'n_estimators': [50,100,200,400,800],
              'reg_alpha': [0,0.1,0.001,1,5,100,200],
              'reg_lambda': [0,0.1,0.001,1,5,100,200]}
xgbc = xgb.XGBClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
xgb_random = RandomizedSearchCV(estimator = xgbc, param_distributions = para
# Fit the random search model
xgb_random.fit(X_tr, y_tr)
```

```
C:\Users\anike\.conda\envs\tf_test\lib\site-packages\xgboost\sklearn.py:1224: U
serWarning: The use of label encoder in XGBClassifier is deprecated and will be
removed in a future release. To remove this warning, do the following: 1) Pass
option use_label_encoder=False when constructing XGBClassifier object; and 2) E
ncode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_clas
s - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[14:03:35] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'multi:softprob' was changed from 'merror' to 'mloglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[10]:
```
RandomizedSearchCV(cv=4,
                   estimator=XGBClassifier(base_score=None, booster=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None,
                                           enable_categorical=False, gamma=Non
e,
                                           gpu_id=None, importance_type=None,
                                           interaction_constraints=None,
                                           learning_rate=None,
                                           max_delta_step=None, max_depth=None,
                                           min_child_weight=None, missing=nan,
                                           monotone_constraints...
                                           subsample=None, tree_method=None,
                                           validate_parameters=None,
                                           verbosity=None),
                   n_iter=50, n_jobs=6,
                   param_distributions={'gamma': [0, 0.1, 0.4, 0.8, 3.2, 6.4,
                                                  25.6, 102.4, 200],
                                        'learning_rate': [0.01, 0.001, 0.1,
                                                          0.5],
                                        'max_depth': [3, 4, 5, 9, 12],
                                        'n_estimators': [50, 100, 200, 400,
                                                         800],
                                        'reg_alpha': [0, 0.1, 0.001, 1, 5, 100,
                                                      200],
```

```
                                                'reg_lambda': [0, 0.1, 0.001, 1, 5, 10
0,
                                                               200]},
                        random_state=42)
```

In [11]:
```
1 xgb_random.best_estimator_
```

Out[11]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
              gamma=0.1, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.1, max_delta_step=0,
              max_depth=12, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=200, n_jobs=16,
              num_parallel_tree=1, objective='multi:softprob', predictor='aut
o',
              random_state=0, reg_alpha=0, reg_lambda=100,
              scale_pos_weight=None, subsample=1, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

In [14]:
```
1 X_train = train_df[to_keep]
2 y_train = train_df['gold_label']
3 X_val = val_df[to_keep]
4 y_val = val_df['gold_label']
5 x_train = np.hstack((X_train, x_train_s1_tfidfw2v, x_train_s2_tfidfw2v))
6 x_val = np.hstack((X_val, x_val_s1_tfidfw2v, x_val_s2_tfidfw2v))
```

In [18]:
```
1  xgbc = xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel
2                colsample_bynode=1, colsample_bytree=1, enable_categorical=Fal
3                gamma=0.1, gpu_id=-1, importance_type=None,
4                interaction_constraints='', learning_rate=0.1, max_delta_step=
5                max_depth=12, min_child_weight=1, missing=np.nan,
6                monotone_constraints='()', n_estimators=200, n_jobs=16,
7                num_parallel_tree=1, objective='multi:softprob', predictor='au
8                random_state=0, reg_alpha=0, reg_lambda=100,
9                scale_pos_weight=None, subsample=1, tree_method='exact',
10               validate_parameters=1, verbosity=None)
```

In [22]:
```
1 y_train.replace(to_replace = ['neutral', 'contradiction', 'entailment'] , va
2 y_val.replace(to_replace = ['neutral', 'contradiction', 'entailment'] , valu
```

In [25]:
```
1 print(x_train.shape)
2 print(x_val.shape)
```

```
(549361, 629)
(9842, 629)
```

In [23]:
```
1 xgbc.fit(x_train, y_train)
```

```
[14:15:04] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'multi:softprob' was changed from 'merror' to 'mloglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[23]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
              gamma=0.1, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.1, max_delta_step=0,
              max_depth=12, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=200, n_jobs=16,
              num_parallel_tree=1, objective='multi:softprob', predictor='aut
o',
              random_state=0, reg_alpha=0, reg_lambda=100,
              scale_pos_weight=None, subsample=1, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

In [26]:
```
1 val_pred = xgbc.predict(x_val)
2 from sklearn.metrics import accuracy_score
3 acc = accuracy_score(y_val, val_pred)
4 acc
```

Out[26]:
```
0.7087990245884983
```

xgboost best score 0.7 on validation set(with extracted features+average word embeddings(GLoVe))

Embeddings using fasttext

In [4]:
```
1 import fasttext
```

In [20]:
```
#loading oretrained fasttext model
#https://fasttext.cc/docs/en/crawl-vectors.html
import fasttext.util
fasttext.util.download_model('en', if_exists='ignore')  # English
ft = fasttext.load_model('cc.en.300.bin')
```

```
Downloading https://dl.fbaipublicfiles.com/fasttext/vectors-crawl/cc.en.300.bin.gz (https://dl.fbaipublicfiles.com/fasttext/vectors-crawl/cc.en.300.bin.gz)
```

train sentence1 average fasttext

In [23]:
```
from tqdm import tqdm
x_train_s1_avg_fasttext  = []
for sentence in tqdm(train_df['sentence1_preprocessed'].values):
    vector = np.zeros(300)
    cnt_words =0
    for word in sentence.split():
        vector += ft.get_word_vector(word)
        cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    x_train_s1_avg_fasttext.append(vector)

print(len(x_train_s1_avg_fasttext))
print(len(x_train_s1_avg_fasttext[0]))
```

```
100%|██████████| 549361/549361 [01:04<00:00, 8521.67it/s]

549361
300
```

train sentence2 average fasttext

In [24]:
```
from tqdm import tqdm
x_train_s2_avg_fasttext  = []
for sentence in tqdm(train_df['sentence2_preprocessed'].values):
    vector = np.zeros(300)
    cnt_words =0
    for word in sentence.split():
        vector += ft.get_word_vector(word)
        cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    x_train_s2_avg_fasttext.append(vector)

print(len(x_train_s2_avg_fasttext))
print(len(x_train_s2_avg_fasttext[0]))
```

```
100%|████████████████████████████████████████████████████████████████████| 5
49361/549361 [00:40<00:00, 13669.84it/s]

549361
300
```

val sentence1 average fasttext

In [25]:
```
from tqdm import tqdm
x_val_s1_avg_fasttext  = []
for sentence in tqdm(val_df['sentence1_preprocessed'].values):
    vector = np.zeros(300)
    cnt_words =0
    for word in sentence.split():
        vector += ft.get_word_vector(word)
        cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    x_val_s1_avg_fasttext.append(vector)

print(len(x_val_s1_avg_fasttext))
print(len(x_val_s1_avg_fasttext[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
██| 9842/9842 [00:01<00:00, 7761.10it/s]

9842
300
```

val sentence2 average fasttext

In [26]:
```
from tqdm import tqdm
x_val_s2_avg_fasttext  = []
for sentence in tqdm(val_df['sentence2_preprocessed'].values):
    vector = np.zeros(300)
    cnt_words =0
    for word in sentence.split():
        vector += ft.get_word_vector(word)
        cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    x_val_s2_avg_fasttext.append(vector)

print(len(x_val_s2_avg_fasttext))
print(len(x_val_s2_avg_fasttext[0]))
```

```
100%|
| 9842/9842 [00:00<00:00, 12560.89it/s]

9842
300
```

test sentence1 average fasttext

In [27]:
```
from tqdm import tqdm
x_test_s1_avg_fasttext  = []
for sentence in tqdm(test_df['sentence1_preprocessed'].values):
    vector = np.zeros(300)
    cnt_words =0
    for word in sentence.split():
        vector += ft.get_word_vector(word)
        cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    x_test_s1_avg_fasttext.append(vector)

print(len(x_test_s1_avg_fasttext))
print(len(x_test_s1_avg_fasttext[0]))
```

```
100%|
| 9824/9824 [00:01<00:00, 7749.76it/s]

9824
300
```

test sentence2 average fasttext

In [28]:
```
from tqdm import tqdm
x_test_s2_avg_fasttext  = []
for sentence in tqdm(test_df['sentence2_preprocessed'].values):
    vector = np.zeros(300)
    cnt_words =0
    for word in sentence.split():
        vector += ft.get_word_vector(word)
        cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    x_test_s2_avg_fasttext.append(vector)

print(len(x_test_s2_avg_fasttext))
print(len(x_test_s2_avg_fasttext[0]))
```

```
100%|
| 9824/9824 [00:00<00:00, 13819.69it/s]

