

## Fake News Classification Task Explanations

Sayak Kundu

(This Task was done on Google Colab)

### File Structure

- ▼  kaggle
  - ▶  LIAR-PLUS
    -  GoogleNews-vectors-negative300.bin
    -  crawl-300d-2M.pkl
    -  glove.840B.300d.pkl
    -  pickled-crawl300d2m-for-kernel-competitio...
    -  pickled-glove840b300d-for-10sec-loading....
  - ▶  sample\_data
    -  binary.h5
    -  binary.json

### Binary Classification

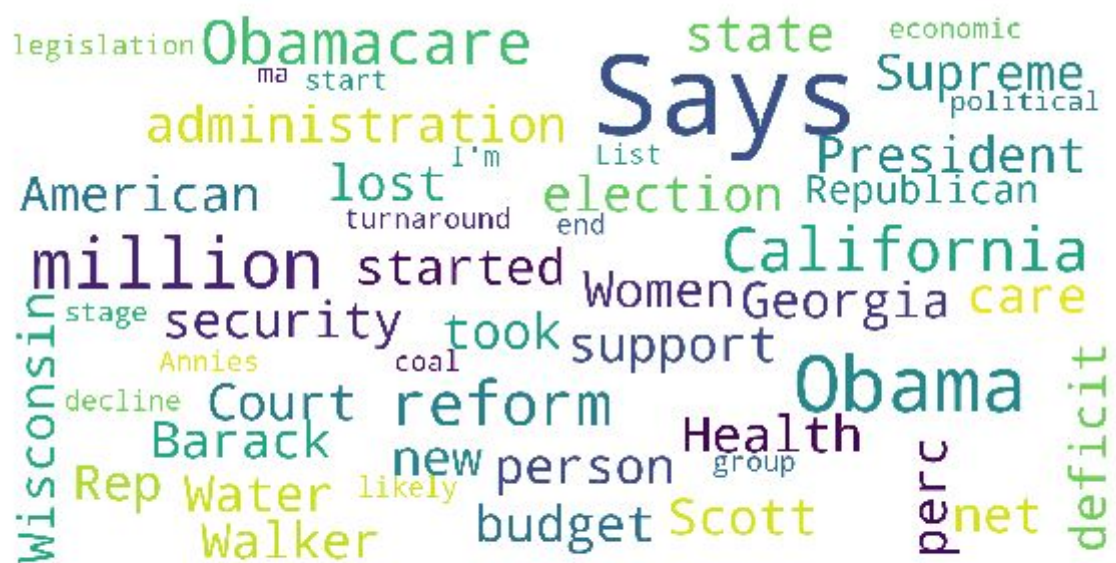
- ▼  kaggle
  - ▶  LIAR-PLUS
    -  GoogleNews-vectors-negative300.bin
    -  crawl-300d-2M.pkl
    -  glove.840B.300d.pkl
    -  pickled-crawl300d2m-for-kernel-competitio...
    -  pickled-glove840b300d-for-10sec-loading....
  - ▶  sample\_data
    -  multi.h5
    -  multi.json

