EMOTION PREDICTION IN CONVERSATIONS: APPROACH

1. Business Problem

```
cd drive/My\ Drive/MELD
```

```
/content/drive/My Drive/MELD
```

2. Loading Required Libraries

```
import nltk
nltk.download('stopwords')
import pandas as pd
import seaborn as sns
import numpy as np
import xgboost as xgb
from tqdm import tqdm
from sklearn.svm import SVC
from keras.models import Sequential
from keras.layers.recurrent import LSTM, GRU
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.embeddings import Embedding
from keras.layers.normalization import BatchNormalization
from keras.utils import np_utils
from sklearn import preprocessing, decomposition, model_selection, metrics, pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from keras.layers import GlobalMaxPooling1D, Conv1D, MaxPooling1D, Flatten, Bidirectional,
from keras.preprocessing import sequence, text
from keras.callbacks import EarlyStopping
from nltk import word_tokenize
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarnir
  import pandas.util.testing as tm
Using TensorFlow backend.
```

3.Loading the Data-set

```
train = pd.read_csv('train_sent_emo.csv')
cv = pd.read_csv('dev_sent_emo.csv')
test = pd.read_csv('test_sent_emo.csv')
```

4.EDA

```
print('The shape of the trian data',train.shape)
print('The shape of the test data',test.shape)
print('The shape of the cv data',cv.shape)
```

```
The shape of the trian data (9989, 11)
The shape of the test data (2610, 11)
The shape of the cv data (1109, 11)
```

```
train.head()
```

	Sr No.	Utterance	Speaker	Emotion	Sentiment	Dialogue_ID	Utterance_ID	Season
0	1	also I was the point person on my company□s tr...	Chandler	neutral	neutral	0	0	8

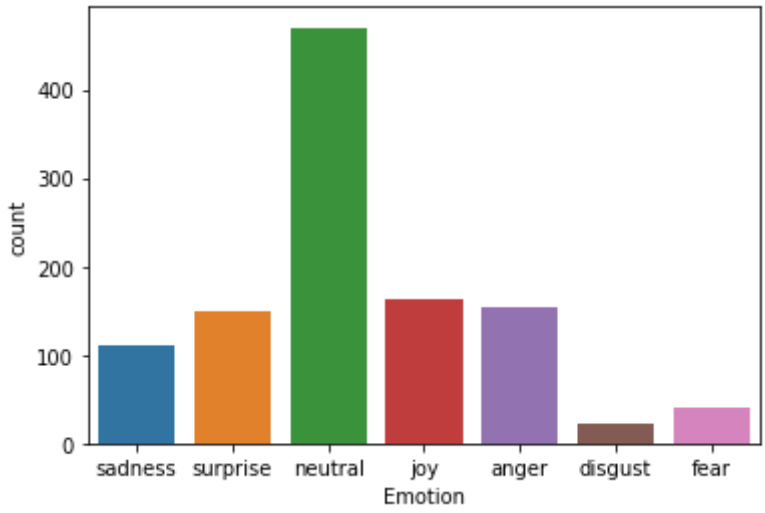
You

```
#Imbalanced Data
ax = sns.countplot(x=train['Emotion'], data=train)
print(train['Emotion'].value_counts())
```

```
neutral     4710
joy         1743
surprise    1205
```

```
#Imbalanced Data
ax = sns.countplot(x=cv['Emotion'], data=cv)
print(cv['Emotion'].value_counts())
```

```
neutral     470
joy         163
anger       153
surprise    150
sadness     111
fear         40
disgust      22
Name: Emotion, dtype: int64
```




```
#Imbalanced Data
ax = sns.countplot(x=test['Emotion'], data=test)
print(test['Emotion'].value_counts())
```

Observation : Neutral emotion is dominating in CV ,Test set and Train set

```
train.info() # No missing values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9989 entries, 0 to 9988
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   Sr No.        9989 non-null   int64 
 1   Utterance     9989 non-null   object
 2   Speaker       9989 non-null   object
 3   Emotion       9989 non-null   object
 4   Sentiment     9989 non-null   object
 5   Dialogue_ID   9989 non-null   int64 
 6   Utterance_ID  9989 non-null   int64 
 7   Season        9989 non-null   int64 
 8   Episode       9989 non-null   int64 
 9   StartTime     9989 non-null   object
 10  EndTime       9989 non-null   object
dtypes: int64(5), object(6)
memory usage: 858.6+ KB
```

```
cv.info()  # No missing values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1109 entries, 0 to 1108
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   Sr No.        1109 non-null   int64 
 1   Utterance     1109 non-null   object
 2   Speaker       1109 non-null   object
 3   Emotion       1109 non-null   object
 4   Sentiment     1109 non-null   object
 5   Dialogue_ID   1109 non-null   int64 
 6   Utterance_ID  1109 non-null   int64 
 7   Season        1109 non-null   int64 
 8   Episode       1109 non-null   int64 
 9   StartTime     1109 non-null   object
 10  EndTime       1109 non-null   object
dtypes: int64(5), object(6)
memory usage: 95.4+ KB
```

```
test.info() # No missing values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2610 entries, 0 to 2609
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   Sr No.        2610 non-null   int64 
 1   Utterance     2610 non-null   object
```

5.Preprocessing

Train Test Split

```
 9   StartTime     2610 non-null   object
```

```
X_train , y_train = train[['Utterance']] , train[['Emotion']]
X_cv , y_cv = cv[['Utterance']] , cv[['Emotion']]
X_test , y_test = test[['Utterance']] , test[['Emotion']]
```