9824
300
```

saving the fastext ecoded array

In [29]:
```
x_train_s1_avg_fasttext = np.vstack(x_train_s1_avg_fasttext)
x_train_s2_avg_fasttext = np.vstack(x_train_s2_avg_fasttext)
x_val_s1_avg_fasttext = np.vstack(x_val_s1_avg_fasttext)
x_val_s2_avg_fasttext = np.vstack(x_val_s2_avg_fasttext)
x_test_s1_avg_fasttext = np.vstack(x_test_s1_avg_fasttext)
x_test_s2_avg_fasttext = np.vstack(x_test_s2_avg_fasttext)
```

In [30]:
```
np.save('x_train_s1_avg_fasttext.npy', x_train_s1_avg_fasttext)
np.save('x_train_s2_avg_fasttext.npy', x_train_s2_avg_fasttext)
np.save('x_val_s1_avg_fasttext.npy', x_val_s1_avg_fasttext)
np.save('x_val_s2_avg_fasttext.npy', x_val_s2_avg_fasttext)
np.save('x_test_s1_avg_fasttext.npy', x_test_s1_avg_fasttext)
np.save('x_test_s2_avg_fasttext.npy', x_test_s2_avg_fasttext)
```

In [31]:
```
x_train_s1_avg_fasttext = np.load('x_train_s1_avg_fasttext.npy', mmap_mode='
x_train_s2_avg_fasttext = np.load('x_train_s2_avg_fasttext.npy', mmap_mode='
x_val_s1_avg_fasttext = np.load('x_val_s1_avg_fasttext.npy', mmap_mode='r')
x_val_s2_avg_fasttext = np.load('x_val_s2_avg_fasttext.npy', mmap_mode='r')
x_test_s1_avg_fasttext = np.load('x_test_s1_avg_fasttext.npy', mmap_mode='r'
x_test_s2_avg_fasttext = np.load('x_test_s2_avg_fasttext.npy', mmap_mode='r'
```

Xgboost with fasttext embeddings

In [33]:
```
X_train = train_df[to_keep]
y_train = train_df['gold_label']
X_val = val_df[to_keep]
y_val = val_df['gold_label']
x_train = np.hstack((X_train, x_train_s1_avg_fasttext, x_train_s2_avg_fastte
x_val = np.hstack((X_val, x_val_s1_avg_fasttext, x_val_s2_avg_fasttext))
```

In [34]:
```
y_train.replace(to_replace = ['neutral', 'contradiction', 'entailment'] , va
y_val.replace(to_replace = ['neutral', 'contradiction', 'entailment'] , valu
```

In [40]:
```
import xgboost as xgb
xgbc = xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel
                colsample_bynode=1, colsample_bytree=1, enable_categorical=Fal
                gamma=0.1, gpu_id=-1, importance_type=None,
                interaction_constraints='', learning_rate=0.1, max_delta_step=
                max_depth=12, min_child_weight=1, missing=np.nan,
                monotone_constraints='()', n_estimators=200, n_jobs=16,
                num_parallel_tree=1, objective='multi:softprob', predictor='au
                random_state=0, reg_alpha=0, reg_lambda=100,
                scale_pos_weight=None, subsample=1, tree_method='exact',
                validate_parameters=1, verbosity=None, use_label_encoder=False
```

In [41]:
```
xgbc.fit(x_train, y_train)
```

```
[11:36:17] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'multi:softprob' was changed from 'merror' to 'mloglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[41]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
              gamma=0.1, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.1, max_delta_step=0,
              max_depth=12, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=200, n_jobs=16,
              num_parallel_tree=1, objective='multi:softprob', predictor='aut
o',
              random_state=0, reg_alpha=0, reg_lambda=100,
              scale_pos_weight=None, subsample=1, tree_method='exact',
              use_label_encoder=False, validate_parameters=1, verbosity=None)
```

In [42]:
```
val_pred = xgbc.predict(x_val)
from sklearn.metrics import accuracy_score
acc = accuracy_score(y_val, val_pred)
acc
```

Out[42]:
```
0.7050396260922577
```

we are getting 0.7 accuracy with average fasttext embeddings. No improvement. Let us try tfidf weighted fasttext

tfidf weighted fasttext embeddings

sentence1

In [72]:
```
#creating tfidf vecotrizer for sentence1
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer_s1 = TfidfVectorizer(ngram_range=(1, 1))
```

In [73]:
```
#fitting on train data
tfidf_vectorizer_s1.fit(train_df['sentence1_preprocessed'])
```

Out[73]:
```
TfidfVectorizer()
```

In [74]:
```
#transforming train-test-val data
train_s1_tfidf = tfidf_vectorizer_s1.transform(train_df['sentence1_preproces
val_s1_tfidf = tfidf_vectorizer_s1.transform(val_df['sentence1_preprocessed'
test_s1_tfidf = tfidf_vectorizer_s1.transform(test_df['sentence1_preprocesse
```

In [75]:
```
len(tfidf_vectorizer_s1.get_feature_names())
```

Out[75]:
```
17960
```

In [82]:
```
# creating a dic with keys as tokens and values and idf values
idf_dict_s1 = dict(zip(tfidf_vectorizer_s1.get_feature_names(), tfidf_vector
```

In [138]:
```
def tfidf_weighted_fasttext(sentences, ft, idf_dict):

    """
    Given a list of sentences, the function returns
    a 2d array where each row is tf-idf weighted fasttext
    vector of each sentence.
    args:
    sentences: list of sentences
    ft: a fasttext model
    idf_dict: a dictionary with tokens as keys and idf values and values

    """

    from collections import Counter
    sentences_tfidf_fasttext = list()
    vocab = list(idf_dict.keys())
    for sent in sentences:
        words = sent.split(' ')
        tf_dict = dict(Counter(words))
        tf_idf_sum = 0
        doc_vector = np.zeros(300)
        for word in words:
            if word in vocab:
                fast_text = ft.get_word_vector(word)
                idf = idf_dict[word]
                tf = tf_dict[word]
                tf_idf = tf*idf
                weighted_word_vec = fast_text*tf_idf
                tf_idf_sum = tf_idf_sum + tf_idf
                doc_vector = doc_vector + weighted_word_vec

        if tf_idf_sum != 0:
            doc_vector = doc_vector/tf_idf_sum

        sentences_tfidf_fasttext.append(doc_vector)

    return np.vstack(sentences_tfidf_fasttext)


```

In [139]:
```
sentences = train_df['sentence1_preprocessed'][0:5].values
v = TfidfVectorizer(ngram_range=(1,1))
v.fit(sentences)
idf_dict = dict(zip(v.get_feature_names(), v.idf_))
```

In [140]:
```
result = tfidf_weighted_fasttext(sentences, ft, idf_dict)
```

In [141]:
```
result.shape
```

Out[141]: (5, 300)

Type *Markdown* and LaTeX: α^2

train sentence1 tfidf weighted fasttext

Type *Markdown* and LaTeX: α^2

In [142]:
```
sentences = train_df['sentence1_preprocessed'].values
train_s1_tfidf_fasttext = tfidf_weighted_fasttext(sentences, ft, idf_dict_s1
```

In [143]:
```
train_s1_tfidf_fasttext.shape
```

Out[143]: (549361, 300)

validation sentence1 tfidf weighted fasttext

In [144]:
```
sentences = val_df['sentence1_preprocessed'].values
val_s1_tfidf_fasttext = tfidf_weighted_fasttext(sentences, ft, idf_dict_s1)
```

In [145]:
```
val_s1_tfidf_fasttext.shape
```

Out[145]: (9842, 300)

test sentence1 tfidf weighted fasttext

In [146]:
```
sentences = test_df['sentence1_preprocessed'].values
test_s1_tfidf_fasttext = tfidf_weighted_fasttext(sentences, ft, idf_dict_s1)
```

In [147]:
```
test_s1_tfidf_fasttext.shape
```

Out[147]: (9824, 300)

sentence2

In [148]:
```
#creating tfidf vecotrizer for sentence2
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer_s2 = TfidfVectorizer(ngram_range=(1, 1))
```

In [149]:
```
#fitting on train data
tfidf_vectorizer_s2.fit(train_df['sentence2_preprocessed'])
```

Out[149]: TfidfVectorizer()

In [150]:
```
len(tfidf_vectorizer_s2.get_feature_names())
```

Out[150]: 30038

In [151]:
```
# creating a dic with keys as tokens and values and idf values
idf_dict_s2 = dict(zip(tfidf_vectorizer_s2.get_feature_names(), tfidf_vector
```

train sentence2 tfidf weighted fasttext

In [153]:
```
sentences = train_df['sentence2_preprocessed'].values
train_s2_tfidf_fasttext = tfidf_weighted_fasttext(sentences, ft, idf_dict_s2
```

In [154]:
```
train_s2_tfidf_fasttext.shape
```

Out[154]: (549361, 300)

validation sentence2 tfidf weighted fasttext

In [155]:
```
sentences = val_df['sentence2_preprocessed'].values
val_s2_tfidf_fasttext = tfidf_weighted_fasttext(sentences, ft, idf_dict_s2)
```

In [157]:
```
val_s2_tfidf_fasttext.shape
```

Out[157]: (9842, 300)

test sentence2 tfidf weighted fasttext

In [158]:
```
sentences = test_df['sentence2_preprocessed'].values
test_s2_tfidf_fasttext = tfidf_weighted_fasttext(sentences, ft, idf_dict_s2)
```

In [159]:
```
test_s2_tfidf_fasttext.shape
```

Out[159]: (9824, 300)

In [160]:
```
#saving the matrices
np.save('train_s1_tfidf_fasttext.npy', train_s1_tfidf_fasttext)
np.save('train_s2_tfidf_fasttext.npy', train_s2_tfidf_fasttext)
np.save('val_s1_tfidf_fasttext.npy', val_s1_tfidf_fasttext)
np.save('val_s2_tfidf_fasttext.npy', val_s2_tfidf_fasttext)
np.save('test_s1_tfidf_fasttext.npy', test_s1_tfidf_fasttext)
np.save('test_s2_tfidf_fasttext.npy', test_s2_tfidf_fasttext)
```

In [161]:
```
# loading
train_s1_tfidf_fasttext = np.load('train_s1_tfidf_fasttext.npy', mmap_mode='
train_s2_tfidf_fasttext = np.load('train_s2_tfidf_fasttext.npy', mmap_mode='
val_s1_tfidf_fasttext = np.load('val_s1_tfidf_fasttext.npy', mmap_mode='r')
val_s2_tfidf_fasttext = np.load('val_s2_tfidf_fasttext.npy', mmap_mode='r')
test_s1_tfidf_fasttext = np.load('test_s1_tfidf_fasttext.npy', mmap_mode='r'
test_s2_tfidf_fasttext = np.load('test_s2_tfidf_fasttext.npy', mmap_mode='r'
```

In [164]:
```
X_train = train_df[to_keep]
y_train = train_df['gold_label']
X_val = val_df[to_keep]
y_val = val_df['gold_label']
x_train = np.hstack((X_train, train_s1_tfidf_fasttext, train_s2_tfidf_fastte
x_val = np.hstack((X_val, val_s1_tfidf_fasttext, val_s2_tfidf_fasttext))
```

In [ ]:
```
y_train.replace(to_replace = ['neutral', 'contradiction', 'entailment'] , va
y_val.replace(to_replace = ['neutral', 'contradiction', 'entailment'] , valu
```

In [165]:
```
import xgboost as xgb
xgbc = xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel
              colsample_bynode=1, colsample_bytree=1, enable_categorical=Fal
              gamma=0.1, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.1, max_delta_step=
              max_depth=12, min_child_weight=1, missing=np.nan,
              monotone_constraints='()', n_estimators=200, n_jobs=16,
              num_parallel_tree=1, objective='multi:softprob', predictor='au
              random_state=0, reg_alpha=0, reg_lambda=100,
              scale_pos_weight=None, subsample=1, tree_method='exact',
              validate_parameters=1, verbosity=None, use_label_encoder=False
```

In [166]:
```
xgbc.fit(x_train, y_train)
```

