Reddit classification

Problem statement: build a classifier that can determine which of ten categories a Reddit post belongs to:

- agreement
- announcement
- answer
- appreciation
- disagreement
- elaboration
- humor
- negative reaction
- other
- question

Goal is to build a classifier that will correctly predict classes above.

Metrics: since we have a multi class classification we will look at **f1-score** and confusion matrix for evaluation. F1 Score is the weighted average of Precision and Recall, it takes both false positives and false negatives into account. F1 is usually more useful than accuracy, especially if you have an uneven class distribution.
Confusion matrix makes it easier to understand how good are you predicting each class.

Data provided: reddit comments subset with:

- Original text
- Label (3 labels by different annotatoes)
- Depth of the message

Working environment: AWS EC2 g3s.xlarge instance

1. Data preparation

Target variable preprocessing:

Dataset comes with 3 labels per message, that were assigned by 3 labelers. To decide which one to use, we will take the most common out of 3. If all three are different we will take a random one.
What we can see after processing, we have a pretty unbalanced classes

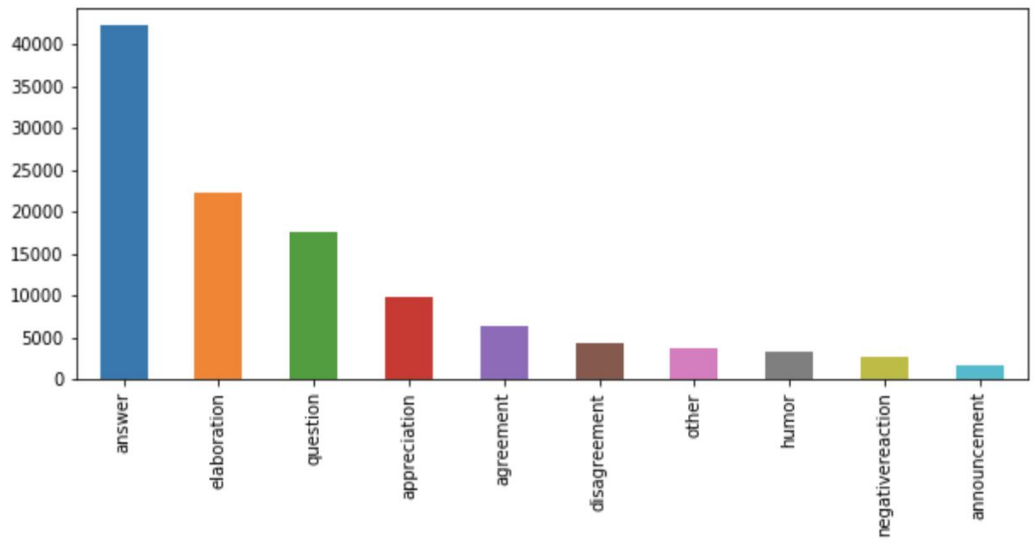


Fig1. Class distribution for target variable

Looks like we have something that we can call linear imbalance. Differences between answer and announcement labels are quite big, we will need to use some balancing techniques.

Unbalanced classes are mainly problematic because, we don't get optimized results for the unbalanced classes, because the model does not get a chance to look into them in detail. It makes it very difficult to make validation sets representative.

*Note: First thought was to take more samples from the available public dataset and make it task easier:) But I believe that the purpose of the test was to do it with this distribution, since normally you do not get good balanced data sets and rarely you get a chance to get more samples for needed classes without implementing tricks of some kind of balancing.

Input variables preprocessing:

We have raw text (messages) and numerical(depth) input variables, we will start with processing text and using text first.

1. Checking for missing values, 1274 rows do not have any text, therefore they will be removed.
2. Raw text processing

Normally text preprocessing includes a bunch of steps, it could be:

- removing unicode

- **remove url, html characters (**we might need to go back to it, since links tend to be in answer and elaboration category more than in others, which could be a special class feature. Though links are not exclusive to those classes**)**
- **removing special character @, # ect**
- **converting to lower**
- **removing numbers**
- removing stop words
- **removing punctuation**
- **removing white spaces**
- spell correction like ‘soooooo goooooood’ to 'so good', which could be interesting for our specific data set
- **converting shortening with apostrophe like she's to she is, which should not matter if we remove stop words, but we will end up with s, though this can also be remedied by removing all words with length less than 1**.
- slang removal (relevant especially for social media data)

Points in bold were used in the first iteration of text cleaning.
*Note: After exporting to csv we loose data for processed text for around 400 rows, this will need to be looked into.