Defining Multi Class LogLoss

```
#Defining Multi Class LogLoss
def multiclass_logloss(actual, predicted, eps=1e-15):
    """Multi class version of Logarithmic Loss metric.
    :param actual: Array containing the actual target classes
    :param predicted: Matrix with class predictions, one probability per class
    """
    # Convert 'actual' to a binary array if it's not already:
    if len(actual.shape) == 1:
        actual2 = np.zeros((actual.shape[0], predicted.shape[1]))
        for i, val in enumerate(actual):
            actual2[i, val] = 1
        actual = actual2

    clip = np.clip(predicted, eps, 1 - eps)
    rows = actual.shape[0]
    vsota = np.sum(actual * np.log(clip))
    return -1.0 / rows * vsota


#Define a scorer to be used in Grid Search
mll_scorer = metrics.make_scorer(multiclass_logloss, greater_is_better=False, needs_proba=


#Using LabelEncoder to vectorise labels
lbl_enc = preprocessing.LabelEncoder()
y_train_enc = lbl_enc.fit_transform(y_train.Emotion.values)
y_cv_enc = lbl_enc.transform(y_cv.Emotion.values)
y_test_enc = lbl_enc.transform(y_test.Emotion.values)


y_train.Emotion.values[1:200]
```

```
array(['neutral', 'neutral', 'neutral', 'surprise', 'neutral', 'neutral',
       'neutral', 'neutral', 'neutral', 'fear', 'neutral', 'surprise',
       'neutral', 'surprise', 'sadness', 'surprise', 'fear', 'neutral',
       'neutral', 'neutral', 'neutral', 'neutral', 'joy', 'sadness',
       'surprise', 'neutral', 'disgust', 'sadness', 'neutral', 'neutral',
       'joy', 'neutral', 'joy', 'surprise', 'surprise', 'surprise',
       'neutral', 'neutral', 'neutral', 'surprise', 'sadness', 'neutral',
       'surprise', 'joy', 'surprise', 'neutral', 'neutral', 'neutral',
       'neutral', 'neutral', 'joy', 'joy', 'joy', 'sadness', 'neutral',
       'neutral', 'neutral', 'neutral', 'neutral', 'neutral', 'surprise',
       'joy', 'surprise', 'joy', 'neutral', 'neutral', 'anger', 'joy',
       'neutral', 'surprise', 'anger', 'anger', 'anger', 'neutral',
       'neutral', 'sadness', 'sadness', 'sadness', 'surprise', 'anger',
       'anger', 'anger', 'anger', 'neutral', 'anger', 'neutral',
       'neutral', 'neutral', 'neutral', 'joy', 'neutral', 'joy',
       'neutral', 'neutral', 'neutral', 'joy', 'neutral', 'neutral',
       'neutral', 'neutral', 'neutral', 'joy', 'neutral', 'neutral',
       'disgust', 'anger', 'anger', 'anger', 'anger', 'anger', 'anger',
       'neutral', 'neutral', 'anger', 'neutral', 'joy', 'neutral',
       'neutral', 'joy', 'joy', 'joy', 'joy', 'neutral', 'joy', 'disgust',
       'surprise', 'disgust', 'neutral', 'fear', 'neutral', 'surprise',
       'fear', 'disgust', 'anger', 'joy', 'neutral', 'surprise',
       'neutral', 'neutral', 'neutral', 'neutral', 'surprise', 'neutral',
       'neutral', 'anger', 'neutral', 'neutral', 'sadness', 'surprise',
       'sadness', 'anger', 'sadness', 'neutral', 'sadness', 'neutral',
       'neutral', 'neutral', 'neutral', 'neutral', 'joy', 'anger',
       'anger', 'anger', 'neutral', 'anger', 'joy', 'joy', 'joy', 'joy',
       'disgust', 'surprise', 'neutral', 'neutral', 'anger', 'joy',
       'neutral', 'neutral', 'fear', 'neutral', 'fear', 'joy', 'joy',
       'joy', 'joy', 'neutral', 'neutral', 'neutral', 'joy', 'neutral',
```

```
y_train_enc[1:200]
```

```
array([4, 4, 4, 6, 4, 4, 4, 4, 4, 2, 4, 6, 4, 6, 5, 6, 2, 4, 4, 4, 4, 4,
       3, 5, 6, 4, 1, 5, 4, 4, 3, 4, 3, 6, 6, 6, 4, 4, 4, 6, 5, 4, 6, 3,
       6, 4, 4, 4, 4, 4, 3, 3, 3, 5, 4, 4, 4, 4, 4, 4, 6, 3, 6, 3, 4, 4,
       0, 3, 4, 6, 0, 0, 0, 4, 4, 5, 5, 5, 6, 0, 0, 0, 0, 4, 0, 4, 4, 4,
       4, 3, 4, 3, 4, 4, 4, 3, 4, 4, 4, 4, 4, 3, 4, 4, 1, 0, 0, 0, 0, 0,
       0, 4, 4, 0, 4, 3, 4, 4, 3, 3, 3, 3, 4, 3, 1, 6, 1, 4, 2, 4, 6, 2,
       1, 0, 3, 4, 6, 4, 4, 4, 4, 6, 4, 4, 0, 4, 4, 5, 6, 5, 0, 5, 4, 5,
       4, 4, 4, 4, 4, 3, 0, 0, 0, 4, 0, 3, 3, 3, 3, 1, 6, 4, 4, 0, 3, 4,
       4, 2, 4, 2, 3, 3, 3, 3, 4, 4, 4, 3, 4, 3, 2, 4, 5, 6, 2, 4, 4, 4,
       3])
```

```
import re

### Dataset Preprocessing training set
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()
train_corpus = []
for i in range(0, len(X_train)):
    review = re.sub('[^a-zA-Z]', ' ', X_train['Utterance'][i])
    review = review.lower()
    review = review.split()

    review = [ps.stem(word) for word in review if not word in stopwords.words('english')]
    review = ' '.join(review)
    train_corpus.append(review)
```

```
import re

### Dataset Preprocessing cv set
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()
cv_corpus = []
for i in range(0, len(X_cv)):
    review = re.sub('[^a-zA-Z]', ' ', X_cv['Utterance'][i])
    review = review.lower()
    review = review.split()

    review = [ps.stem(word) for word in review if not word in stopwords.words('english')]
    review = ' '.join(review)
    cv_corpus.append(review)


import re

### Dataset Preprocessing test set
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()
test_corpus = []
for i in range(0, len(X_test)):
    review = re.sub('[^a-zA-Z]', ' ', X_test['Utterance'][i])
    review = review.lower()
    review = review.split()

    review = [ps.stem(word) for word in review if not word in stopwords.words('english')]
    review = ' '.join(review)
    test_corpus.append(review)


X_train['clean_utterance'] = train_corpus
X_train.drop('Utterance',axis=1,inplace=True)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarnir
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/stable/us
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.6/dist-packages/pandas/core/frame.py:3997: SettingWithCopyWarr
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  errors=errors,
```

```
X_cv['clean_utterance'] = cv_corpus
X_cv.drop('Utterance',axis=1,inplace=True)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarnir
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/stable/us
```

```
X_test['clean_utterance'] = test_corpus
X_test.drop('Utterance',axis=1,inplace=True)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarnir
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/stable/us
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.6/dist-packages/pandas/core/frame.py:3997: SettingWithCopyWarr
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  errors=errors,
```

6.MODEL'S

```
#TFIDF + LR Model
tfv = TfidfVectorizer(min_df=3,  max_features=None,
            strip_accents='unicode', analyzer='word',token_pattern=r'\w{1,}',
            ngram_range=(1, 3), use_idf=1,smooth_idf=1,sublinear_tf=1,
            stop_words = 'english')

# Fitting TF-IDF to both training and test sets (semi-supervised learning)
tfv.fit(list(X_train['clean_utterance']) + list(X_cv['clean_utterance']) + list(X_test['cl
X_train_tfv =  tfv.transform(X_train['clean_utterance'])
X_valid_tfv = tfv.transform(X_cv['clean_utterance'])
X_test_tfv = tfv.transform(X_test['clean_utterance'])


X_train_tfv.shape,X_valid_tfv.shape,X_test_tfv.shape
```

```
((9989, 3179), (1109, 3179), (2610, 3179))
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score


alpha = [10 ** x for x in range(-6, 3)]


# initialize Our first RandomForestRegressor model...
regr2 = LogisticRegression()

# declare parameters for hyperparameter tuning
parameters = {'C':alpha}
```

```
# Perform cross validation
clf = GridSearchCV(regr2,
                   param_grid = parameters,
                   scoring=mll_scorer,
                   n_jobs = -1,
                   verbose = 10, refit=True, cv=2)
result = clf.fit(X_train_tfv, y_train_enc)

# Summarize results
print("Best: %f using %s" % (result.best_score_, result.best_params_))
means = result.cv_results_['mean_test_score']
stds = result.cv_results_['std_test_score']
params = result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f 1(%f) with: %r" % (mean, stdev, param))
```

```
Fitting 2 folds for each of 9 candidates, totalling 18 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    1.6s
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:    2.1s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    2.6s
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:    3.8s
[Parallel(n_jobs=-1)]: Done  18 out of  18 | elapsed:    5.5s finished
Best: -1.417605 using {'C': 1}
-1.536665 1(0.000177) with: {'C': 1e-06}
-1.536648 1(0.000177) with: {'C': 1e-05}
-1.536480 1(0.000177) with: {'C': 0.0001}
-1.534834 1(0.000178) with: {'C': 0.001}
-1.520896 1(0.000221) with: {'C': 0.01}
-1.463340 1(0.000332) with: {'C': 0.1}
-1.417605 1(0.004722) with: {'C': 1}
-1.622374 1(0.001400) with: {'C': 10}
-2.300202 1(0.032847) with: {'C': 100}
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_logistic.py:940: Converg
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
lr = LogisticRegression(C = 1)
lr.fit(X_train_tfv, y_train_enc)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_logistic.py:940: Converg
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Train,Test and CV loss

```
predictions = lr.predict_proba(X_train_tfv)
print ("logloss: %0.3f " % multiclass_logloss(y_train_enc, predictions))
```

```
logloss: 1.129
```

```
predictions = lr.predict_proba(X_valid_tfv)
print ("logloss: %0.3f " % multiclass_logloss(y_cv_enc, predictions))
```

```
logloss: 1.477
```

```
predictions = lr.predict_proba(X_test_tfv)
print ("logloss: %0.3f " % multiclass_logloss(y_test_enc, predictions))
```

```
logloss: 1.372
```

```
# BOW + LR Model
ctv = CountVectorizer(analyzer='word',token_pattern=r'\w{1,}',
            ngram_range=(1, 3), stop_words = 'english')

# Fitting Count Vectorizer to both training and test sets (semi-supervised learning)
ctv.fit(list(X_train['clean_utterance']) + list(X_cv['clean_utterance']) + list(X_test['cl
X_train_ctv =  ctv.transform(X_train['clean_utterance'])
X_valid_ctv = ctv.transform(X_cv['clean_utterance'])
X_test_ctv = ctv.transform(X_test['clean_utterance'])
```

```
X_train_ctv.shape,X_valid_ctv.shape,X_test_ctv.shape
```

```
((9989, 51082), (1109, 51082), (2610, 51082))
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score


alpha = [10 ** x for x in range(-6, 3)]


# initialize Our first RandomForestRegressor model...
regr2 = LogisticRegression()

# declare parameters for hyperparameter tuning
parameters = {'C':alpha}

# Perform cross validation
clf = GridSearchCV(regr2,
                   param_grid = parameters,
                   scoring=mll_scorer,
                   n_jobs = -1,
                   verbose = 10, refit=True, cv=2)
```

```
                    verbose = 10, refit=True, cv=2)
result = clf.fit(X_train_ctv, y_train_enc)

# Summarize results
print("Best: %f using %s" % (result.best_score_, result.best_params_))
means = result.cv_results_['mean_test_score']
stds = result.cv_results_['std_test_score']
params = result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f 1(%f) with: %r" % (mean, stdev, param))
```