```
[18:22:41] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'multi:softprob' was changed from 'merror' to 'mloglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[166]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
              gamma=0.1, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.1, max_delta_step=0,
              max_depth=12, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=200, n_jobs=16,
              num_parallel_tree=1, objective='multi:softprob', predictor='aut
o',
              random_state=0, reg_alpha=0, reg_lambda=100,
              scale_pos_weight=None, subsample=1, tree_method='exact',
              use_label_encoder=False, validate_parameters=1, verbosity=None)
```

In [167]:
```
val_pred = xgbc.predict(x_val)
from sklearn.metrics import accuracy_score
acc = accuracy_score(y_val, val_pred)
acc
```

Out[167]:
```
0.7159114001219264
```

little improvement after taking tfidf weighted fasttext embeddings

In [4]:
```
from prettytable import PrettyTable
myTable = PrettyTable(["Feature set", "Model", "accuracy"])
myTable.add_row(["Only extracted features", "RandomForest", "0.58"])
myTable.add_row(["Extracted features + avg GloVe embeddings", "RandomForest"
myTable.add_row(["Extracted features + avg GloVe embeddings", "XGBoost", "0.
myTable.add_row(["Extracted features + avg fasttext embeddings", "XGBoost",
myTable.add_row(["Extracted features + tfidf weighted fasttext embeddings",
print('Summary of classical machine learning aproaches:')
print(myTable)
```

```
Summary of classical machine learning aproaches:
+-----------------------------------------------------------+--------------+-----
-----+
|                        Feature set                        |    Model     | accu
racy |
+-----------------------------------------------------------+--------------+-----
-----+
|                  Only extracted features                  | RandomForest |   0.
58   |
|        Extracted features + avg GloVe embeddings          | RandomForest |   0.
66   |
|        Extracted features + avg GloVe embeddings          |   XGBoost    |   0.
70   |
|       Extracted features + avg fasttext embeddings        |   XGBoost    |   0.
70   |
| Extracted features + tfidf weighted fasttext embeddings |   XGBoost    |   0.
71   |
+-----------------------------------------------------------+--------------+-----
-----+
```

Moving towards Deep Learning

In [3]:
```
from tensorflow.keras.layers import Input, Dense, concatenate, BatchNormaliz
from tensorflow.keras.models import Model
from tensorflow.keras.utils import plot_model
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.regularizers import L1, L2
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
```

In [4]:
```
import tensorflow as tf
tf.config.experimental.allow_growth = True

```

In [5]:
```
# loading the dataframes in which features have already been extracted
train_df = pd.read_csv("train_df.csv")
val_df = pd.read_csv("val_df.csv")
test_df = pd.read_csv("test_df.csv")
```

In [6]:
```
# loading avg glove vectors
x_train_s1_tfidfw2v = np.load('x_train_s1_tfidfw2v.npy', mmap_mode='r')
x_train_s2_tfidfw2v = np.load('x_train_s2_tfidfw2v.npy', mmap_mode='r')
x_val_s1_tfidfw2v = np.load('x_val_s1_tfidfw2v.npy', mmap_mode='r')
x_val_s2_tfidfw2v = np.load('x_val_s2_tfidfw2v.npy', mmap_mode='r')
# x_test_s1_tfidfw2v = np.load('x_test_s1_tfidfw2v.npy', mmap_mode='r')
# x_test_s2_tfidfw2v = np.load('x_test_s2_tfidfw2v.npy', mmap_mode='r')
```

In [6]:
```
# list of columns to be kept from loaded dataframes
to_keep = ['log_bleu_score', 'sum_polarity', 'diff_subjectivity',
       'cosine_similarity', 'tfidf_cosine_sim', 'diff_num_words', 's1_CC',
       's1_IN','s1_PRP', 's1_PRP$', 's1_RB', 's1_RBR', 's1_RBS', 's1_UH', 's
       's2_PRP$', 's2_RB', 's2_RBR', 's2_RBS', 's2_UH', 'ratio_noun', 'ratio
       'diff_adjective', 'levenshtein_dist', 'tfidf_w2v_cosine',
       'tfidf_cosine_sim_ngram']
```

In [7]:
```
#selecting required columns
X_train_1 = train_df[to_keep]
y_train = train_df['gold_label']
X_val_1 = val_df[to_keep]
y_val = val_df['gold_label']
```

In [8]:
```
#converting target categories to numbers: neutral:0, contradiction:1, entail
y_train.replace(to_replace = ['neutral', 'contradiction', 'entailment'] , va
y_val.replace(to_replace = ['neutral', 'contradiction', 'entailment'] , valu
```

In [9]:
```
#one hot encoding target
y_train_enc = to_categorical(y_train)
y_val_enc = to_categorical(y_val)
```

model0: considering only avg glove embeddings

In [12]:
```
input2 = Input(shape=(300,)) #for sentence1 embeddings
input3 = Input(shape=(300,)) #for sentence2 embeddings
merged = concatenate([input2, input3]) #merging all inputs
norm1 = BatchNormalization()(merged)
dense1 = Dense(units = 50, activation = 'relu')(norm1)
dense2 = Dense(units = 40, activation = 'relu')(dense1)
dense3 = Dense(units = 30, activation = 'relu')(dense2)
norm2 = BatchNormalization()(dense3)
dense4 = Dense(units = 20, activation = 'relu')(norm2)
dense5 = Dense(units = 10, activation = 'relu')(dense4)
output = Dense(units = 3, activation = 'softmax')(dense5)
model0 = Model(inputs = [input2, input3], outputs = output)
```

In [13]:
```
model0.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metric
```

In [14]:
```
model0.fit([x_train_s1_tfidfw2v, x_train_s2_tfidfw2v], y_train_enc, epochs =
           validation_data = ([x_val_s1_tfidfw2v, x_val_s2_tfidfw2v], y_val_
```

```
Epoch 1/50
4292/4292 [==============================] - 20s 4ms/step - loss: 0.8394 - ac
curacy: 0.6137 - val_loss: 0.7503 - val_accuracy: 0.6715
Epoch 2/50
4292/4292 [==============================] - 18s 4ms/step - loss: 0.7406 - ac
curacy: 0.6768 - val_loss: 0.7123 - val_accuracy: 0.6944
Epoch 3/50
4292/4292 [==============================] - 18s 4ms/step - loss: 0.7113 - ac
curacy: 0.6929 - val_loss: 0.6980 - val_accuracy: 0.7017
Epoch 4/50
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6938 - ac
curacy: 0.7021 - val_loss: 0.6847 - val_accuracy: 0.7038
Epoch 5/50
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6816 - ac
curacy: 0.7092 - val_loss: 0.6860 - val_accuracy: 0.7074
Epoch 6/50
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6731 - ac
curacy: 0.7139 - val_loss: 0.6701 - val_accuracy: 0.7166
Epoch 7/50
```

In [15]:
```
model0.fit([x_train_s1_tfidfw2v, x_train_s2_tfidfw2v], y_train_enc, epochs =
           validation_data = ([x_val_s1_tfidfw2v, x_val_s2_tfidfw2v], y_val_
```