2. Models and techniques

There are a lot of ML and DL models that can be used for multi class classification.
For our test purposes we will use:

- baseline : Tfidf + Naive Bayers (Multinomial variant is suitable for text and can serve as a good baseline model)
- SGDClassifier (stochastic gradient descent with linear support vector machine, which has a reputation of one of the very good text classificators)
- Bow+MLP+class_weights (Bag of words approach with multilayer Perceptron with adding class weights to address class imbalance)
- LSTM

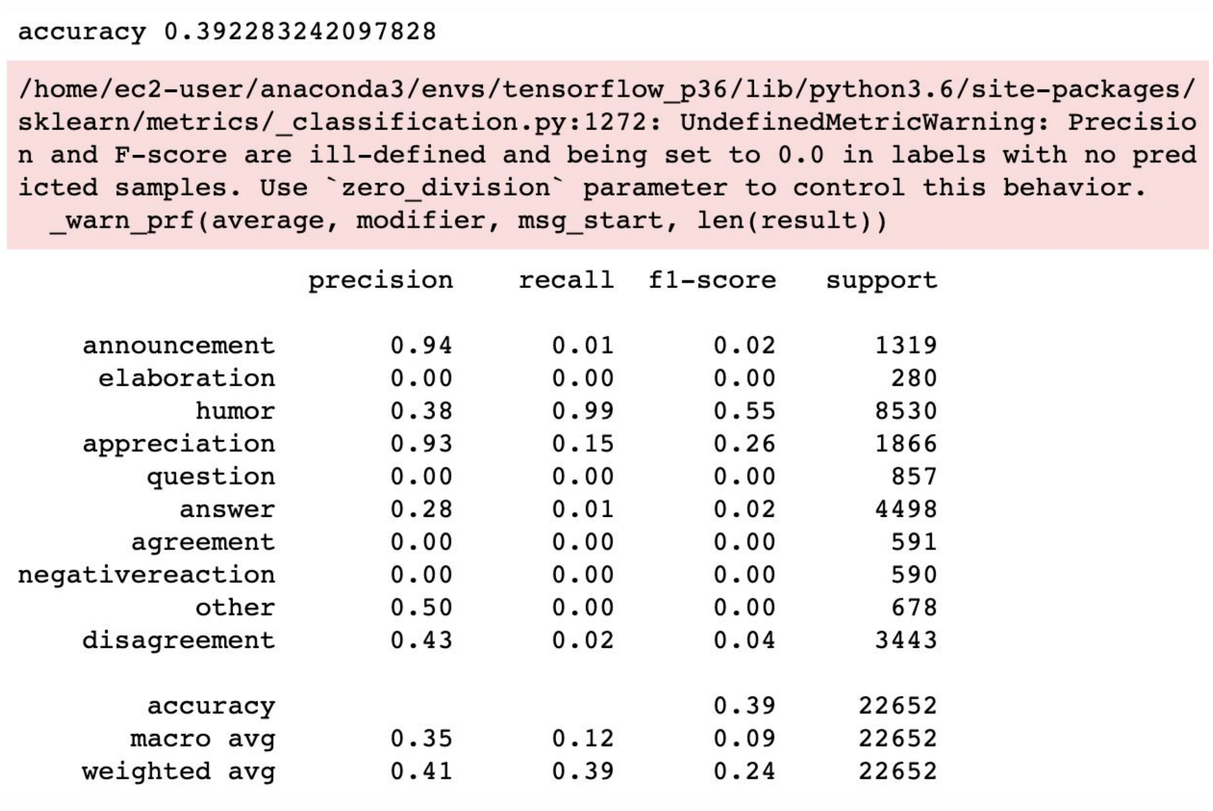
Processing for following models will include:

- Dividing to train/test sample
- Vectorizing text
- Encoding labels
- Limit dataset to n number of top words
- Set max number of words in each sequence

Last two were used only for neural nets.