### Six Way Classification



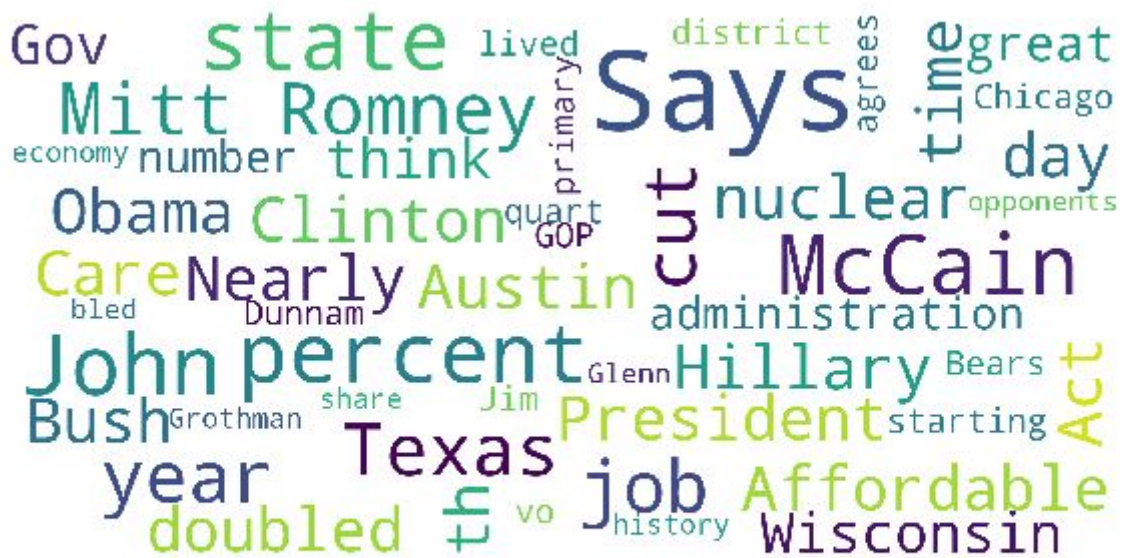
## EDA and Word Cloud

## Data Analysis



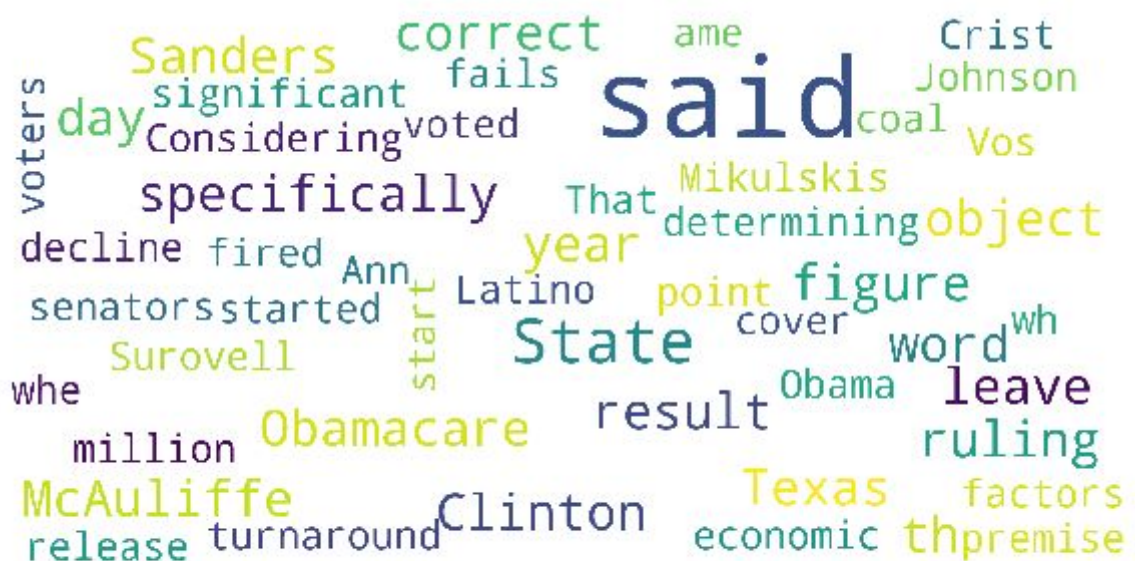
### Statement word cloud

Most common words for statement column, where label is 0