```
Epoch 1/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.6099 - accu
racy: 0.7463 - val_loss: 0.6453 - val_accuracy: 0.7322
Epoch 2/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.6092 - accu
racy: 0.7466 - val_loss: 0.6461 - val_accuracy: 0.7303
Epoch 3/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.6093 - accu
racy: 0.7460 - val_loss: 0.6455 - val_accuracy: 0.7280
Epoch 4/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.6088 - accu
racy: 0.7466 - val_loss: 0.6423 - val_accuracy: 0.7314
Epoch 5/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.6084 - accu
racy: 0.7471 - val_loss: 0.6435 - val_accuracy: 0.7311
Epoch 6/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.6082 - accu
racy: 0.7470 - val_loss: 0.6428 - val_accuracy: 0.7304
Epoch 7/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.6078 - accu
racy: 0.7476 - val_loss: 0.6431 - val_accuracy: 0.7306
Epoch 8/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.6075 - accu
racy: 0.7472 - val_loss: 0.6444 - val_accuracy: 0.7309
Epoch 9/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.6067 - accu
racy: 0.7472 - val_loss: 0.6431 - val_accuracy: 0.7307
Epoch 10/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.6063 - accu
racy: 0.7478 - val_loss: 0.6485 - val_accuracy: 0.7321
Epoch 11/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.6063 - accu
racy: 0.7479 - val_loss: 0.6466 - val_accuracy: 0.7306
Epoch 12/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6058 - accu
racy: 0.7485 - val_loss: 0.6424 - val_accuracy: 0.7342
Epoch 13/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6058 - accu
racy: 0.7484 - val_loss: 0.6394 - val_accuracy: 0.7344
Epoch 14/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.6060 - accu
racy: 0.7478 - val_loss: 0.6458 - val_accuracy: 0.7332
Epoch 15/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6053 - accu
racy: 0.7480 - val_loss: 0.6461 - val_accuracy: 0.7303
Epoch 16/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.6045 - accu
racy: 0.7482 - val_loss: 0.6459 - val_accuracy: 0.7296
Epoch 17/20
4292/4292 [==============================] - 19s 4ms/step - loss: 0.6047 - accu
racy: 0.7484 - val_loss: 0.6429 - val_accuracy: 0.7326
Epoch 18/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.6044 - accu
racy: 0.7485 - val_loss: 0.6416 - val_accuracy: 0.7322
```

```
Epoch 19/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.6040 - accu
racy: 0.7491 - val_loss: 0.6417 - val_accuracy: 0.7320
Epoch 20/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.6041 - accu
racy: 0.7491 - val_loss: 0.6388 - val_accuracy: 0.7357
```

Out[15]: <keras.callbacks.History at 0x213f4fdd780>

In [16]:
```
model0.fit([x_train_s1_tfidfw2v, x_train_s2_tfidfw2v], y_train_enc, epochs =
           validation_data = ([x_val_s1_tfidfw2v, x_val_s2_tfidfw2v], y_val_
```

```
Epoch 1/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6038 - accu
racy: 0.7489 - val_loss: 0.6420 - val_accuracy: 0.7335
Epoch 2/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6033 - accu
racy: 0.7493 - val_loss: 0.6446 - val_accuracy: 0.7331
Epoch 3/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6033 - accu
racy: 0.7487 - val_loss: 0.6431 - val_accuracy: 0.7311
Epoch 4/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6030 - accu
racy: 0.7497 - val_loss: 0.6425 - val_accuracy: 0.7337
Epoch 5/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6028 - accu
racy: 0.7500 - val_loss: 0.6430 - val_accuracy: 0.7315
Epoch 6/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6025 - accu
racy: 0.7494 - val_loss: 0.6432 - val_accuracy: 0.7283
Epoch 7/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6023 - accu
racy: 0.7499 - val_loss: 0.6439 - val_accuracy: 0.7282
Epoch 8/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6021 - accu
racy: 0.7493 - val_loss: 0.6427 - val_accuracy: 0.7312
Epoch 9/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6015 - accu
racy: 0.7500 - val_loss: 0.6424 - val_accuracy: 0.7322
Epoch 10/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6013 - accu
racy: 0.7507 - val_loss: 0.6465 - val_accuracy: 0.7327
Epoch 11/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6012 - accu
racy: 0.7502 - val_loss: 0.6449 - val_accuracy: 0.7331
Epoch 12/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6008 - accu
racy: 0.7506 - val_loss: 0.6412 - val_accuracy: 0.7324
Epoch 13/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6008 - accu
racy: 0.7503 - val_loss: 0.6396 - val_accuracy: 0.7313
Epoch 14/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6012 - accu
racy: 0.7501 - val_loss: 0.6441 - val_accuracy: 0.7315
Epoch 15/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6006 - accu
racy: 0.7507 - val_loss: 0.6449 - val_accuracy: 0.7316
Epoch 16/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.5999 - accu
racy: 0.7500 - val_loss: 0.6460 - val_accuracy: 0.7312
Epoch 17/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6002 - accu
racy: 0.7505 - val_loss: 0.6446 - val_accuracy: 0.7342
Epoch 18/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6001 - accu
racy: 0.7502 - val_loss: 0.6418 - val_accuracy: 0.7337
```

```
Epoch 19/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.5997 - accu
racy: 0.7510 - val_loss: 0.6398 - val_accuracy: 0.7308
Epoch 20/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.5997 - accu
racy: 0.7509 - val_loss: 0.6390 - val_accuracy: 0.7351
```

Out[16]: <keras.callbacks.History at 0x213f4fd0ac8>

Model0_a

In [31]:
```
input2 = Input(shape=(300,)) #for sentence1 embeddings
norm2 = BatchNormalization()(input2)
input3 = Input(shape=(300,)) #for sentence2 embeddings
norm3 = BatchNormalization()(input3)
merged = concatenate([norm2, norm3]) #merging all inputs
#norm1 = BatchNormalization()(merged)
dense1 = Dense(units = 100, activation = 'relu', kernel_regularizer = L1(0.0
drop1 = Dropout(0.01)(dense1)
dense2 = Dense(units = 100, activation = 'relu', kernel_regularizer = L1(0.0
drop2 = Dropout(0.01)(dense2)
dense3 = Dense(units = 100, activation = 'relu', kernel_regularizer = L1(0.0
dense4 = Dense(units = 50, activation = 'relu')(dense3)
output = Dense(units = 3, activation = 'softmax')(dense4)
model0_a = Model(inputs = [input2, input3], outputs = output)
```

In [32]:
```
model0_a.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metr
```

In [34]:
```
tf.debugging.set_log_device_placement(True)

# Place tensors on the CPU
with tf.device('/CPU:0'):
    model0_a.fit([x_train_s1_tfidfw2v, x_train_s2_tfidfw2v], y_train_enc, ep
           validation_data = ([x_val_s1_tfidfw2v, x_val_s2_tfidfw2v], y_val_
```

```
Epoch 1/100
1073/1073 [==============================] - 11s 8ms/step - loss: 0.8531 - ac
curacy: 0.6338 - val_loss: 0.7762 - val_accuracy: 0.6831
Epoch 2/100
1073/1073 [==============================] - 8s 7ms/step - loss: 0.7543 - acc
uracy: 0.6929 - val_loss: 0.7307 - val_accuracy: 0.7064
Epoch 3/100
1073/1073 [==============================] - 8s 7ms/step - loss: 0.7202 - acc
uracy: 0.7111 - val_loss: 0.7142 - val_accuracy: 0.7123
Epoch 4/100
1073/1073 [==============================] - 8s 7ms/step - loss: 0.6995 - acc
uracy: 0.7226 - val_loss: 0.6951 - val_accuracy: 0.7253
Epoch 5/100
1073/1073 [==============================] - 8s 7ms/step - loss: 0.6845 - acc
uracy: 0.7308 - val_loss: 0.6869 - val_accuracy: 0.7301
Epoch 6/100
1073/1073 [==============================] - 8s 8ms/step - loss: 0.6736 - acc
uracy: 0.7368 - val_loss: 0.6803 - val_accuracy: 0.7350
Epoch 7/100
```

model1: considering extracted features+avg glove embeddings

In [16]:
```
input1 = Input(shape=(29,)) #for extracted features
input2 = Input(shape=(300,)) #for sentence1 embeddings
input3 = Input(shape=(300,)) #for sentence2 embeddings
merged = concatenate([input1, input2, input3]) #merging all inputs
norm1 = BatchNormalization()(merged)
dense1 = Dense(units = 50, activation = 'relu')(norm1)
dense2 = Dense(units = 40, activation = 'relu')(dense1)
dense3 = Dense(units = 30, activation = 'relu')(dense2)
norm2 = BatchNormalization()(dense3)
dense4 = Dense(units = 20, activation = 'relu')(norm2)
dense5 = Dense(units = 10, activation = 'relu')(dense4)
output = Dense(units = 3, activation = 'softmax')(dense5)
model1 = Model(inputs = [input1, input2, input3], outputs = output)
```

In [17]:
```
model1.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metric
```

In [18]:
```
model1.fit([X_train_1, x_train_s1_tfidfw2v, x_train_s2_tfidfw2v], y_train_en
           validation_data = ([X_val_1, x_val_s1_tfidfw2v, x_val_s2_tfidfw2v
```

```
Epoch 1/50
4292/4292 [==============================] - 23s 5ms/step - loss: 0.7971 - ac
curacy: 0.6417 - val_loss: 0.7009 - val_accuracy: 0.6971
Epoch 2/50
4292/4292 [==============================] - 17s 4ms/step - loss: 0.7027 - ac
curacy: 0.6977 - val_loss: 0.6650 - val_accuracy: 0.7143
Epoch 3/50
4292/4292 [==============================] - 17s 4ms/step - loss: 0.6749 - ac
curacy: 0.7125 - val_loss: 0.6568 - val_accuracy: 0.7214
Epoch 4/50
4292/4292 [==============================] - 18s 4ms/step - loss: 0.6584 - ac
curacy: 0.7211 - val_loss: 0.6435 - val_accuracy: 0.7313
Epoch 5/50
4292/4292 [==============================] - 17s 4ms/step - loss: 0.6469 - ac
curacy: 0.7266 - val_loss: 0.6365 - val_accuracy: 0.7297
Epoch 6/50
4292/4292 [==============================] - 17s 4ms/step - loss: 0.6388 - ac
curacy: 0.7309 - val_loss: 0.6262 - val_accuracy: 0.7379
Epoch 7/50
```

In [19]:
```
model1.fit([X_train_1, x_train_s1_tfidfw2v, x_train_s2_tfidfw2v], y_train_en
           validation_data = ([X_val_1, x_val_s1_tfidfw2v, x_val_s2_tfidfw2v
```