Baseline (Tfidf + Naive Bayers)

Simple pipeline with CountVectorizer, TfidfTransformer and MultinomialNB were used.



```
accuracy 0.392283242097828
/home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python3.6/site-packages/
sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precisio
n and F-score are ill-defined and being set to 0.0 in labels with no pred
icted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
                  precision    recall  f1-score   support

    announcement       0.94      0.01      0.02      1319
     elaboration       0.00      0.00      0.00       280
           humor       0.38      0.99      0.55      8530
    appreciation       0.93      0.15      0.26      1866
        question       0.00      0.00      0.00       857
          answer       0.28      0.01      0.02      4498
       agreement       0.00      0.00      0.00       591
negativereaction       0.00      0.00      0.00       590
           other       0.50      0.00      0.00       678
    disagreement       0.43      0.02      0.04      3443

        accuracy                           0.39     22652
       macro avg       0.35      0.12      0.09     22652
    weighted avg       0.41      0.39      0.24     22652
```

Results that we got are very far from a good classifier, f1 score is 24%. I believe, unbalanced classes play a big part, our testing sample didnt even include all needed labels.
But it was quite expected.

SGDClassifier + linear Support Vector Machine

Pipeline with CountVectorizer, TfidfTransformer and SGDClassifier with loss='hinge', penalty='l2', random_state=500, max_iter=5, tol=None was used. Standart loss hinge was used, with default penalty l2,tol None max_iter_5

Results:

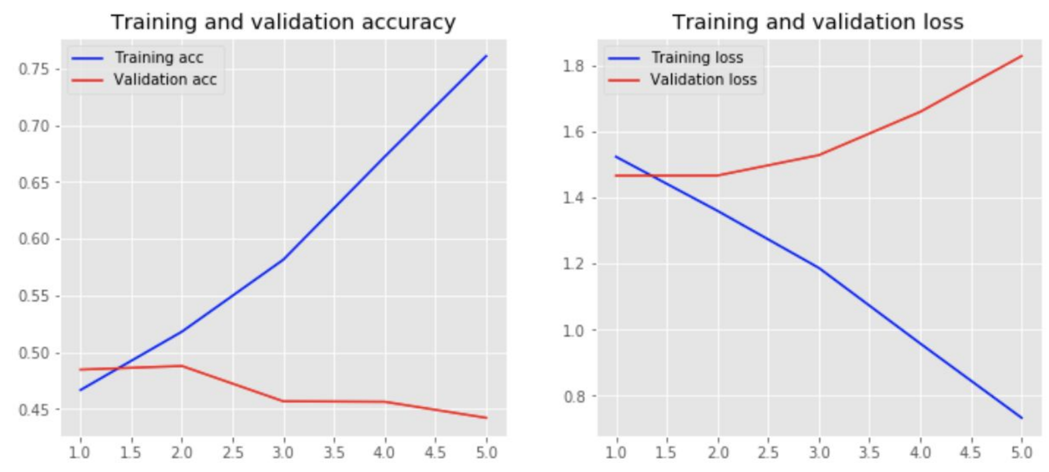
```
accuracy 0.46794984990287836
                 precision    recall  f1-score   support

    announcement       0.45      0.20      0.27      1319
     elaboration       0.06      0.02      0.03       280
           humor       0.49      0.82      0.61      8530
    appreciation       0.50      0.68      0.57      1866
        question       0.21      0.05      0.08       857
          answer       0.33      0.08      0.12      4498
       agreement       0.13      0.03      0.05       591
negativereaction       0.22      0.09      0.13       590
           other       0.22      0.06      0.09       678
    disagreement       0.46      0.46      0.46      3443

        accuracy                           0.47     22652
       macro avg       0.31      0.25      0.24     22652
    weighted avg       0.41      0.47      0.40     22652
```

F1 score improved to 40%, but still not a good result.
What we can do is to remedy this:

- redefine our cleaning steps
- try to adjust parameters
- try SMOTE for oversampling or undersampling
- try one-vs-rest scheme of classifying