### Statement word cloud

Most common words for statement column, where label is 1

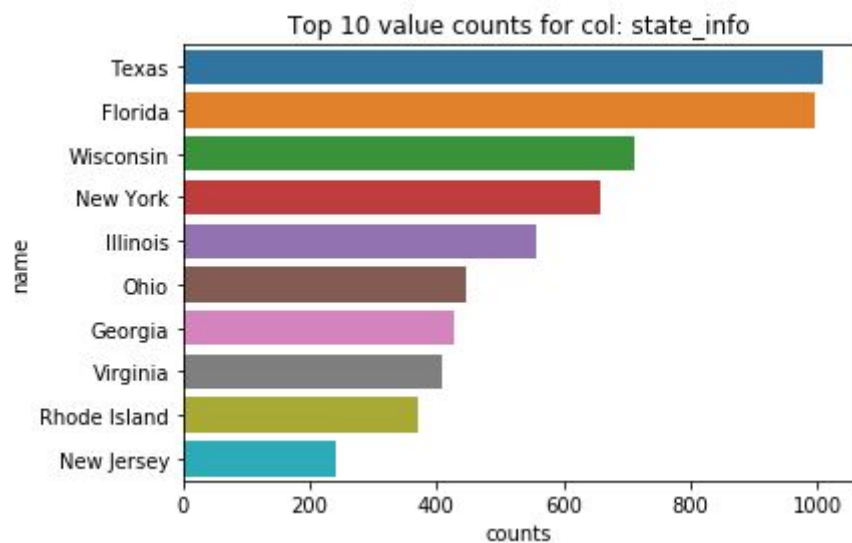
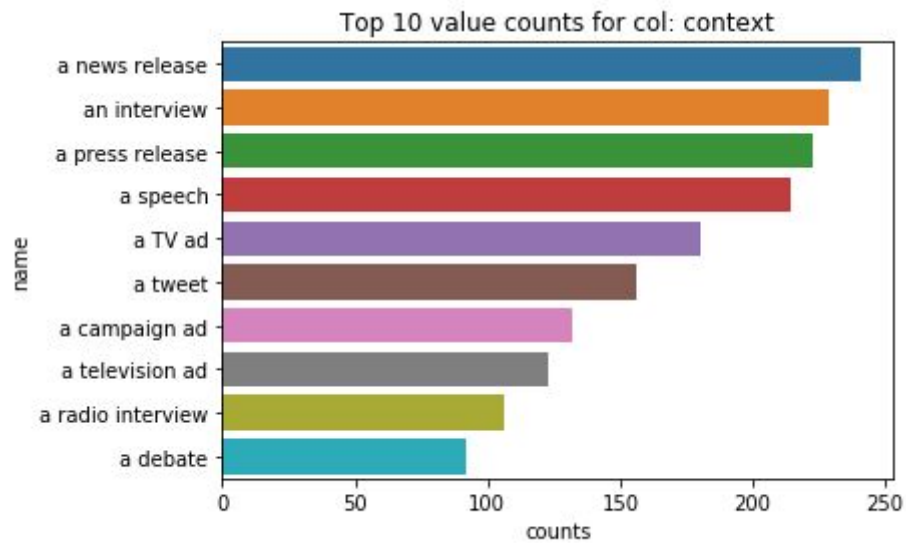
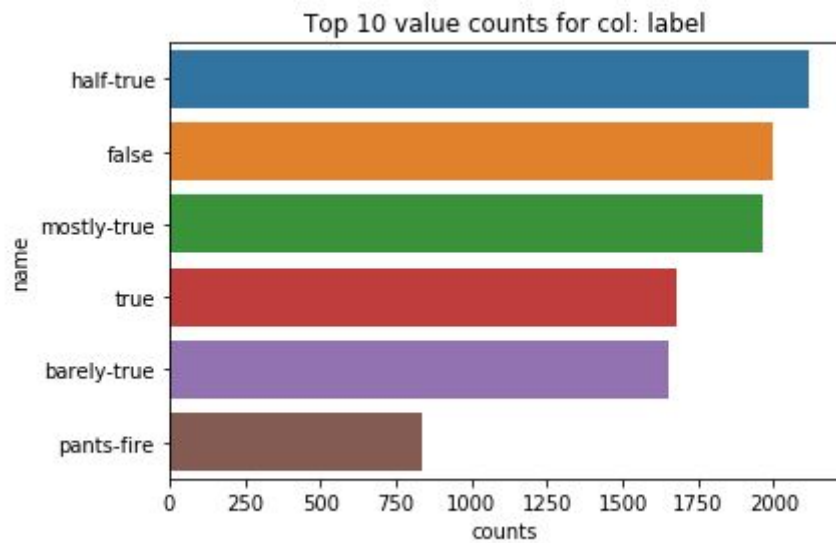