```
Epoch 1/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.5752 - accu
racy: 0.7624 - val_loss: 0.5992 - val_accuracy: 0.7520
Epoch 2/20
4292/4292 [==============================] - 19s 4ms/step - loss: 0.5748 - accu
racy: 0.7624 - val_loss: 0.5974 - val_accuracy: 0.7552
Epoch 3/20
4292/4292 [==============================] - 18s 4ms/step - loss: 0.5749 - accu
racy: 0.7620 - val_loss: 0.5993 - val_accuracy: 0.7522
Epoch 4/20
4292/4292 [==============================] - 19s 4ms/step - loss: 0.5744 - accu
racy: 0.7625 - val_loss: 0.5978 - val_accuracy: 0.7555
Epoch 5/20
4292/4292 [==============================] - 19s 4ms/step - loss: 0.5738 - accu
racy: 0.7626 - val_loss: 0.5985 - val_accuracy: 0.7526
Epoch 6/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.5733 - accu
racy: 0.7629 - val_loss: 0.6009 - val_accuracy: 0.7555
Epoch 7/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.5731 - accu
racy: 0.7636 - val_loss: 0.6005 - val_accuracy: 0.7534
Epoch 8/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.5728 - accu
racy: 0.7630 - val_loss: 0.6010 - val_accuracy: 0.7510
Epoch 9/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.5721 - accu
racy: 0.7635 - val_loss: 0.5982 - val_accuracy: 0.7551
Epoch 10/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.5719 - accu
racy: 0.7642 - val_loss: 0.6077 - val_accuracy: 0.7534
Epoch 11/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.5711 - accu
racy: 0.7644 - val_loss: 0.6005 - val_accuracy: 0.7516
Epoch 12/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.5715 - accu
racy: 0.7639 - val_loss: 0.5984 - val_accuracy: 0.7543
Epoch 13/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.5708 - accu
racy: 0.7646 - val_loss: 0.5962 - val_accuracy: 0.7537
Epoch 14/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.5712 - accu
racy: 0.7648 - val_loss: 0.6000 - val_accuracy: 0.7539
Epoch 15/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.5702 - accu
racy: 0.7650 - val_loss: 0.6015 - val_accuracy: 0.7525
Epoch 16/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.5701 - accu
racy: 0.7648 - val_loss: 0.6028 - val_accuracy: 0.7577
Epoch 17/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.5699 - accu
racy: 0.7645 - val_loss: 0.5993 - val_accuracy: 0.7561
Epoch 18/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.5694 - accu
racy: 0.7647 - val_loss: 0.5994 - val_accuracy: 0.7552
```

```
Epoch 19/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.5693 - accu
racy: 0.7647 - val_loss: 0.5987 - val_accuracy: 0.7546
Epoch 20/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.5693 - accu
racy: 0.7645 - val_loss: 0.5980 - val_accuracy: 0.7571
```

Out[19]: <keras.callbacks.History at 0x2aca478d860>

model1_a

In [61]:
```
input1 = Input(shape=(29,)) #for extracted features
input2 = Input(shape=(300,)) #for sentence1 embeddings
input3 = Input(shape=(300,)) #for sentence2 embeddings
merged = concatenate([input1, input2, input3]) #merging all inputs
norm1 = BatchNormalization()(merged)
dense1 = Dense(units = 50, activation = 'relu')(merged)
dense2 = Dense(units = 50, activation = 'relu')(dense1)
dense3 = Dense(units = 50, activation = 'relu')(dense2)
output = Dense(units = 3, activation = 'softmax')(dense3)
model1_a = Model(inputs = [input1, input2, input3], outputs = output)
```

In [62]:
```
model1_a.compile(optimizer = Adam(0.001), loss = 'categorical_crossentropy',
tf.debugging.set_log_device_placement(True)

# Place tensors on the CPU
with tf.device('/CPU:0'):
    model1_a.fit([X_train_1, x_train_s1_tfidfw2v, x_train_s2_tfidfw2v], y_tr
           validation_data = ([X_val_1, x_val_s1_tfidfw2v, x_val_s2_tfidfw2v
```

```
Epoch 1/100
1073/1073 [==============================] - 4s 4ms/step - loss: 0.9425 - acc
uracy: 0.5589 - val_loss: 0.8820 - val_accuracy: 0.5935
Epoch 2/100
1073/1073 [==============================] - 4s 4ms/step - loss: 0.8529 - acc
uracy: 0.6141 - val_loss: 0.8227 - val_accuracy: 0.6304
Epoch 3/100
1073/1073 [==============================] - 4s 4ms/step - loss: 0.8321 - acc
uracy: 0.6260 - val_loss: 0.8034 - val_accuracy: 0.6467
Epoch 4/100
1073/1073 [==============================] - 4s 4ms/step - loss: 0.8176 - acc
uracy: 0.6335 - val_loss: 0.7844 - val_accuracy: 0.6520
Epoch 5/100
1073/1073 [==============================] - 4s 4ms/step - loss: 0.8003 - acc
uracy: 0.6437 - val_loss: 0.8355 - val_accuracy: 0.6159
Epoch 6/100
1073/1073 [==============================] - 4s 4ms/step - loss: 0.7881 - acc
uracy: 0.6508 - val_loss: 0.7705 - val_accuracy: 0.6612
Epoch 7/100
```

In [63]:
```
with tf.device('/CPU:0'):
    model1_a.fit([X_train_1, x_train_s1_tfidfw2v, x_train_s2_tfidfw2v], y_tr
           validation_data = ([X_val_1, x_val_s1_tfidfw2v, x_val_s2_tfidfw2v
```

```
Epoch 1/50
1073/1073 [==============================] - 4s 4ms/step - loss: 0.6231 - acc
uracy: 0.7374 - val_loss: 0.6644 - val_accuracy: 0.7195
Epoch 2/50
1073/1073 [==============================] - 4s 4ms/step - loss: 0.6219 - acc
uracy: 0.7382 - val_loss: 0.6702 - val_accuracy: 0.7167
Epoch 3/50
1073/1073 [==============================] - 4s 4ms/step - loss: 0.6216 - acc
uracy: 0.7384 - val_loss: 0.6611 - val_accuracy: 0.7228
Epoch 4/50
1073/1073 [==============================] - 4s 4ms/step - loss: 0.6206 - acc
uracy: 0.7394 - val_loss: 0.6817 - val_accuracy: 0.7150
Epoch 5/50
1073/1073 [==============================] - 4s 4ms/step - loss: 0.6218 - acc
uracy: 0.7385 - val_loss: 0.6624 - val_accuracy: 0.7214
Epoch 6/50
1073/1073 [==============================] - 4s 4ms/step - loss: 0.6214 - acc
uracy: 0.7385 - val_loss: 0.6626 - val_accuracy: 0.7198
Epoch 7/50
```

model1_b

In [11]:
```
input1 = Input(shape=(29,)) #for extracted features
input2 = Input(shape=(300,)) #for sentence1 embeddings
input3 = Input(shape=(300,)) #for sentence2 embeddings
merged = concatenate([input1, input2, input3]) #merging all inputs
norm1 = BatchNormalization()(merged)
dense1 = Dense(units = 70, activation = 'relu')(norm1)
dense2 = Dense(units = 60, activation = 'relu')(dense1)
dense3 = Dense(units = 50, activation = 'relu')(dense2)
dense4 = Dense(units = 40, activation = 'relu')(dense3)
dense5 = Dense(units = 30, activation = 'relu')(dense4)
dense6 = Dense(units = 20, activation = 'relu')(dense5)
dense7 = Dense(units = 10, activation = 'relu')(dense6)
output = Dense(units = 3, activation = 'softmax')(dense7)
model1_b = Model(inputs = [input1, input2, input3], outputs = output)
```

In [12]:
```
model1_b.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metr
model1_b.fit([X_train_1, x_train_s1_tfidfw2v, x_train_s2_tfidfw2v], y_train_
          validation_data = ([X_val_1, x_val_s1_tfidfw2v, x_val_s2_tfidfw2v
```

```
Epoch 1/100
4292/4292 [==============================] - 31s 7ms/step - loss: 0.7687 - ac
curacy: 0.6592 - val_loss: 0.6857 - val_accuracy: 0.7053
Epoch 2/100
4292/4292 [==============================] - 25s 6ms/step - loss: 0.6825 - ac
curacy: 0.7081 - val_loss: 0.6555 - val_accuracy: 0.7228
Epoch 3/100
4292/4292 [==============================] - 22s 5ms/step - loss: 0.6551 - ac
curacy: 0.7228 - val_loss: 0.6410 - val_accuracy: 0.7342
Epoch 4/100
4292/4292 [==============================] - 21s 5ms/step - loss: 0.6375 - ac
curacy: 0.7314 - val_loss: 0.6333 - val_accuracy: 0.7318
Epoch 5/100
4292/4292 [==============================] - 23s 5ms/step - loss: 0.6251 - ac
curacy: 0.7381 - val_loss: 0.6214 - val_accuracy: 0.7413
Epoch 6/100
4292/4292 [==============================] - 23s 5ms/step - loss: 0.6158 - ac
curacy: 0.7428 - val_loss: 0.6189 - val_accuracy: 0.7393
Epoch 7/100
```

In [13]:
```
from keras import backend as K
K.set_value(model1_b.optimizer.learning_rate, 0.0001)
```

In [14]:
```
model1_b.fit([X_train_1, x_train_s1_tfidfw2v, x_train_s2_tfidfw2v], y_train_
          validation_data = ([X_val_1, x_val_s1_tfidfw2v, x_val_s2_tfidfw2v
```