BoW+Keras+class_weights

We will use a bag of words approach with multilayer Perceptron and we will add class weights to address class imbalance.
We will use max_words to use 1000,batch size-32, epochs-5,
F1 score hasn't improved, 41%



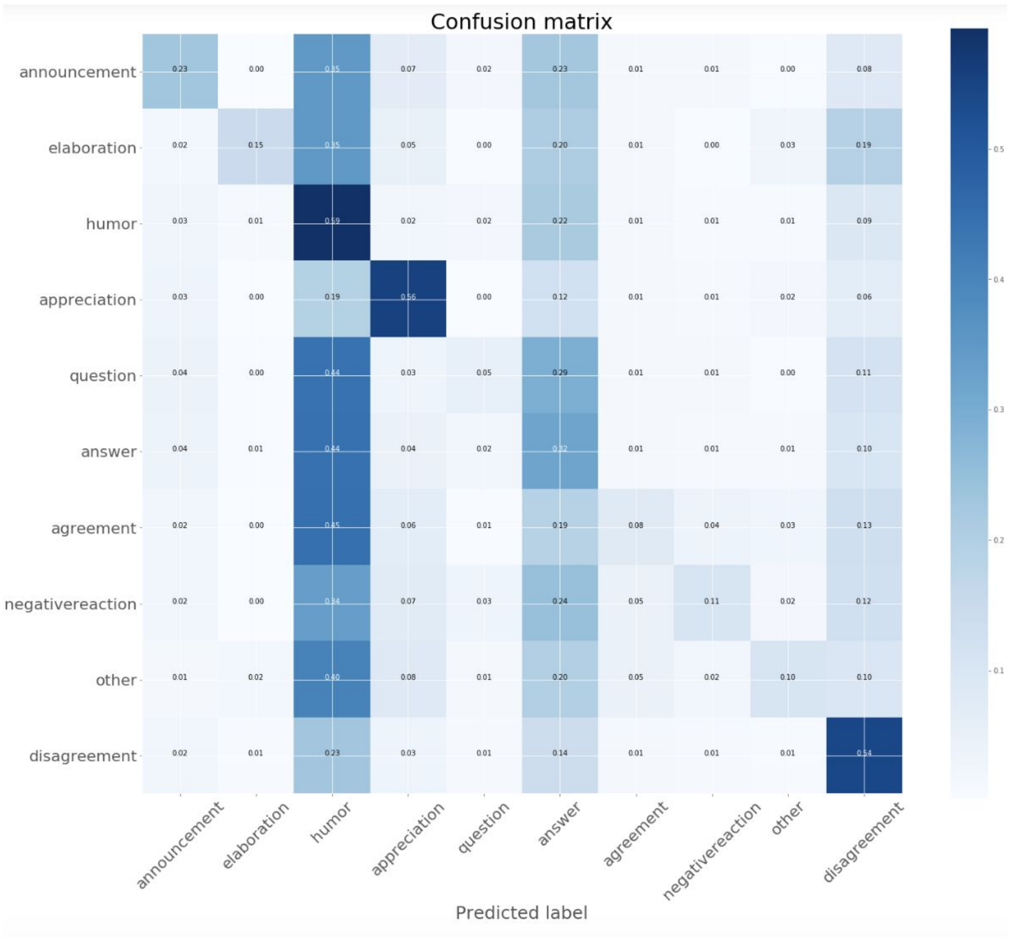
Training and validation loss and accuracy shows overfitting, which could be still caused by unbalanced classes.

Some parameters tuning could help us understand

Overall performance degraded, but confusion matrix shows that we are getting a bit better with generalizing for all classes, which means that adding weight was a good idea.

Model tends to predict humor and answer, which is very interesting.

But adding class weight will usually degrade overall performance as it is designed to allow increased loss on lower-weighted classes.