```
Fitting 2 folds for each of 9 candidates, totalling 18 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    4.6s
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:    9.8s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:   12.7s
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:   24.1s
[Parallel(n_jobs=-1)]: Done  18 out of  18 | elapsed:   39.4s finished
Best: -1.427496 using {'C': 0.1}
-1.536658 1(0.000176) with: {'C': 1e-06}
-1.536569 1(0.000177) with: {'C': 1e-05}
-1.535706 1(0.000178) with: {'C': 0.0001}
-1.528054 1(0.000209) with: {'C': 0.001}
-1.489068 1(0.000305) with: {'C': 0.01}
-1.427496 1(0.001608) with: {'C': 0.1}
-1.460494 1(0.008207) with: {'C': 1}
-1.784036 1(0.024584) with: {'C': 10}
-2.629078 1(0.109224) with: {'C': 100}
```

```
lr = LogisticRegression(C = 0.1)
lr.fit(X_train_ctv, y_train_enc)
```

```
LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
predictions = lr.predict_proba(X_train_ctv)
print ("logloss: %0.3f " % multiclass_logloss(y_train_enc, predictions))
```

```
logloss: 1.115
```

```
predictions = lr.predict_proba(X_valid_ctv)
print ("logloss: %0.3f " % multiclass_logloss(y_cv_enc, predictions))
```

```
logloss: 1.501
```

```
predictions = lr.predict_proba(X_test_ctv)
print ("logloss: %0.3f " % multiclass_logloss(y_test_enc, predictions))
```

```
logloss: 1.394
```

Since TFID + LR model was better so we will move forward with TFIDF

```
# Word Vectors Model

# make sure you have the glove_vectors file
import pickle
with open('/content/drive/My Drive/MELD/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())


# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_utterance']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train.append(vector)

print(len(avg_w2v_vectors_train))
print(len(avg_w2v_vectors_train[0]))
```

```
100%|██████████| 9989/9989 [00:00<00:00, 85670.50it/s]9989
300
```

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_utterance']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_cv.append(vector)

print(len(avg_w2v_vectors_cv))
print(len(avg_w2v_vectors_cv[0]))
```

```
100%|██████████| 1109/1109 [00:00<00:00, 54411.58it/s]1109
300
```

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_utterance']): # for each review/sentence
```

```
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test.append(vector)

print(len(avg_w2v_vectors_test))
print(len(avg_w2v_vectors_test[0]))
```

```
100%|██████████| 2610/2610 [00:00<00:00, 49256.83it/s]2610
300
```

```
xtrain_glove = np.array(avg_w2v_vectors_train)
xvalid_glove = np.array(avg_w2v_vectors_cv)
xtest_glove = np.array(avg_w2v_vectors_test)


import xgboost as xgb

# Fitting a simple xgboost on glove features
clf = xgb.XGBClassifier(max_depth=7, n_estimators=200, colsample_bytree=0.8,
                        subsample=0.8, nthread=10, learning_rate=0.1, silent=False)
clf.fit(xtrain_glove, y_train_enc)
predictions = clf.predict_proba(xvalid_glove)


print ("Valid logloss: %0.3f " % multiclass_logloss(y_cv_enc, predictions))
```

```
Valid logloss: 1.726
```

```
# Truncated SVD + SVM Model

# Apply SVD, I chose 120 components. 120-200 components are good enough for SVM model.
svd = decomposition.TruncatedSVD(n_components=120)
svd.fit(X_train_tfv)
X_train_svd = svd.transform(X_train_tfv)
X_valid_svd = svd.transform(X_valid_tfv)
X_test_svd = svd.transform(X_test_tfv)

# Scale the data obtained from SVD. Renaming variable to reuse without scaling.
scl = preprocessing.StandardScaler()
scl.fit(X_train_svd)
X_train_svd_scl = scl.transform(X_train_svd)
X_valid_svd_scl = scl.transform(X_valid_svd)
X_test_svd_scl = scl.transform(X_test_svd)


X_train_svd_scl.shape,X_valid_svd_scl.shape,X_test_svd_scl.shape
```

```
((9989, 120), (1109, 120), (2610, 120))
```

```
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.calibration import CalibratedClassifierCV

# initialize Our first RandomForestRegressor model...
svm = SVC(C=1)
regr2 = CalibratedClassifierCV(svm)
# declare parameters for hyperparameter tuning
regr2.fit(X_train_svd_scl, y_train_enc)
```

```
CalibratedClassifierCV(base_estimator=SVC(C=1, break_ties=False, cache_size=200,
                                          class_weight=None, coef0=0.0,
                                          decision_function_shape='ovr',
                                          degree=3, gamma='scale', kernel='rbf',
                                          max_iter=-1, probability=False,
                                          random_state=None, shrinking=True,
                                          tol=0.001, verbose=False),
                       cv=None, method='sigmoid')
```

```
predictions = regr2.predict_proba(X_train_svd_scl)
print ("logloss: %0.3f " % multiclass_logloss(y_train_enc, predictions))
```

```
logloss: 1.359
```

```
predictions = regr2.predict_proba(X_valid_svd_scl)
print ("logloss: %0.3f " % multiclass_logloss(y_cv_enc, predictions))
```

```
logloss: 1.556
```

```
predictions = regr2.predict_proba(X_test_svd_scl)
print ("logloss: %0.3f " % multiclass_logloss(y_test_enc, predictions))
```

```
logloss: 1.455
```

```
#MLP Model

# scale the data before any neural net:
scl = preprocessing.StandardScaler()
xtrain_glove_scl = scl.fit_transform(xtrain_glove)
xvalid_glove_scl = scl.transform(xvalid_glove)
xtest_glove_scl = scl.transform(xtest_glove)


y_train_enc_nn = np_utils.to_categorical(y_train_enc)
y_cv_enc_nn = np_utils.to_categorical(y_cv_enc)
y_test_enc_nn = np_utils.to_categorical(y_test_enc)


model = Sequential()

model.add(Dense(128, input_dim=300, activation='relu'))
model.add(Dropout(0.2))
```

```
model.add(BatchNormalization())

model.add(Dense(256, activation='relu'))
model.add(Dropout(0.3))
model.add(BatchNormalization())

model.add(Dense(256, activation='relu'))
model.add(Dropout(0.3))
model.add(BatchNormalization())

model.add(Dense(128, activation='relu'))
model.add(Dropout(0.3))
model.add(BatchNormalization())

model.add(Dense(7))
model.add(Activation('softmax'))

# compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam')


model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 128)               38528     
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 128)               512       
_________________________________________________________________
dense_2 (Dense)              (None, 256)               33024     
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0         
_________________________________________________________________
batch_normalization_2 (Batch (None, 256)               1024      
_________________________________________________________________
dense_3 (Dense)              (None, 256)               65792     
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0         
_________________________________________________________________
batch_normalization_3 (Batch (None, 256)               1024      
_________________________________________________________________
dense_4 (Dense)              (None, 128)               32896     
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0         
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512       
_________________________________________________________________
dense_5 (Dense)              (None, 7)                 903       
_________________________________________________________________
activation_1 (Activation)    (None, 7)                 0         
=================================================================
Total params: 174,215
Trainable params: 172,679
Non-trainable params: 1,536
```

```
y_cv_enc.shape
```

```
(1109,)
```

```
history = model.fit(xtrain_glove_scl, y=y_train_enc_nn, batch_size=64,
          epochs=10, verbose=1,
          validation_data=(xvalid_glove_scl, y_cv_enc_nn))
```

```
Train on 9989 samples, validate on 1109 samples
Epoch 1/10
9989/9989 [==============================] - 3s 340us/step - loss: 2.0240 - val_loss
Epoch 2/10
9989/9989 [==============================] - 1s 118us/step - loss: 1.6321 - val_loss
Epoch 3/10
9989/9989 [==============================] - 1s 120us/step - loss: 1.5517 - val_loss
Epoch 4/10
9989/9989 [==============================] - 1s 119us/step - loss: 1.4975 - val_loss
Epoch 5/10
9989/9989 [==============================] - 1s 139us/step - loss: 1.4646 - val_loss
Epoch 6/10
9989/9989 [==============================] - 1s 118us/step - loss: 1.4432 - val_loss
Epoch 7/10
9989/9989 [==============================] - 1s 119us/step - loss: 1.4356 - val_loss
Epoch 8/10
9989/9989 [==============================] - 1s 117us/step - loss: 1.4194 - val_loss
Epoch 9/10
9989/9989 [==============================] - 1s 119us/step - loss: 1.4066 - val_loss
Epoch 10/10
9989/9989 [==============================] - 1s 121us/step - loss: 1.3958 - val_loss
```