```
Epoch 1/20
4292/4292 [==============================] - 17s 4ms/step - loss: 0.5067 - accu
racy: 0.7943 - val_loss: 0.6106 - val_accuracy: 0.7558
Epoch 2/20
4292/4292 [==============================] - 15s 4ms/step - loss: 0.5030 - accu
racy: 0.7957 - val_loss: 0.6165 - val_accuracy: 0.7540
Epoch 3/20
4292/4292 [==============================] - 15s 4ms/step - loss: 0.5018 - accu
racy: 0.7963 - val_loss: 0.6167 - val_accuracy: 0.7544
Epoch 4/20
4292/4292 [==============================] - 15s 4ms/step - loss: 0.5012 - accu
racy: 0.7962 - val_loss: 0.6164 - val_accuracy: 0.7552
Epoch 5/20
4292/4292 [==============================] - 15s 4ms/step - loss: 0.5002 - accu
racy: 0.7971 - val_loss: 0.6217 - val_accuracy: 0.7525
Epoch 6/20
4292/4292 [==============================] - 15s 4ms/step - loss: 0.5000 - accu
racy: 0.7970 - val_loss: 0.6203 - val_accuracy: 0.7544
Epoch 7/20
4292/4292 [==============================] - 16s 4ms/step - loss: 0.4994 - accu
racy: 0.7976 - val_loss: 0.6193 - val_accuracy: 0.7537
Epoch 8/20
4292/4292 [==============================] - 16s 4ms/step - loss: 0.4998 - accu
racy: 0.7973 - val_loss: 0.6187 - val_accuracy: 0.7537
Epoch 9/20
4292/4292 [==============================] - 15s 4ms/step - loss: 0.4992 - accu
racy: 0.7976 - val_loss: 0.6171 - val_accuracy: 0.7535
Epoch 10/20
4292/4292 [==============================] - 16s 4ms/step - loss: 0.4991 - accu
racy: 0.7974 - val_loss: 0.6229 - val_accuracy: 0.7535
Epoch 11/20
4292/4292 [==============================] - 15s 4ms/step - loss: 0.4983 - accu
racy: 0.7981 - val_loss: 0.6209 - val_accuracy: 0.7542
Epoch 12/20
4292/4292 [==============================] - 15s 4ms/step - loss: 0.4983 - accu
racy: 0.7979 - val_loss: 0.6199 - val_accuracy: 0.7543
Epoch 13/20
4292/4292 [==============================] - 16s 4ms/step - loss: 0.4985 - accu
racy: 0.7977 - val_loss: 0.6202 - val_accuracy: 0.7552
Epoch 14/20
4292/4292 [==============================] - 15s 4ms/step - loss: 0.4984 - accu
racy: 0.7979 - val_loss: 0.6233 - val_accuracy: 0.7540
Epoch 15/20
4292/4292 [==============================] - 16s 4ms/step - loss: 0.4979 - accu
racy: 0.7982 - val_loss: 0.6217 - val_accuracy: 0.7536
Epoch 16/20
4292/4292 [==============================] - 15s 4ms/step - loss: 0.4979 - accu
racy: 0.7979 - val_loss: 0.6231 - val_accuracy: 0.7541
Epoch 17/20
4292/4292 [==============================] - 15s 4ms/step - loss: 0.4976 - accu
racy: 0.7983 - val_loss: 0.6224 - val_accuracy: 0.7537
Epoch 18/20
4292/4292 [==============================] - 15s 4ms/step - loss: 0.4977 - accu
racy: 0.7982 - val_loss: 0.6253 - val_accuracy: 0.7544
```

```
Epoch 19/20
4292/4292 [==============================] - 15s 4ms/step - loss: 0.4976 - accu
racy: 0.7981 - val_loss: 0.6231 - val_accuracy: 0.7531
Epoch 20/20
4292/4292 [==============================] - 15s 4ms/step - loss: 0.4978 - accu
racy: 0.7981 - val_loss: 0.6241 - val_accuracy: 0.7549
```

Out[14]: <keras.callbacks.History at 0x1d0a5d7e978>

Model2: using only average fasttext embeddings

In [14]:
```
x_train_s1_avg_fasttext = np.load('x_train_s1_avg_fasttext.npy', mmap_mode='
x_train_s2_avg_fasttext = np.load('x_train_s2_avg_fasttext.npy', mmap_mode='
x_val_s1_avg_fasttext = np.load('x_val_s1_avg_fasttext.npy', mmap_mode='r')
x_val_s2_avg_fasttext = np.load('x_val_s2_avg_fasttext.npy', mmap_mode='r')
x_test_s1_avg_fasttext = np.load('x_test_s1_avg_fasttext.npy', mmap_mode='r'
x_test_s2_avg_fasttext = np.load('x_test_s2_avg_fasttext.npy', mmap_mode='r'
```

In [16]:
```
input2 = Input(shape=(300,)) #for sentence1 embeddings
norm2 = BatchNormalization()(input2)
input3 = Input(shape=(300,)) #for sentence2 embeddings
norm3 = BatchNormalization()(input3)
merged = concatenate([norm2, norm3]) #merging all inputs
#norm1 = BatchNormalization()(merged)
dense1 = Dense(units = 100, activation = 'relu', kernel_regularizer = L1(0.0
drop1 = Dropout(0.01)(dense1)
dense2 = Dense(units = 100, activation = 'relu', kernel_regularizer = L1(0.0
drop2 = Dropout(0.01)(dense2)
dense3 = Dense(units = 100, activation = 'relu', kernel_regularizer = L1(0.0
dense4 = Dense(units = 50, activation = 'relu')(dense3)
output = Dense(units = 3, activation = 'softmax')(dense4)
model2_a = Model(inputs = [input2, input3], outputs = output)
```

In [18]:
```
model2_a.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metr

tf.debugging.set_log_device_placement(True)

# Place tensors on the CPU
with tf.device('/CPU:0'):
    model2_a.fit([x_train_s1_avg_fasttext, x_train_s2_avg_fasttext], y_train
           validation_data = ([x_val_s1_avg_fasttext, x_val_s2_avg_fasttext]
```

```
Epoch 1/100
4292/4292 [==============================] - 23s 5ms/step - loss: 0.8644 - ac
curacy: 0.6295 - val_loss: 0.7856 - val_accuracy: 0.6819
Epoch 2/100
4292/4292 [==============================] - 21s 5ms/step - loss: 0.7768 - ac
curacy: 0.6826 - val_loss: 0.7426 - val_accuracy: 0.7025
Epoch 3/100
4292/4292 [==============================] - 20s 5ms/step - loss: 0.7464 - ac
curacy: 0.7010 - val_loss: 0.7265 - val_accuracy: 0.7112
Epoch 4/100
4292/4292 [==============================] - 20s 5ms/step - loss: 0.7280 - ac
curacy: 0.7114 - val_loss: 0.7223 - val_accuracy: 0.7166
Epoch 5/100
4292/4292 [==============================] - 20s 5ms/step - loss: 0.7162 - ac
curacy: 0.7192 - val_loss: 0.7038 - val_accuracy: 0.7273
Epoch 6/100
4292/4292 [==============================] - 20s 5ms/step - loss: 0.7073 - ac
curacy: 0.7241 - val_loss: 0.7044 - val_accuracy: 0.7298
Epoch 7/100
```

In [19]:
```
from keras import backend as K
K.set_value(model2_a.optimizer.learning_rate, 0.0001)
```

In [20]:
```
with tf.device('/CPU:0'):
    model2_a.fit([x_train_s1_avg_fasttext, x_train_s2_avg_fasttext], y_train
           validation_data = ([x_val_s1_avg_fasttext, x_val_s2_avg_fasttext]
```

```
Epoch 1/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.6116 - accu
racy: 0.7845 - val_loss: 0.6761 - val_accuracy: 0.7562
Epoch 2/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.6013 - accu
racy: 0.7890 - val_loss: 0.6778 - val_accuracy: 0.7566
Epoch 3/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.5973 - accu
racy: 0.7901 - val_loss: 0.6782 - val_accuracy: 0.7568
Epoch 4/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.5947 - accu
racy: 0.7915 - val_loss: 0.6782 - val_accuracy: 0.7579
Epoch 5/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.5934 - accu
racy: 0.7912 - val_loss: 0.6788 - val_accuracy: 0.7576
Epoch 6/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.5918 - accu
racy: 0.7922 - val_loss: 0.6771 - val_accuracy: 0.7573
Epoch 7/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.5903 - accu
racy: 0.7930 - val_loss: 0.6789 - val_accuracy: 0.7590
Epoch 8/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.5897 - accu
racy: 0.7930 - val_loss: 0.6780 - val_accuracy: 0.7571
Epoch 9/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.5884 - accu
racy: 0.7937 - val_loss: 0.6797 - val_accuracy: 0.7588
Epoch 10/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.5873 - accu
racy: 0.7940 - val_loss: 0.6799 - val_accuracy: 0.7568
Epoch 11/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.5869 - accu
racy: 0.7941 - val_loss: 0.6811 - val_accuracy: 0.7574
Epoch 12/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.5861 - accu
racy: 0.7948 - val_loss: 0.6798 - val_accuracy: 0.7569
Epoch 13/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.5851 - accu
racy: 0.7951 - val_loss: 0.6813 - val_accuracy: 0.7559
Epoch 14/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.5850 - accu
racy: 0.7946 - val_loss: 0.6817 - val_accuracy: 0.7564
Epoch 15/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.5842 - accu
racy: 0.7948 - val_loss: 0.6841 - val_accuracy: 0.7552
Epoch 16/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.5839 - accu
racy: 0.7947 - val_loss: 0.6820 - val_accuracy: 0.7580
Epoch 17/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.5830 - accu
racy: 0.7954 - val_loss: 0.6804 - val_accuracy: 0.7576
Epoch 18/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.5827 - accu
```