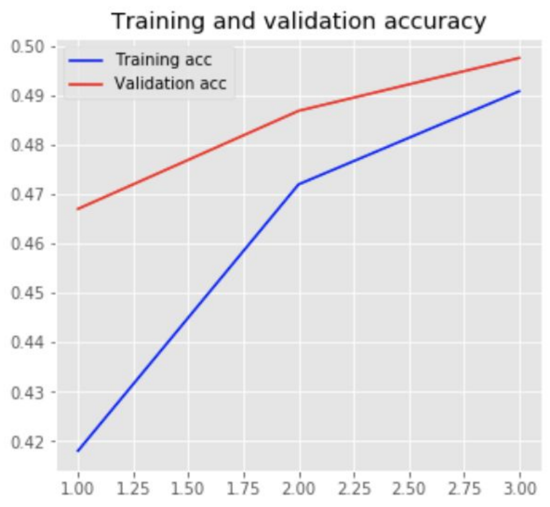
LSTM

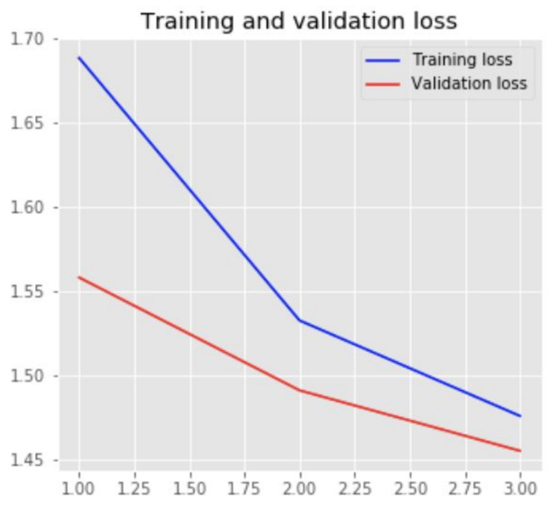
LSTM are popular with sequential data like text.

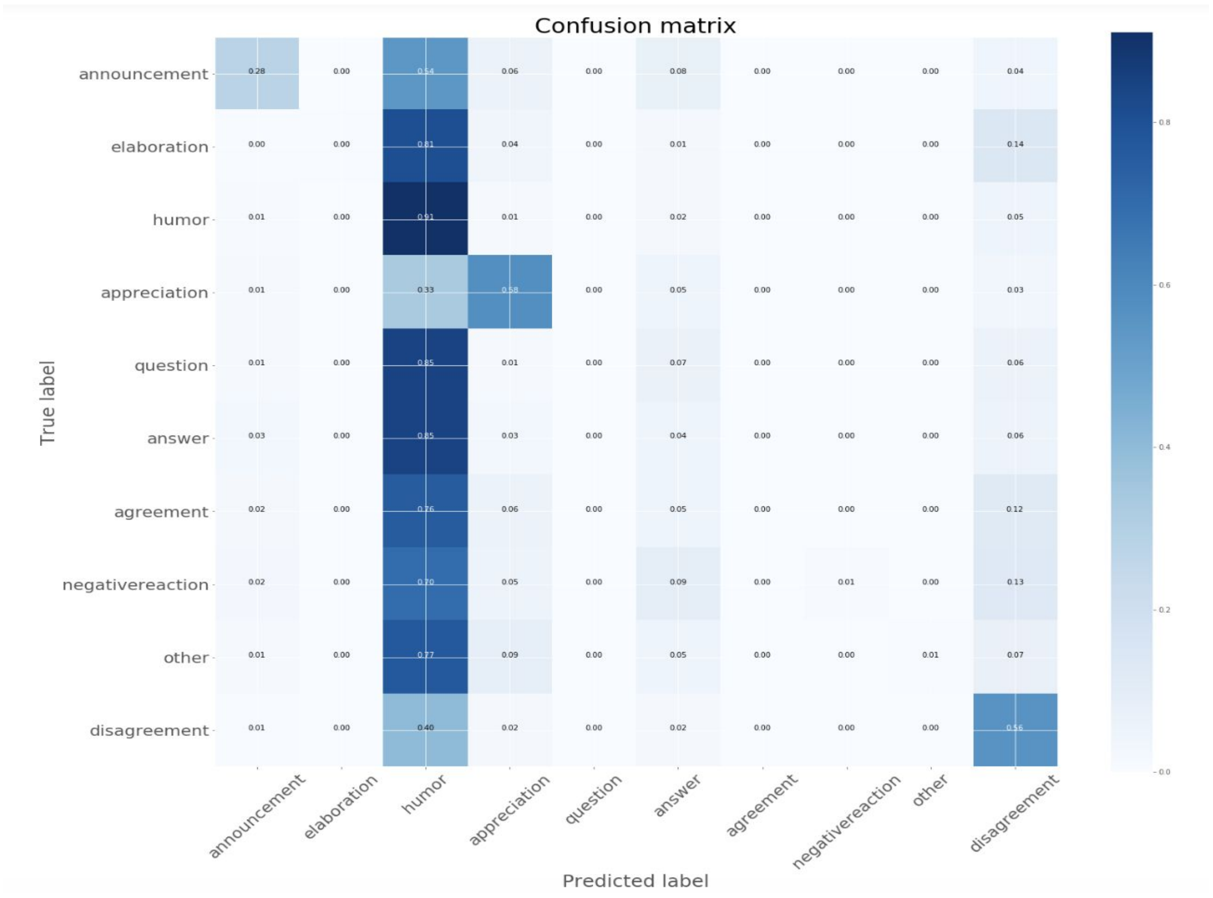
LSTM layer should go together with Dropout layer. It helps with overfitting, ignoring random neurons, this way reduces the sensitivity to the specific weights of individual neurons. 0.2 value is often used.