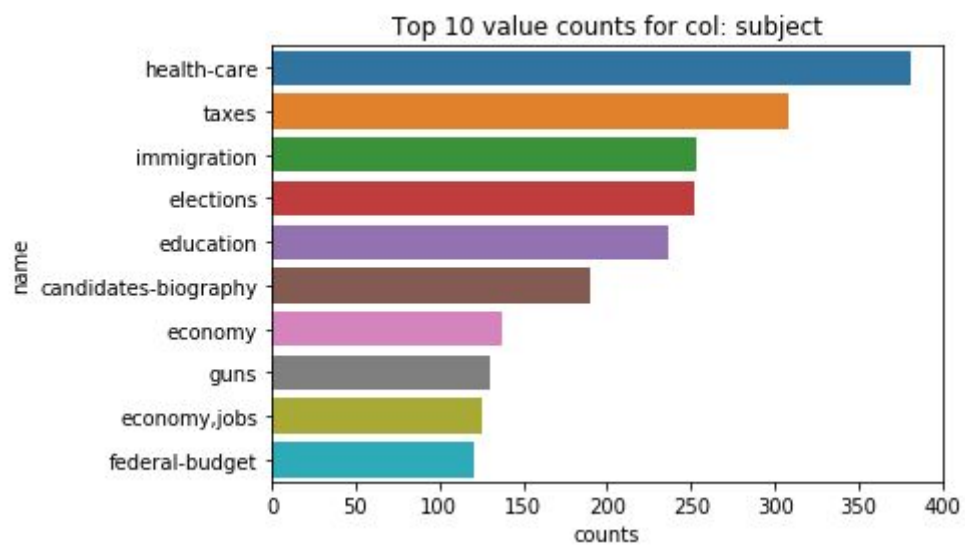
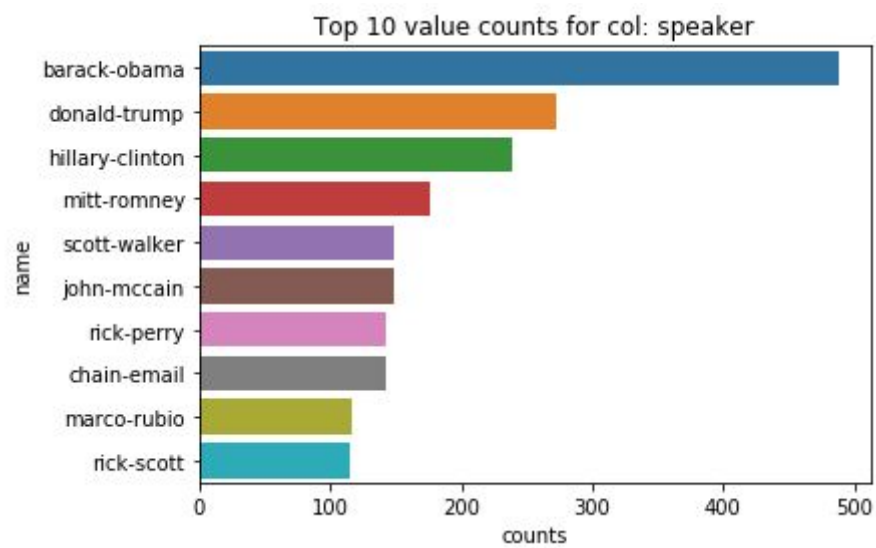
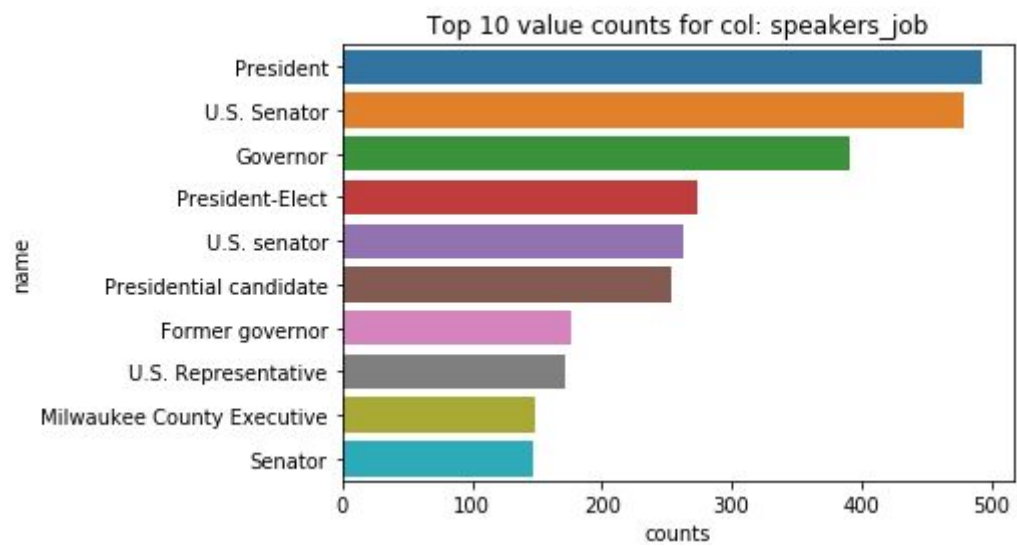
### Justification word cloud