```
# using keras tokenizer here
token = text.Tokenizer(num_words=None)
max_len = 70
a =list(X_train['clean_utterance']) + list(X_cv['clean_utterance']) + list(X_test['clean_u
token.fit_on_texts(a)
xtrain_seq = token.texts_to_sequences(X_train['clean_utterance'])
xvalid_seq = token.texts_to_sequences(X_cv['clean_utterance'])
xtest_seq = token.texts_to_sequences(X_test['clean_utterance'])

# zero pad the sequences
xtrain_pad = sequence.pad_sequences(xtrain_seq, maxlen=max_len)
xvalid_pad = sequence.pad_sequences(xvalid_seq, maxlen=max_len)
xtest_pad = sequence.pad_sequences(xtest_seq, maxlen=max_len)

word_index = token.word_index


# create an embedding matrix for the words we have in the dataset
embedding_matrix = np.zeros((len(word_index) + 1, 300))
for word, i in tqdm(word_index.items()):
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

```
100%|██████████| 4709/4709 [00:00<00:00, 1526114.78it/s]
```

```
#LSTM Model

# A simple LSTM with glove embeddings and two dense layers
model = Sequential()
model.add(Embedding(len(word_index) + 1,
                    300,
                    weights=[embedding_matrix],
                    input_length=max_len,
                    trainable=False))
model.add(SpatialDropout1D(0.3))
model.add(LSTM(300, dropout=0.3, recurrent_dropout=0.3))

model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.8))

model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.8))

model.add(Dense(7))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')

# Fit the model with early stopping callback
earlystop = EarlyStopping(monitor='val_loss', min_delta=0, patience=3, verbose=0, mode='au

history=model.fit(xtrain_pad, y=y_train_enc_nn, batch_size=512, epochs=200, verbose=1, val
```

```
Train on 9989 samples, validate on 1109 samples
Epoch 1/200
9989/9989 [==============================] - 3s 338us/step - loss: 1.9392 - val_loss
Epoch 2/200
9989/9989 [==============================] - 3s 319us/step - loss: 1.9254 - val_loss
Epoch 3/200
9989/9989 [==============================] - 3s 292us/step - loss: 1.9119 - val_loss
Epoch 4/200
9989/9989 [==============================] - 3s 302us/step - loss: 1.8990 - val_loss
Epoch 5/200
9989/9989 [==============================] - 3s 297us/step - loss: 1.8864 - val_loss
Epoch 6/200
9989/9989 [==============================] - 3s 285us/step - loss: 1.8743 - val_loss
Epoch 7/200
9989/9989 [==============================] - 3s 307us/step - loss: 1.8625 - val_loss
Epoch 8/200
9989/9989 [==============================] - 3s 293us/step - loss: 1.8510 - val_loss
Epoch 9/200
9989/9989 [==============================] - 3s 292us/step - loss: 1.8400 - val_loss
Epoch 10/200
9989/9989 [==============================] - 3s 300us/step - loss: 1.8292 - val_loss
Epoch 11/200
9989/9989 [==============================] - 3s 293us/step - loss: 1.8187 - val_loss
Epoch 12/200
9989/9989 [==============================] - 3s 315us/step - loss: 1.8085 - val_loss
Epoch 13/200
9989/9989 [==============================] - 3s 290us/step - loss: 1.7986 - val_loss
Epoch 14/200
9989/9989 [==============================] - 3s 308us/step - loss: 1.7889 - val_loss
Epoch 15/200
9989/9989 [==============================] - 3s 300us/step - loss: 1.7796 - val_loss
Epoch 16/200
9989/9989 [==============================] - 3s 281us/step - loss: 1.7705 - val_loss
Epoch 17/200
9989/9989 [==============================] - 3s 326us/step - loss: 1.7617 - val_loss
Epoch 18/200
9989/9989 [==============================] - 3s 316us/step - loss: 1.7532 - val_loss
Epoch 19/200
9989/9989 [==============================] - 3s 303us/step - loss: 1.7449 - val_loss
Epoch 20/200
9989/9989 [==============================] - 3s 303us/step - loss: 1.7368 - val_loss
Epoch 21/200
9989/9989 [==============================] - 3s 308us/step - loss: 1.7291 - val_loss
Epoch 22/200
9989/9989 [==============================] - 3s 336us/step - loss: 1.7215 - val_loss
Epoch 23/200
9989/9989 [==============================] - 3s 313us/step - loss: 1.7142 - val_loss
Epoch 24/200
9989/9989 [==============================] - 3s 313us/step - loss: 1.7071 - val_loss
Epoch 25/200
9989/9989 [==============================] - 3s 313us/step - loss: 1.7003 - val_loss
Epoch 26/200
9989/9989 [==============================] - 3s 287us/step - loss: 1.6937 - val_loss
Epoch 27/200
9989/9989 [==============================] - 3s 322us/step - loss: 1.6874 - val_loss
Epoch 28/200
9989/9989 [==============================] - 3s 309us/step - loss: 1.6812 - val_loss
Epoch 29/200
9989/9989 [==============================] - 3s 295us/step - loss: 1.6754 - val_loss
Epoch 30/200
9989/9989 [==============================] - 3s 314us/step - loss: 1.6696 - val_loss
```

```
Epoch 31/200
9989/9989 [==============================] - 3s 308us/step - loss: 1.6642 - val_loss
Epoch 32/200
9989/9989 [==============================] - 3s 295us/step - loss: 1.6589 - val_loss
Epoch 33/200
9989/9989 [==============================] - 3s 286us/step - loss: 1.6538 - val_loss
Epoch 34/200
9989/9989 [==============================] - 3s 302us/step - loss: 1.6489 - val_loss
Epoch 35/200
9989/9989 [==============================] - 3s 309us/step - loss: 1.6442 - val_loss
Epoch 36/200
9989/9989 [==============================] - 3s 298us/step - loss: 1.6397 - val_loss
Epoch 37/200
9989/9989 [==============================] - 3s 306us/step - loss: 1.6354 - val_loss
Epoch 38/200
9989/9989 [==============================] - 3s 316us/step - loss: 1.6312 - val_loss
Epoch 39/200
9989/9989 [==============================] - 3s 309us/step - loss: 1.6272 - val_loss
Epoch 40/200
9989/9989 [==============================] - 3s 323us/step - loss: 1.6234 - val_loss
Epoch 41/200
9989/9989 [==============================] - 3s 302us/step - loss: 1.6197 - val_loss
Epoch 42/200
9989/9989 [==============================] - 3s 292us/step - loss: 1.6163 - val_loss
Epoch 43/200
9989/9989 [==============================] - 3s 323us/step - loss: 1.6129 - val_loss
Epoch 44/200
9989/9989 [==============================] - 3s 303us/step - loss: 1.6096 - val_loss
Epoch 45/200
9989/9989 [==============================] - 3s 285us/step - loss: 1.6065 - val_loss
Epoch 46/200
9989/9989 [==============================] - 3s 317us/step - loss: 1.6035 - val_loss
Epoch 47/200
9989/9989 [==============================] - 3s 332us/step - loss: 1.6007 - val_loss
Epoch 48/200
9989/9989 [==============================] - 3s 303us/step - loss: 1.5979 - val_loss
Epoch 49/200
9989/9989 [==============================] - 3s 310us/step - loss: 1.5953 - val_loss
Epoch 50/200
9989/9989 [==============================] - 3s 289us/step - loss: 1.5928 - val_loss
Epoch 51/200
9989/9989 [==============================] - 3s 333us/step - loss: 1.5904 - val_loss
Epoch 52/200
9989/9989 [==============================] - 3s 310us/step - loss: 1.5882 - val_loss
Epoch 53/200
9989/9989 [==============================] - 3s 313us/step - loss: 1.5860 - val_loss
Epoch 54/200
9989/9989 [==============================] - 3s 304us/step - loss: 1.5839 - val_loss
Epoch 55/200
9989/9989 [==============================] - 3s 323us/step - loss: 1.5819 - val_loss
Epoch 56/200
9989/9989 [==============================] - 3s 300us/step - loss: 1.5800 - val_loss
Epoch 57/200
9989/9989 [==============================] - 3s 295us/step - loss: 1.5781 - val_loss
Epoch 58/200
9989/9989 [==============================] - 3s 302us/step - loss: 1.5764 - val_loss
Epoch 59/200
9989/9989 [==============================] - 3s 314us/step - loss: 1.5747 - val_loss
Epoch 60/200
9989/9989 [==============================] - 3s 336us/step - loss: 1.5731 - val_loss
Epoch 61/200
9989/9989 [==============================] - 3s 346us/step - loss: 1.5715 - val_loss
```

```
9989/9989 [==============================] - 3s 340us/step - loss: 1.5719 - val_loss