For multiple classes, generally, the softmax activation function works best, because it helps to interpret the outputs as probabilities. Normally, softmax comes with cross-entropy, we used categorical_crossentropy, since we have multiclass. Those two functions work well with each other because the cross-entropy helps to speed up learning process.For optimizer, adam, has been shown to work well in most practical applications and works well with only little changes in the hyperparameters.

We used, max_words = 1000, max_length   = 200 and dropout 0.5
With all expectations LSTM f1 score is better significantly better, 44%





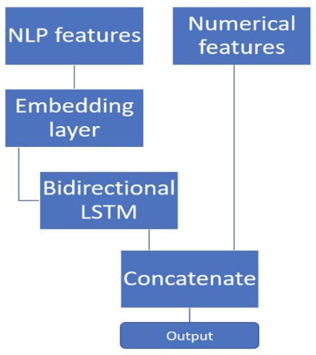


Training accuracy is higher probably due to dropout being high, makes model much more robust, but overall accuracy needs to get better.
Some hyperparameter experimentation was done: with max_word = 5000 and increasing epochs to 5.
Hyperparameter experimentation:

- Experiment with level of max_words, we used 1000,5000, we have some slang, misspellings etc
- Increase level of Max number of words in each message, we used 200, distribution of words per message, needs to be checked
- decrease dropout, we tried 0.5 and 0.2
- testing amount of epochs

Summary:
In this text classification won LSTM with 44% of f1 score, though there are a lot of room of improvement

What we can do to improve performance:

- Define other data cleaning steps, maybe keep links and do not remove question marks
- Use different vectorizers, like ngram level tf-idf or characters level tf-idf
- Use another embeddings
- Add depth level to modelling. Numerical features can be concatenated to text based features. For example, like this
- 

- Try deep CNN, character level CNN or Attention models
- Hyper parameters testing of max_words, epochs, adding dropout layers

Iterations gone bad

Taken steps were not described above, since they led to worse performance

- removing stopwords, cause drop in model accuracy
- Adding stemming, the same
- Using Smote for SGDClassifier. Oversampling, Undersampling with NearMiss surprisingly did not improve performance
- Using Glove embeddings also worsen performance, glove glove.6B.50d covered only 0.66% of vocabulary, which makes me think that there might be something wrong with preprocessing or my pipeline. First try went to glove, since it computer faster
- Using gensim word2vec embeddings worsen performance

Out of all tried models and
Some inspiration was taken from following papers:

- Bag of Tricks for Efficient Text Classification
- Characterizing Online Discussion Using Coarse Discourse Sequences
- Survey of resampling techniques for improving classification performance in unbalanced datasets

- A systematic study of the class imbalance problem in convolutional neural networks