### Most common words for justification column

Total number of unique values







## Preprocessing.

Same preprocessing methods were used both for multiclass and binary tasks.

For statement and justifications columns the following steps have been made:

- Removed some unnecessary symbols
- Isolated some symbols (He's -> He s)
- Remove quotation marks

```
from nltk.tokenize.treebank import TreebankWordTokenizer
def preprocess_data(train):
    t1 = time.time()

    isolate_dict = {ord(c):f' {c} ' for c in symbols_to_isolate}
    remove_dict = {ord(c):f'' for c in symbols_to_delete}

    def handle_punctuation(x):
        x = x.translate(remove_dict)
        x = x.translate(isolate_dict)
        return x

    train['statement'] = train['statement'].progress_apply(lambda x: handle_punctuation(x))
    train['justification'] = train['justification'].progress_apply(lambda x: handle_punctuation(x))

    tokenizer = TreebankWordTokenizer()
    def handle_contractions(x):
        x = tokenizer.tokenize(x)
        x = ' '.join(x)
        return x

    train['statement'] = train['statement'].progress_apply(lambda x: handle_contractions(x))
    train['justification'] = train['justification'].progress_apply(lambda x: handle_contractions(x))

    def fix_quote(x):
        x = [x_[1:] if x_.startswith('"') else x_ for x_ in x]
        x = ' '.join(x)
        return x

    train['statement'] = train['statement'].progress_apply(lambda x: fix_quote(x.split()))
    train['justification'] = train['justification'].progress_apply(lambda x: fix_quote(x.split()))

    print(f'Preprocess took {time.time() - t1}')
    return train
```

For the preprocessed text tokenization has been applied and after that, the crawl precalculated embedding matrix has been loaded for further use in the model.