Epoch 62/200
9989/9989 [==============================] - 4s 356us/step - loss: 1.5701 - val_loss
Epoch 63/200
9989/9989 [==============================] - 3s 293us/step - loss: 1.5686 - val_loss
Epoch 64/200
9989/9989 [==============================] - 3s 322us/step - loss: 1.5673 - val_loss
Epoch 65/200
9989/9989 [==============================] - 3s 328us/step - loss: 1.5660 - val_loss
Epoch 66/200
9989/9989 [==============================] - 3s 298us/step - loss: 1.5648 - val_loss
Epoch 67/200
9989/9989 [==============================] - 3s 298us/step - loss: 1.5636 - val_loss
Epoch 68/200
9989/9989 [==============================] - 3s 318us/step - loss: 1.5625 - val_loss
Epoch 69/200
9989/9989 [==============================] - 3s 294us/step - loss: 1.5614 - val_loss
Epoch 70/200
9989/9989 [==============================] - 3s 318us/step - loss: 1.5603 - val_loss
Epoch 71/200
9989/9989 [==============================] - 3s 289us/step - loss: 1.5593 - val_loss
Epoch 72/200
9989/9989 [==============================] - 3s 296us/step - loss: 1.5584 - val_loss
Epoch 73/200
9989/9989 [==============================] - 3s 305us/step - loss: 1.5575 - val_loss
Epoch 74/200
9989/9989 [==============================] - 3s 293us/step - loss: 1.5566 - val_loss
Epoch 75/200
9989/9989 [==============================] - 3s 295us/step - loss: 1.5557 - val_loss
Epoch 76/200
9989/9989 [==============================] - 3s 312us/step - loss: 1.5549 - val_loss
Epoch 77/200
9989/9989 [==============================] - 3s 323us/step - loss: 1.5542 - val_loss
Epoch 78/200
9989/9989 [==============================] - 3s 297us/step - loss: 1.5534 - val_loss
Epoch 79/200
9989/9989 [==============================] - 3s 305us/step - loss: 1.5527 - val_loss
Epoch 80/200
9989/9989 [==============================] - 3s 300us/step - loss: 1.5521 - val_loss
Epoch 81/200
9989/9989 [==============================] - 3s 315us/step - loss: 1.5514 - val_loss
Epoch 82/200
9989/9989 [==============================] - 3s 292us/step - loss: 1.5508 - val_loss
Epoch 83/200
9989/9989 [==============================] - 3s 310us/step - loss: 1.5502 - val_loss
Epoch 84/200
9989/9989 [==============================] - 3s 302us/step - loss: 1.5496 - val_loss
Epoch 85/200
9989/9989 [==============================] - 3s 296us/step - loss: 1.5491 - val_loss
Epoch 86/200
9989/9989 [==============================] - 3s 330us/step - loss: 1.5485 - val_loss
Epoch 87/200
9989/9989 [==============================] - 3s 299us/step - loss: 1.5480 - val_loss
Epoch 88/200
9989/9989 [==============================] - 3s 282us/step - loss: 1.5476 - val_loss
Epoch 89/200
9989/9989 [==============================] - 3s 293us/step - loss: 1.5471 - val_loss
Epoch 90/200
9989/9989 [==============================] - 3s 298us/step - loss: 1.5467 - val_loss
Epoch 91/200
9989/9989 [==============================] - 3s 311us/step - loss: 1.5462 - val_loss
Epoch 92/200
```

```
9989/9989 [==============================] - 3s 314us/step - loss: 1.5458 - val_loss
Epoch 93/200
9989/9989 [==============================] - 3s 338us/step - loss: 1.5454 - val_loss
Epoch 94/200
9989/9989 [==============================] - 3s 305us/step - loss: 1.5451 - val_loss
Epoch 95/200
9989/9989 [==============================] - 3s 298us/step - loss: 1.5447 - val_loss
Epoch 96/200
9989/9989 [==============================] - 3s 298us/step - loss: 1.5444 - val_loss
Epoch 97/200
9989/9989 [==============================] - 3s 318us/step - loss: 1.5440 - val_loss
Epoch 98/200
9989/9989 [==============================] - 3s 290us/step - loss: 1.5437 - val_loss
Epoch 99/200
9989/9989 [==============================] - 3s 316us/step - loss: 1.5434 - val_loss
Epoch 100/200
9989/9989 [==============================] - 3s 305us/step - loss: 1.5431 - val_loss
Epoch 101/200
9989/9989 [==============================] - 3s 320us/step - loss: 1.5429 - val_loss
Epoch 102/200
9989/9989 [==============================] - 3s 304us/step - loss: 1.5426 - val_loss
Epoch 103/200
9989/9989 [==============================] - 3s 285us/step - loss: 1.5423 - val_loss
Epoch 104/200
9989/9989 [==============================] - 3s 290us/step - loss: 1.5421 - val_loss
Epoch 105/200
9989/9989 [==============================] - 3s 287us/step - loss: 1.5418 - val_loss
Epoch 106/200
9989/9989 [==============================] - 3s 311us/step - loss: 1.5416 - val_loss
Epoch 107/200
9989/9989 [==============================] - 3s 283us/step - loss: 1.5414 - val_loss
Epoch 108/200
9989/9989 [==============================] - 3s 332us/step - loss: 1.5412 - val_loss
Epoch 109/200
9989/9989 [==============================] - 3s 297us/step - loss: 1.5410 - val_loss
Epoch 110/200
9989/9989 [==============================] - 3s 306us/step - loss: 1.5408 - val_loss
Epoch 111/200
9989/9989 [==============================] - 3s 300us/step - loss: 1.5406 - val_loss
Epoch 112/200
9989/9989 [==============================] - 3s 302us/step - loss: 1.5404 - val_loss
Epoch 113/200
9989/9989 [==============================] - 3s 295us/step - loss: 1.5403 - val_loss
Epoch 114/200
9989/9989 [==============================] - 3s 286us/step - loss: 1.5401 - val_loss
Epoch 115/200
9989/9989 [==============================] - 3s 330us/step - loss: 1.5399 - val_loss
Epoch 116/200
9989/9989 [==============================] - 3s 329us/step - loss: 1.5398 - val_loss
Epoch 117/200
9989/9989 [==============================] - 3s 304us/step - loss: 1.5397 - val_loss
Epoch 118/200
9989/9989 [==============================] - 3s 295us/step - loss: 1.5395 - val_loss
Epoch 119/200
9989/9989 [==============================] - 3s 301us/step - loss: 1.5394 - val_loss
Epoch 120/200
9989/9989 [==============================] - 3s 299us/step - loss: 1.5393 - val_loss
Epoch 121/200
9989/9989 [==============================] - 3s 290us/step - loss: 1.5391 - val_loss
Epoch 122/200
9989/9989 [==============================] - 3s 335us/step - loss: 1.5390 - val_loss
Epoch 123/200
```

```
Epoch 123/200
9989/9989 [==============================] - 3s 293us/step - loss: 1.5389 - val_loss
Epoch 124/200
9989/9989 [==============================] - 3s 305us/step - loss: 1.5388 - val_loss
Epoch 125/200
9989/9989 [==============================] - 3s 305us/step - loss: 1.5387 - val_loss
Epoch 126/200
9989/9989 [==============================] - 3s 298us/step - loss: 1.5386 - val_loss
Epoch 127/200
9989/9989 [==============================] - 3s 306us/step - loss: 1.5385 - val_loss
Epoch 128/200
9989/9989 [==============================] - 3s 304us/step - loss: 1.5384 - val_loss
Epoch 129/200
9989/9989 [==============================] - 3s 300us/step - loss: 1.5383 - val_loss
Epoch 130/200
9989/9989 [==============================] - 3s 298us/step - loss: 1.5383 - val_loss
Epoch 131/200
9989/9989 [==============================] - 3s 308us/step - loss: 1.5382 - val_loss
Epoch 132/200
9989/9989 [==============================] - 3s 307us/step - loss: 1.5381 - val_loss
Epoch 133/200
9989/9989 [==============================] - 3s 311us/step - loss: 1.5380 - val_loss
Epoch 134/200
9989/9989 [==============================] - 3s 291us/step - loss: 1.5380 - val_loss
Epoch 135/200
9989/9989 [==============================] - 3s 322us/step - loss: 1.5379 - val_loss
Epoch 136/200
9989/9989 [==============================] - 3s 305us/step - loss: 1.5379 - val_loss
Epoch 137/200
9989/9989 [==============================] - 3s 289us/step - loss: 1.5378 - val_loss
Epoch 138/200
9989/9989 [==============================] - 3s 330us/step - loss: 1.5377 - val_loss
Epoch 139/200
9989/9989 [==============================] - 3s 305us/step - loss: 1.5377 - val_loss
Epoch 140/200
9989/9989 [==============================] - 3s 314us/step - loss: 1.5376 - val_loss
```