```
racy: 0.7954 - val_loss: 0.6826 - val_accuracy: 0.7581
Epoch 19/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.5823 - accu
racy: 0.7958 - val_loss: 0.6818 - val_accuracy: 0.7580
Epoch 20/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.5820 - accu
racy: 0.7953 - val_loss: 0.6835 - val_accuracy: 0.7579
```

model3: tfidf weighted fasttext embeddings

In [17]:
```
x_train_s1_tfidf_fasttext = np.load('train_s1_tfidf_fasttext.npy', mmap_mode
x_train_s2_tfidf_fasttext = np.load('train_s2_tfidf_fasttext.npy', mmap_mode
x_val_s1_tfidf_fasttext = np.load('val_s1_tfidf_fasttext.npy', mmap_mode='r'
x_val_s2_tfidf_fasttext = np.load('val_s2_tfidf_fasttext.npy', mmap_mode='r'
x_test_s1_tfidf_fasttext = np.load('test_s1_tfidf_fasttext.npy', mmap_mode='
x_test_s2_tfidf_fasttext = np.load('test_s2_tfidf_fasttext.npy', mmap_mode='
```

In [26]:
```
input2 = Input(shape=(300,)) #for sentence1 embeddings
norm2 = BatchNormalization()(input2)
input3 = Input(shape=(300,)) #for sentence2 embeddings
norm3 = BatchNormalization()(input3)
merged = concatenate([norm2, norm3]) #merging all inputs
#norm1 = BatchNormalization()(merged)
dense1 = Dense(units = 100, activation = 'relu', kernel_regularizer = L1(0.0
drop1 = Dropout(0.01)(dense1)
dense2 = Dense(units = 100, activation = 'relu', kernel_regularizer = L1(0.0
drop2 = Dropout(0.01)(dense2)
dense3 = Dense(units = 100, activation = 'relu', kernel_regularizer = L1(0.0
dense4 = Dense(units = 50, activation = 'relu')(dense3)
output = Dense(units = 3, activation = 'softmax')(dense4)
model3 = Model(inputs = [input2, input3], outputs = output)
```

In [ ]:
```
model3.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metric

tf.debugging.set_log_device_placement(True)

# Place tensors on the CPU
with tf.device('/CPU:0'):
    model3.fit([x_train_s1_tfidf_fasttext, x_train_s2_tfidf_fasttext], y_tra
           validation_data = ([x_val_s1_tfidf_fasttext, x_val_s2_tfidf_fastt
```

In [28]:
```
from keras import backend as K
K.set_value(model2_a.optimizer.learning_rate, 0.0001)
```

In [29]:
```
with tf.device('/CPU:0'):
    model3.fit([x_train_s1_tfidf_fasttext, x_train_s2_tfidf_fasttext], y_tra
           validation_data = ([x_val_s1_tfidf_fasttext, x_val_s2_tfidf_fastt
```

```
Epoch 1/20
4292/4292 [==============================] - 22s 5ms/step - loss: 0.6575 - accu
racy: 0.7645 - val_loss: 0.7161 - val_accuracy: 0.7389
Epoch 2/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.6574 - accu
racy: 0.7648 - val_loss: 0.7147 - val_accuracy: 0.7398
Epoch 3/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.6570 - accu
racy: 0.7644 - val_loss: 0.7154 - val_accuracy: 0.7404
Epoch 4/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.6566 - accu
racy: 0.7650 - val_loss: 0.7211 - val_accuracy: 0.7345
Epoch 5/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.6569 - accu
racy: 0.7642 - val_loss: 0.7158 - val_accuracy: 0.7375
Epoch 6/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.6562 - accu
racy: 0.7650 - val_loss: 0.7148 - val_accuracy: 0.7400
Epoch 7/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.6573 - accu
racy: 0.7646 - val_loss: 0.7190 - val_accuracy: 0.7347
Epoch 8/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.6566 - accu
racy: 0.7646 - val_loss: 0.7182 - val_accuracy: 0.7392
Epoch 9/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.6564 - accu
racy: 0.7649 - val_loss: 0.7261 - val_accuracy: 0.7327
Epoch 10/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.6561 - accu
racy: 0.7654 - val_loss: 0.7198 - val_accuracy: 0.7415
Epoch 11/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.6568 - accu
racy: 0.7655 - val_loss: 0.7195 - val_accuracy: 0.7376
Epoch 12/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.6566 - accu
racy: 0.7652 - val_loss: 0.7172 - val_accuracy: 0.7385
Epoch 13/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.6557 - accu
racy: 0.7653 - val_loss: 0.7186 - val_accuracy: 0.7360
Epoch 14/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.6568 - accu
racy: 0.7652 - val_loss: 0.7143 - val_accuracy: 0.7430
Epoch 15/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.6558 - accu
racy: 0.7658 - val_loss: 0.7180 - val_accuracy: 0.7387
Epoch 16/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.6566 - accu
racy: 0.7650 - val_loss: 0.7182 - val_accuracy: 0.7397
Epoch 17/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.6563 - accu
racy: 0.7648 - val_loss: 0.7168 - val_accuracy: 0.7434
Epoch 18/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.6564 - accu
```

```
racy: 0.7656 - val_loss: 0.7197 - val_accuracy: 0.7402
Epoch 19/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.6566 - accu
racy: 0.7650 - val_loss: 0.7179 - val_accuracy: 0.7381
Epoch 20/20
4292/4292 [==============================] - 20s 5ms/step - loss: 0.6559 - accu
racy: 0.7651 - val_loss: 0.7208 - val_accuracy: 0.7375
```

model4: tfidf weighted fasttext embeddings + extracted features

In [11]:

```
input1 = Input(shape=(29,)) #for extracted features
input2 = Input(shape=(300,)) #for sentence1 embeddings
input3 = Input(shape=(300,)) #for sentence2 embeddings
merged = concatenate([input1, input2, input3]) #merging all inputs
norm1 = BatchNormalization()(merged)
dense1 = Dense(units = 100, activation = 'relu')(norm1)
dense2 = Dense(units = 80, activation = 'relu')(dense1)
dense3 = Dense(units = 50, activation = 'relu')(dense2)
dense4 = Dense(units = 40, activation = 'relu')(dense3)
dense5 = Dense(units = 30, activation = 'relu')(dense4)
dense6 = Dense(units = 20, activation = 'relu')(dense5)
dense7 = Dense(units = 10, activation = 'relu')(dense6)
output = Dense(units = 3, activation = 'softmax')(dense7)
model4 = Model(inputs = [input1, input2, input3], outputs = output)
```

In [18]:

```
model4.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metric
tf.debugging.set_log_device_placement(True)
# Place tensors on the CPU
with tf.device('/CPU:0'):
    model4.fit([X_train_1, x_train_s1_tfidf_fasttext, x_train_s2_tfidf_fastt
           validation_data = ([X_val_1, x_val_s1_tfidf_fasttext, x_val_s2_tf
```