We would obviouslt get better results with LSTM but we need more epochs

```
Epoch 142/200
```

```
# MODEL 7 : Bi -LSTM


# A simple bidirectional LSTM with glove embeddings and two dense layers
model = Sequential()
model.add(Embedding(len(word_index) + 1,
                    300,
                    weights=[embedding_matrix],
                    input_length=max_len,
                    trainable=False))
model.add(SpatialDropout1D(0.3))
model.add(Bidirectional(LSTM(300, dropout=0.3, recurrent_dropout=0.3)))

model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.8))

model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.8))

model.add(Dense(7))
```

```
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')

# Fit the model with early stopping callback
earlystop = EarlyStopping(monitor='val_loss', min_delta=0, patience=3, verbose=0, mode='au

history=model.fit(xtrain_pad, y=y_train_enc_nn, batch_size=512, epochs=200, verbose=1, val
```

```
Train on 9989 samples, validate on 1109 samples
Epoch 1/200
9989/9989 [==============================] - 6s 636us/step - loss: 1.9397 - val_loss
Epoch 2/200
9989/9989 [==============================] - 6s 563us/step - loss: 1.9259 - val_loss
Epoch 3/200
9989/9989 [==============================] - 5s 541us/step - loss: 1.9123 - val_loss
Epoch 4/200
9989/9989 [==============================] - 5s 546us/step - loss: 1.8993 - val_loss
Epoch 5/200
9989/9989 [==============================] - 6s 589us/step - loss: 1.8866 - val_loss
Epoch 6/200
9989/9989 [==============================] - 6s 588us/step - loss: 1.8744 - val_loss
Epoch 7/200
9989/9989 [==============================] - 6s 588us/step - loss: 1.8625 - val_loss
Epoch 8/200
9989/9989 [==============================] - 6s 567us/step - loss: 1.8510 - val_loss
Epoch 9/200
9989/9989 [==============================] - 6s 571us/step - loss: 1.8398 - val_loss
Epoch 10/200
9989/9989 [==============================] - 6s 572us/step - loss: 1.8289 - val_loss
Epoch 11/200
9989/9989 [==============================] - 6s 587us/step - loss: 1.8184 - val_loss
Epoch 12/200
9989/9989 [==============================] - 6s 623us/step - loss: 1.8082 - val_loss
Epoch 13/200
9989/9989 [==============================] - 6s 583us/step - loss: 1.7983 - val_loss
Epoch 14/200
9989/9989 [==============================] - 5s 539us/step - loss: 1.7886 - val_loss
Epoch 15/200
9989/9989 [==============================] - 6s 581us/step - loss: 1.7793 - val_loss
Epoch 16/200
9989/9989 [==============================] - 6s 619us/step - loss: 1.7702 - val_loss
Epoch 17/200
9989/9989 [==============================] - 6s 623us/step - loss: 1.7614 - val_loss
Epoch 18/200
9989/9989 [==============================] - 6s 602us/step - loss: 1.7528 - val_loss
Epoch 19/200
9989/9989 [==============================] - 6s 604us/step - loss: 1.7445 - val_loss
Epoch 20/200
9989/9989 [==============================] - 6s 616us/step - loss: 1.7364 - val_loss
Epoch 21/200
9989/9989 [==============================] - 6s 562us/step - loss: 1.7287 - val_loss
Epoch 22/200
9989/9989 [==============================] - 6s 557us/step - loss: 1.7211 - val_loss
Epoch 23/200
9989/9989 [==============================] - 6s 596us/step - loss: 1.7139 - val_loss
Epoch 24/200
9989/9989 [==============================] - 6s 598us/step - loss: 1.7068 - val_loss
Epoch 25/200
9989/9989 [==============================] - 6s 594us/step - loss: 1.7001 - val_loss
Epoch 26/200
9989/9989 [==============================] - 6s 553us/step - loss: 1.6935 - val_loss
Epoch 27/200
9989/9989 [==============================] - 5s 545us/step - loss: 1.6871 - val_loss
Epoch 28/200
9989/9989 [==============================] - 6s 585us/step - loss: 1.6810 - val_loss
Epoch 29/200
9989/9989 [==============================] - 5s 549us/step - loss: 1.6751 - val_loss
Epoch 30/200
9989/9989 [==============================] - 6s 593us/step - loss: 1.6695 - val_loss
```

```
Epoch 31/200
9989/9989 [==============================] - 6s 596us/step - loss: 1.6640 - val_loss
Epoch 32/200
9989/9989 [==============================] - 6s 558us/step - loss: 1.6587 - val_loss
Epoch 33/200
9989/9989 [==============================] - 6s 610us/step - loss: 1.6537 - val_loss
Epoch 34/200
9989/9989 [==============================] - 6s 566us/step - loss: 1.6488 - val_loss
Epoch 35/200
9989/9989 [==============================] - 6s 580us/step - loss: 1.6441 - val_loss
Epoch 36/200
9989/9989 [==============================] - 5s 539us/step - loss: 1.6396 - val_loss
Epoch 37/200
9989/9989 [==============================] - 6s 566us/step - loss: 1.6353 - val_loss
Epoch 38/200
9989/9989 [==============================] - 6s 551us/step - loss: 1.6311 - val_loss
Epoch 39/200
9989/9989 [==============================] - 6s 561us/step - loss: 1.6271 - val_loss
Epoch 40/200
9989/9989 [==============================] - 6s 588us/step - loss: 1.6233 - val_loss
Epoch 41/200
9989/9989 [==============================] - 6s 554us/step - loss: 1.6197 - val_loss
Epoch 42/200
9989/9989 [==============================] - 6s 565us/step - loss: 1.6161 - val_loss
Epoch 43/200
9989/9989 [==============================] - 6s 568us/step - loss: 1.6127 - val_loss
Epoch 44/200
9989/9989 [==============================] - 6s 572us/step - loss: 1.6095 - val_loss
Epoch 45/200
9989/9989 [==============================] - 6s 554us/step - loss: 1.6064 - val_loss
Epoch 46/200
9989/9989 [==============================] - 6s 568us/step - loss: 1.6034 - val_loss
Epoch 47/200
9989/9989 [==============================] - 6s 562us/step - loss: 1.6006 - val_loss
Epoch 48/200
9989/9989 [==============================] - 6s 572us/step - loss: 1.5979 - val_loss
Epoch 49/200
9989/9989 [==============================] - 6s 627us/step - loss: 1.5952 - val_loss
Epoch 50/200
9989/9989 [==============================] - 6s 600us/step - loss: 1.5927 - val_loss
Epoch 51/200
9989/9989 [==============================] - 6s 566us/step - loss: 1.5903 - val_loss
Epoch 52/200
9989/9989 [==============================] - 6s 558us/step - loss: 1.5880 - val_loss
Epoch 53/200
9989/9989 [==============================] - 6s 587us/step - loss: 1.5859 - val_loss
Epoch 54/200
9989/9989 [==============================] - 6s 645us/step - loss: 1.5838 - val_loss
Epoch 55/200
9989/9989 [==============================] - 6s 585us/step - loss: 1.5818 - val_loss
Epoch 56/200
9989/9989 [==============================] - 6s 561us/step - loss: 1.5799 - val_loss
Epoch 57/200
9989/9989 [==============================] - 6s 620us/step - loss: 1.5780 - val_loss
Epoch 58/200
9989/9989 [==============================] - 6s 598us/step - loss: 1.5763 - val_loss
Epoch 59/200
9989/9989 [==============================] - 6s 639us/step - loss: 1.5746 - val_loss
Epoch 60/200
9989/9989 [==============================] - 6s 584us/step - loss: 1.5730 - val_loss
Epoch 61/200
9989/9989 [==============================] - 6s 606us/step - loss: 1.5714 - val_loss
```

```
9989/9989 [==============================] - 6s 606us/step - loss: 1.5714 - val_loss
Epoch 62/200
9989/9989 [==============================] - 6s 588us/step - loss: 1.5700 - val_loss
Epoch 63/200
9989/9989 [==============================] - 6s 622us/step - loss: 1.5686 - val_loss
Epoch 64/200
9989/9989 [==============================] - 6s 593us/step - loss: 1.5672 - val_loss
Epoch 65/200
9989/9989 [==============================] - 6s 622us/step - loss: 1.5659 - val_loss
Epoch 66/200
9989/9989 [==============================] - 6s 582us/step - loss: 1.5647 - val_loss
Epoch 67/200
9989/9989 [==============================] - 6s 568us/step - loss: 1.5635 - val_loss
Epoch 68/200
9989/9989 [==============================] - 6s 608us/step - loss: 1.5624 - val_loss
Epoch 69/200
9989/9989 [==============================] - 6s 599us/step - loss: 1.5613 - val_loss
Epoch 70/200
9989/9989 [==============================] - 6s 588us/step - loss: 1.5603 - val_loss
Epoch 71/200
9989/9989 [==============================] - 6s 575us/step - loss: 1.5593 - val_loss
Epoch 72/200
9989/9989 [==============================] - 6s 630us/step - loss: 1.5583 - val_loss
Epoch 73/200
9989/9989 [==============================] - 6s 600us/step - loss: 1.5574 - val_loss
Epoch 74/200
9989/9989 [==============================] - 6s 583us/step - loss: 1.5565 - val_loss
Epoch 75/200
9989/9989 [==============================] - 6s 569us/step - loss: 1.5557 - val_loss
Epoch 76/200
9989/9989 [==============================] - 6s 598us/step - loss: 1.5549 - val_loss
Epoch 77/200
9989/9989 [==============================] - 6s 603us/step - loss: 1.5541 - val_loss
Epoch 78/200
9989/9989 [==============================] - 6s 615us/step - loss: 1.5534 - val_loss
Epoch 79/200
9989/9989 [==============================] - 6s 649us/step - loss: 1.5527 - val_loss
Epoch 80/200
9989/9989 [==============================] - 6s 593us/step - loss: 1.5520 - val_loss
Epoch 81/200
9989/9989 [==============================] - 6s 606us/step - loss: 1.5514 - val_loss
Epoch 82/200
9989/9989 [==============================] - 6s 585us/step - loss: 1.5507 - val_loss
Epoch 83/200
9989/9989 [==============================] - 6s 599us/step - loss: 1.5502 - val_loss
Epoch 84/200
9989/9989 [==============================] - 6s 612us/step - loss: 1.5496 - val_loss
Epoch 85/200
9989/9989 [==============================] - 6s 565us/step - loss: 1.5490 - val_loss
Epoch 86/200
9989/9989 [==============================] - 6s 629us/step - loss: 1.5485 - val_loss
Epoch 87/200
9989/9989 [==============================] - 6s 644us/step - loss: 1.5480 - val_loss
Epoch 88/200
9989/9989 [==============================] - 6s 594us/step - loss: 1.5475 - val_loss
Epoch 89/200
9989/9989 [==============================] - 6s 553us/step - loss: 1.5471 - val_loss
Epoch 90/200
9989/9989 [==============================] - 6s 591us/step - loss: 1.5466 - val_loss
Epoch 91/200
9989/9989 [==============================] - 6s 582us/step - loss: 1.5462 - val_loss
Epoch 92/200
```

```
9989/9989 [==============================] - 6s 600us/step - loss: 1.5458 - val_loss
Epoch 93/200
9989/9989 [==============================] - 6s 609us/step - loss: 1.5454 - val_loss
Epoch 94/200
9989/9989 [==============================] - 6s 625us/step - loss: 1.5451 - val_loss
Epoch 95/200
9989/9989 [==============================] - 6s 558us/step - loss: 1.5447 - val_loss
Epoch 96/200
9989/9989 [==============================] - 6s 605us/step - loss: 1.5444 - val_loss
Epoch 97/200
9989/9989 [==============================] - 6s 570us/step - loss: 1.5440 - val_loss
Epoch 98/200
9989/9989 [==============================] - 6s 601us/step - loss: 1.5437 - val_loss
Epoch 99/200
9989/9989 [==============================] - 6s 607us/step - loss: 1.5434 - val_loss
Epoch 100/200
9989/9989 [==============================] - 6s 574us/step - loss: 1.5431 - val_loss
Epoch 101/200
9989/9989 [==============================] - 6s 565us/step - loss: 1.5428 - val_loss
Epoch 102/200
9989/9989 [==============================] - 6s 562us/step - loss: 1.5426 - val_loss
Epoch 103/200
9989/9989 [==============================] - 6s 609us/step - loss: 1.5423 - val_loss
Epoch 104/200
9989/9989 [==============================] - 6s 598us/step - loss: 1.5421 - val_loss
Epoch 105/200
9989/9989 [==============================] - 6s 631us/step - loss: 1.5418 - val_loss
Epoch 106/200
9989/9989 [==============================] - 6s 623us/step - loss: 1.5416 - val_loss
Epoch 107/200
9989/9989 [==============================] - 6s 600us/step - loss: 1.5414 - val_loss
Epoch 108/200
9989/9989 [==============================] - 6s 583us/step - loss: 1.5412 - val_loss
Epoch 109/200
9989/9989 [==============================] - 6s 580us/step - loss: 1.5410 - val_loss
Epoch 110/200
9989/9989 [==============================] - 6s 577us/step - loss: 1.5408 - val_loss
Epoch 111/200
9989/9989 [==============================] - 6s 620us/step - loss: 1.5406 - val_loss
Epoch 112/200
9989/9989 [==============================] - 6s 649us/step - loss: 1.5404 - val_loss
Epoch 113/200
9989/9989 [==============================] - 6s 620us/step - loss: 1.5403 - val_loss
```

```
#MODEL 8 : Trying customised Ensembling

# this is the main ensembling class. how to use it is in the next cell!
import numpy as np
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold, KFold
import pandas as pd
import os
import sys
import logging

logging.basicConfig(
    level=logging.DEBUG,
    format="[%(asctime)s] %(levelname)s %(message)s",
    datefmt="%H:%M:%S", stream=sys.stdout)
```

```
logger = logging.getLogger(__name__)


class Ensembler(object):
    def __init__(self, model_dict, num_folds=3, task_type='classification', optimize=roc_a
                 lower_is_better=False, save_path=None):
        """
        Ensembler init function
        :param model_dict: model dictionary, see README for its format
        :param num_folds: the number of folds for ensembling
        :param task_type: classification or regression
        :param optimize: the function to optimize for, e.g. AUC, logloss, etc. Must have t
        :param lower_is_better: is lower value of optimization function better or higher
        :param save_path: path to which model pickles will be dumped to along with generat
        """

        self.model_dict = model_dict
        self.levels = len(self.model_dict)
        self.num_folds = num_folds
        self.task_type = task_type
        self.optimize = optimize
        self.lower_is_better = lower_is_better
        self.save_path = save_path

        self.training_data = None
        self.test_data = None
        self.y = None
        self.lbl_enc = None
        self.y_enc = None
        self.train_prediction_dict = None
        self.test_prediction_dict = None
        self.num_classes = None

    def fit(self, training_data, y, lentrain):
        """
        :param training_data: training data in tabular format
        :param y: binary, multi-class or regression
        :return: chain of models to be used in prediction
        """

        self.training_data = training_data
        self.y = y

        if self.task_type == 'classification':
            self.num_classes = len(np.unique(self.y))
            logger.info("Found %d classes", self.num_classes)
            self.lbl_enc = LabelEncoder()
            self.y_enc = self.lbl_enc.fit_transform(self.y)
            kf = StratifiedKFold(n_splits=self.num_folds)
            train_prediction_shape = (lentrain, self.num_classes)
        else:
            self.num_classes = -1
            self.y_enc = self.y
            kf = KFold(n_splits=self.num_folds)
            train_prediction_shape = (lentrain, 1)
```

```
        self.train_prediction_dict = {}
        for level in range(self.levels):
            self.train_prediction_dict[level] = np.zeros((train_prediction_shape[0],
                                                          train_prediction_shape[1] * len(

        for level in range(self.levels):

            if level == 0:
                temp_train = self.training_data
            else:
                temp_train = self.train_prediction_dict[level - 1]

            for model_num, model in enumerate(self.model_dict[level]):
                validation_scores = []
                foldnum = 1
                for train_index, valid_index in kf.split(self.train_prediction_dict[0], se
                    logger.info("Training Level %d Fold # %d. Model # %d", level, foldnum,

                    if level != 0:
                        l_training_data = temp_train[train_index]
                        l_validation_data = temp_train[valid_index]
                        model.fit(l_training_data, self.y_enc[train_index])
                    else:
                        l0_training_data = temp_train[0][model_num]
                        if type(l0_training_data) == list:
                            l_training_data = [x[train_index] for x in l0_training_data]
                            l_validation_data = [x[valid_index] for x in l0_training_data]
                        else:
                            l_training_data = l0_training_data[train_index]
                            l_validation_data = l0_training_data[valid_index]
                        model.fit(l_training_data, self.y_enc[train_index])

                    logger.info("Predicting Level %d. Fold # %d. Model # %d", level, foldn

                    if self.task_type == 'classification':
                        temp_train_predictions = model.predict_proba(l_validation_data)
                        self.train_prediction_dict[level][valid_index,
                        (model_num * self.num_classes):(model_num * self.num_classes) +
                                                        self.num_classes] = temp_train_pred

                    else:
                        temp_train_predictions = model.predict(l_validation_data)
                        self.train_prediction_dict[level][valid_index, model_num] = temp_t
                    validation_score = self.optimize(self.y_enc[valid_index], temp_train_p
                    validation_scores.append(validation_score)
                    logger.info("Level %d. Fold # %d. Model # %d. Validation Score = %f",
                                level, foldnum, model_num,
                                validation_score)
                    foldnum += 1
                avg_score = np.mean(validation_scores)
                std_score = np.std(validation_scores)
                logger.info("Level %d. Model # %d. Mean Score = %f. Std Dev = %f", level,
                            avg_score, std_score)

            logger.info("Saving predictions for level # %d", level)
```

```
            train_predictions_df = pd.DataFrame(self.train_prediction_dict[level])
            train_predictions_df.to_csv(os.path.join(self.save_path, "train_predictions_le
                                        index=False, header=None)

        return self.train_prediction_dict

    def predict(self, test_data, lentest):
        self.test_data = test_data
        if self.task_type == 'classification':
            test_prediction_shape = (lentest, self.num_classes)
        else:
            test_prediction_shape = (lentest, 1)

        self.test_prediction_dict = {}
        for level in range(self.levels):
            self.test_prediction_dict[level] = np.zeros((test_prediction_shape[0],
                                                         test_prediction_shape[1] * len(se
        self.test_data = test_data
        for level in range(self.levels):
            if level == 0:
                temp_train = self.training_data
                temp_test = self.test_data
            else:
                temp_train = self.train_prediction_dict[level - 1]
                temp_test = self.test_prediction_dict[level - 1]

            for model_num, model in enumerate(self.model_dict[level]):

                logger.info("Training Fulldata Level %d. Model # %d", level, model_num)
                if level == 0:
                    model.fit(temp_train[0][model_num], self.y_enc)
                else:
                    model.fit(temp_train, self.y_enc)

                logger.info("Predicting Test Level %d. Model # %d", level, model_num)

                if self.task_type == 'classification':
                    if level == 0:
                        temp_test_predictions = model.predict_proba(temp_test[0][model_num
                    else:
                        temp_test_predictions = model.predict_proba(temp_test)
                    self.test_prediction_dict[level][:, (model_num * self.num_classes): (m
                                                                                          se

                else:
                    if level == 0:
                        temp_test_predictions = model.predict(temp_test[0][model_num])
                    else:
                        temp_test_predictions = model.predict(temp_test)
                    self.test_prediction_dict[level][:, model_num] = temp_test_predictions

            test_predictions_df = pd.DataFrame(self.test_prediction_dict[level])
            test_predictions_df.to_csv(os.path.join(self.save_path, "test_predictions_leve
                                       index=False, header=None)
```

```
        return self.test_prediction_dict


import warnings
warnings.filterwarnings("ignore")
# specify the data to be used for every level of ensembling:
train_data_dict = {0: [X_train_tfv, X_train_ctv, X_train_tfv, X_train_ctv], 1: [xtrain_glo
test_data_dict = {0: [X_valid_tfv, X_valid_ctv, X_valid_tfv, X_valid_ctv], 1: [xvalid_glov

model_dict = {0: [LogisticRegression(), LogisticRegression(), MultinomialNB(alpha=0.1), Mu

              1: [xgb.XGBClassifier(silent=True, n_estimators=120, max_depth=7)]}

ens = Ensembler(model_dict=model_dict, num_folds=3, task_type='classification',
                optimize=multiclass_logloss, lower_is_better=True, save_path='')

ens.fit(train_data_dict, y_train_enc, lentrain=xtrain_glove.shape[0])
preds = ens.predict(test_data_dict, lentest=xvalid_glove.shape[0])
```

```
[02:19:31] INFO Found 7 classes
[02:19:31] INFO Training Level 0 Fold # 1. Model # 0
[02:19:32] INFO Predicting Level 0. Fold # 1. Model # 0
[02:19:32] INFO Level 0. Fold # 1. Model # 0. Validation Score = 1.425598
[02:19:32] INFO Training Level 0 Fold # 2. Model # 0
[02:19:33] INFO Predicting Level 0. Fold # 2. Model # 0
[02:19:33] INFO Level 0. Fold # 2. Model # 0. Validation Score = 1.402926
[02:19:33] INFO Training Level 0 Fold # 3. Model # 0
[02:19:34] INFO Predicting Level 0. Fold # 3. Model # 0
[02:19:34] INFO Level 0. Fold # 3. Model # 0. Validation Score = 1.408808
[02:19:34] INFO Level 0. Model # 0. Mean Score = 1.412444. Std Dev = 0.009607
[02:19:34] INFO Training Level 0 Fold # 1. Model # 1
[02:19:40] INFO Predicting Level 0. Fold # 1. Model # 1
[02:19:40] INFO Level 0. Fold # 1. Model # 1. Validation Score = 1.475822
[02:19:40] INFO Training Level 0 Fold # 2. Model # 1
[02:19:45] INFO Predicting Level 0. Fold # 2. Model # 1
[02:19:45] INFO Level 0. Fold # 2. Model # 1. Validation Score = 1.444641
[02:19:45] INFO Training Level 0 Fold # 3. Model # 1
[02:19:51] INFO Predicting Level 0. Fold # 3. Model # 1
[02:19:51] INFO Level 0. Fold # 3. Model # 1. Validation Score = 1.463754
[02:19:51] INFO Level 0. Model # 1. Mean Score = 1.461406. Std Dev = 0.012837
[02:19:51] INFO Training Level 0 Fold # 1. Model # 2
```

```
# check error:
multiclass_logloss(y_cv_enc, preds[1])
```

```
1.5316193489752536
[02:19:51] INFO Training Level 0 Fold # 3. Model # 2
```

```
# Also check train error
preds = ens.predict(train_data_dict, lentest=xtrain_glove.shape[0])
multiclass_logloss(y_train_enc, preds[1])
```

```
[02:23:44] INFO Training Fulldata Level 0. Model # 0
[02:23:45] INFO Predicting Test Level 0. Model # 0
[02:23:45] INFO Training Fulldata Level 0. Model # 1
[02:23:52] INFO Predicting Test Level 0. Model # 1
[02:23:52] INFO Training Fulldata Level 0. Model # 2
[02:23:52] INFO Predicting Test Level 0. Model # 2
[02:23:52] INFO Training Fulldata Level 0. Model # 3
[02:23:52] INFO Predicting Test Level 0. Model # 3
[02:23:52] INFO Training Fulldata Level 1. Model # 0
[02:24:22] INFO Predicting Test Level 1. Model # 0
1.0485667689038733
[02:20:13] INFO Level 1. Fold # 1. Model # 0. Validation Score = 1.496972
```

```
#INFERENCE MODELLING:

from sklearn.externals import joblib

# Save the model as a pickle in a file
joblib.dump(ens, 'custom_ensembler.pkl')
joblib.dump(model, 'lstm.pkl')
joblib.dump(lr, 'lr2.pkl')
joblib.dump(tfv, 'tfidf.pkl')
```

```
['tfidf.pkl']
[02:21:02] INFO Predicting Test Level 0. Model # 2
```

```
# Utility Functions:
```

```
import re

### Dataset Preprocessing
from nltk.stem.porter import PorterStemmer
def preprocessor(sentence):
  ps = PorterStemmer()

  review = re.sub('[^a-zA-Z]', ' ', sentence)
  review = review.lower()
  review = review.split()

  review = [ps.stem(word) for word in review if not word in stopwords.words('english')]
  review = ' '.join(review)
  return review


def mapper(ans):
  for i in ans:
    if i==0:
      return 'Anger'
    elif i==1:
      return 'Disgust'
    elif i==2:
      return 'Fear'
    elif i==3:
      return 'Joy'
    elif i==4:
      return 'Neutral'
    elif i==5:
      return 'Sadness'
    elif i==6:
      return 'Surprise'




from sklearn.externals import joblib
def predictor(sentence):
  lst = []
  my_model = joblib.load('lr2.pkl')
  tfv = joblib.load('tfidf.pkl')

  sent = preprocessor(sentence)

  lst.append(sent)
  sent_tfv = tfv.transform(lst)
  ans = my_model.predict(sent_tfv)
  mapped_ans = mapper(ans)
  return mapped_ans


# Let's Predict
ans = predictor('I hate you from bottom of my heart')
print(ans)
```

```
Anger
```

```
ans = predictor('Oh Wow what a beautiful place')
print(ans)
```

```
Joy
```

```
ans = predictor('I am really sorry , i am feeling reaaly very sad')
print(ans)
```

```
Sadness
```

CONCLUSION:

The results of Infernece model are okay'ish since we used Logistic Regression + TFIDF which performed awesome. We see that ensembling improves the score by a great extent! LSTM can win this with higher number of epochs or altering the learnng rate.