```
4292/4292 [==============================] - 17s 4ms/step - loss: 0.4808 - ac
curacy: 0.8075 - val_loss: 0.6431 - val_accuracy: 0.7488
Epoch 87/100
4292/4292 [==============================] - 18s 4ms/step - loss: 0.4800 - ac
curacy: 0.8078 - val_loss: 0.6495 - val_accuracy: 0.7484
Epoch 88/100
4292/4292 [==============================] - 18s 4ms/step - loss: 0.4798 - ac
curacy: 0.8079 - val_loss: 0.6540 - val_accuracy: 0.7456
Epoch 89/100
4292/4292 [==============================] - 17s 4ms/step - loss: 0.4797 - ac
curacy: 0.8081 - val_loss: 0.6521 - val_accuracy: 0.7466
Epoch 90/100
4292/4292 [==============================] - 17s 4ms/step - loss: 0.4792 - ac
curacy: 0.8081 - val_loss: 0.6741 - val_accuracy: 0.7452
Epoch 91/100
4292/4292 [==============================] - 17s 4ms/step - loss: 0.4786 - ac
curacy: 0.8088 - val_loss: 0.6479 - val_accuracy: 0.7469
Epoch 92/100
4292/4292 [==============================] - 17s 4ms/step - loss: 0.4787 - ac
```

Error analysis on model4

train accuracy = 0.8, validation accuracy = 0.75

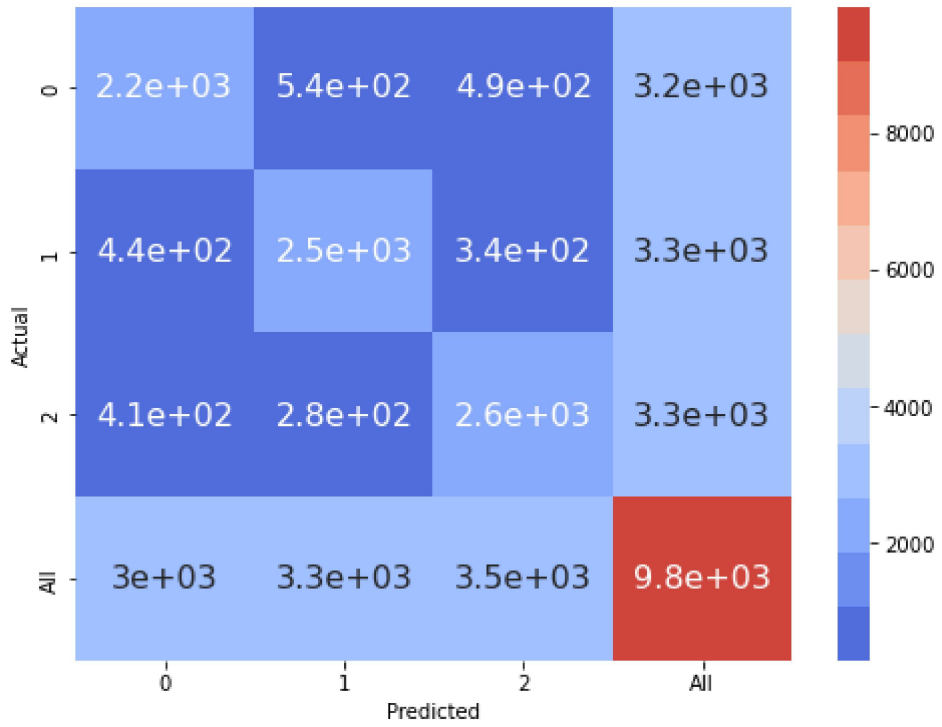
Confusion matrix: model4

In [19]:
```
val_pred = model4.predict([X_val_1, x_val_s1_tfidf_fasttext, x_val_s2_tfidf_
```

In [22]:
```
label_pred = tf.argmax(val_pred, axis = 1)
```

In [31]:
```
data =  {'y_Actual':  y_val,
        'y_Predicted': label_pred}
df = pd.DataFrame(data, columns=['y_Actual','y_Predicted'])
confusion_matrix = pd.crosstab(df['y_Actual'], df['y_Predicted'], rownames=[
```

In [42]:
```
plt.figure(figsize = (8,6))
colormap = sns.color_palette("coolwarm", 12)
sns.heatmap(confusion_matrix, annot=True, cmap = colormap, annot_kws={"fonts
plt.show()
```



meaning of labels = 0:neutral, 1:contradiction, 2:entailment

-It seems that most errors are occuring for neutral pairs which are predicted as contradiction

-best oerformance is for entailment pairs

-

Erronous points: model4

In [50]:
```
df2 = val_df[['sentence1', 'sentence2']]
```

In [51]:
```
df2['y_true'] = y_val
df2['y_pred'] = label_pred
```

```
C:\Users\anike\.conda\envs\tf_test\lib\site-packages\ipykernel_launcher.py:1: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  """Entry point for launching an IPython kernel.
C:\Users\anike\.conda\envs\tf_test\lib\site-packages\ipykernel_launcher.py:2: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
```

In [53]:
```
# all incorrectly predicted pairs
erronous_pairs = df2[df2['y_true']!=df2['y_pred']]
```

In [69]:
```
#true:entailment | pred:contradiction
ec = df2[(df2['y_true'] == 2) & (df2['y_pred'] == 1)]
print('some pairs of sentences belonging to entailment but predicted as cont
print('.'*100)
for i,row in ec.sample(5).iterrows():
    print('sentence1 : {}\nsentence2 : {}\nactual label : {}\npredicted labe
    print('.'*100)
```

```
some pairs of sentences belonging to entailment but predicted as contradiction:
....................................................................................

sentence1 : A guy wearing a black shirt and gray pants pushing his shopping car
t past a candy aisle of a Walmart.
sentence2 : A man is inside a store.
actual label : 2
predicted label : 1
....................................................................................

sentence1 : A girl is throwing a football on the beach.
sentence2 : A girl is playing with a football.
actual label : 2
predicted label : 1
....................................................................................

sentence1 : A middle-aged man in a bicycle race trying his hardest on the sand
surrounded by watching crowd.
sentence2 : A man is riding a bicycle.
actual label : 2
predicted label : 1
....................................................................................

sentence1 : A motorcycle rider is making a sharp turn.
sentence2 : The motorcyclist is turning.
actual label : 2
predicted label : 1
....................................................................................

sentence1 : Men and women in swimsuits hangout on rocks above water.
sentence2 : The men and women are dressed in clothes for the beach.
actual label : 2
predicted label : 1
....................................................................................
```

Observations: It seems that the model is not being able to draw the meaning of the sentences. Because two sentences here mean essentially the same ,still the model thinks that they are opposite in meaning. The model is incapable of learning the meaning, sequence information and symantics.

In [70]:
```
#true:contradiction | pred:contradiction
cc = df2[(df2['y_true'] == 1) & (df2['y_pred'] == 1)]
print('some pairs of sentences belonging to contradiction and predicted as c
print('.'*100)
for i,row in cc.sample(5).iterrows():
    print('sentence1 : {}\nsentence2 : {}\nactual label : {}\npredicted labe
    print('.'*100)
```

```
some pairs of sentences belonging to contradiction and predicted as contradicti
on:
....................................................................................

sentence1 : A boy wearing blue jeans is skateboarding.
sentence2 : a boy sleeps in a car
actual label : 1
predicted label : 1
....................................................................................

sentence1 : A person in gray snowboarding down a hill.
sentence2 : The snowboarder is wearing white.
actual label : 1
predicted label : 1
....................................................................................

sentence1 : A man in a green safety jacket and pants stands, facing away, near
bags of trash while people stand on the curb across the street.
sentence2 : Nobody has a jacket
actual label : 1
predicted label : 1
....................................................................................

sentence1 : A cyclists rides up a hill amid cheering bystanders.
sentence2 : The person is on a motorcycle.
actual label : 1
predicted label : 1
....................................................................................

sentence1 : A man snowboards.
sentence2 : A man is drinking coffee at the cafe.
actual label : 1
predicted label : 1
....................................................................................
```

In [71]:
```
#true:entailment | pred:entailment
ee = df2[(df2['y_true'] == 2) & (df2['y_pred'] == 2)]
print('some pairs of sentences belonging to entailment and predicted as enta
print('.'*100)
for i,row in ee.sample(5).iterrows():
    print('sentence1 : {}\nsentence2 : {}\nactual label : {}\npredicted labe
    print('.'*100)
```

```
some pairs of sentences belonging to entailment and predicted as entailment:
....................................................................................

sentence1 : Two people in winter clothing sliding down a snow-covered hill.
sentence2 : People are sledding down a hill.
actual label : 2
predicted label : 2
....................................................................................

sentence1 : Speakers in a room full of people sitting in chairs.
sentence2 : There are people indoors.
actual label : 2
predicted label : 2
....................................................................................

sentence1 : A man in a black coat eats a doughnut while other men look out of f
rame.
sentence2 : A man in a black coat eats a doughnut.
actual label : 2
predicted label : 2
....................................................................................

sentence1 : A group of teenagers are standing in front of some tents, most of t
hem are holding skateboards.
sentence2 : A group of teenagers are holding skateboards
actual label : 2
predicted label : 2
....................................................................................

sentence1 : A woman is writing something on a post-it note which is hanging on
a bulletin board with a lot of other post-it notes.
sentence2 : The woman is writing a note.
actual label : 2
predicted label : 2
....................................................................................
```

observation: It seems that when two sentences share some words or pos then the model is abe to identify them correctly

In [68]:
```
#true:contradiction | pred:entailment
contradict_entailment = df2[(df2['y_true'] == 1) & (df2['y_pred'] == 2)]
print('some pairs of sentences belonging to contradiction but predicted as e
print('.'*100)
for i,row in contradict_entailment.sample(5).iterrows():
    print('sentence1 : {}\nsentence2 : {}\nactual label : {}\npredicted labe
    print('.'*100)
```

```
some pairs of sentences belonging to contradiction but predicted as entailment:
....................................................................................

sentence1 : the shadow silhouette of a woman standing near the water looking at
a large attraction on the other side.
sentence2 : She is in the water.
actual label : 1
predicted label : 2
....................................................................................

sentence1 : A woman prepares ingredients for a bowl of soup.
sentence2 : A soup bowl prepares a woman.
actual label : 1
predicted label : 2
....................................................................................

sentence1 : A girl in a red polka dot bikini is jumping off a sand dune.
sentence2 : A girl in a bikini is buried in the sand.
actual label : 1
predicted label : 2
....................................................................................

sentence1 : A husky and a black cat nuzzling.
sentence2 : A cat attacks a dog.
actual label : 1
predicted label : 2
....................................................................................

sentence1 : A woman holds her child out of a red window, next to a Color TV sig
n.
sentence2 : The TV holds a woman who holds a child.
actual label : 1
predicted label : 2
....................................................................................
```

observation: Again we can see that the model is only considering presence of similar words to predict the label. It is not considering semantic meaning or polarity. Also it is not being able to recognize structure and voice of sentences.

In [ ]:
```
1
```