For tabular data label encoding has been used for the following columns: 'subject', 'speaker', 'speakers\_job', 'state\_info', 'party\_aff', 'context'.

```

from sklearn.preprocessing import LabelEncoder
COLUMNS_TO_PROCESS = ['subject', 'speaker', 'speakers_job', 'state_info', 'party_aff', 'context']

def preprocess(d):
    data = d.copy()
    lc = LabelEncoder()
    for col in COLUMNS_TO_PROCESS:
        data[col] = data[col].fillna('Other')
        col_dict = data[col].value_counts()
        data[col].apply(lambda x: x if col_dict[x] > np.percentile(col_dict.values,
BASIC_FEAT_PERCENTILE) else 'Other')
        data[col] = lc.fit_transform(data[col])
    return data
data = preprocess(data)

```

Moreover, for each column only the labels that had been found more than 95<sup>th</sup> percentile of value counts for the concrete column have been used for Label Encoding, other values were label encoded as class “Other”.

Also, for the columns “mt\_cnt”, “f\_cnt”, “pt\_cnt”, “bt\_cnt”, “ht\_cnt” the value of the real target was subtracted from the corresponding column, as it was proposed in the paper “*Liar, Liar Pants on Fire: A New Benchmark Dataset for Fake News Detection*”

```

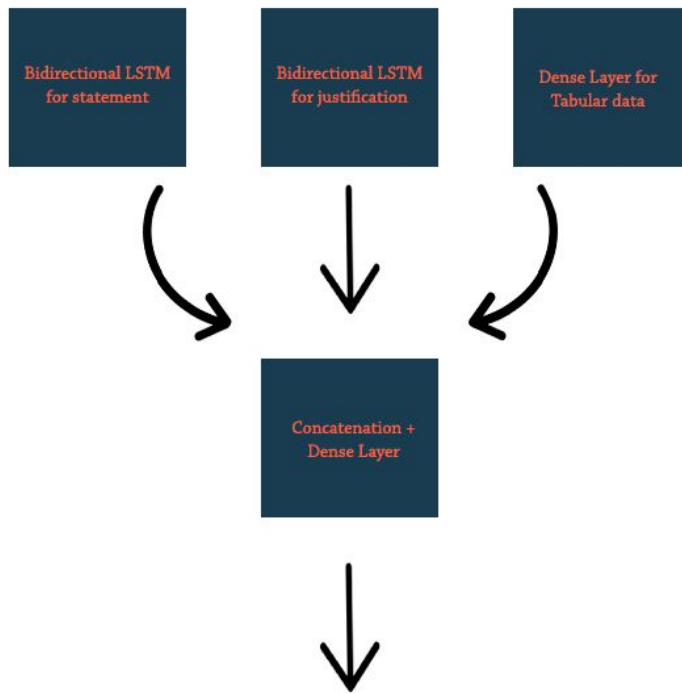
for index, value in enumerate(data['label']):
    if value == 'pants-fire':
        data.loc[index, 'pf_cnt'] -= 1
    elif value == 'false':
        data.loc[index, 'f_cnt'] -= 1
    elif value == 'half-true':
        data.loc[index, 'ht_cnt'] -= 1
    elif value == 'barely-true':
        data.loc[index, 'bt_cnt'] -= 1
    elif value == 'mostly-true':
        data.loc[index, 'mt_cnt'] -= 1

```

**Modelling for binary target.**



I have used a 3-input Neural Net architecture, which consists of the following parts:



The first input takes tokenized sequences for statement, proceeds them through embedding and Bidirectional LSTM layers. The second input does the same thing but with Justification data and the 3<sup>rd</sup> input processes the tabular data.

```

from keras.layers import Input, Dense, Embedding, SpatialDropout1D, add, concatenate
from keras.layers import CuDNNLSTM, Bidirectional, GlobalMaxPooling1D, GlobalAveragePooling1D
from keras.preprocessing import text, sequence
from keras.metrics import binary_accuracy
from keras.callbacks import LearningRateScheduler, ModelCheckpoint, EarlyStopping

LSTM_UNITS = 128
DENSE_HIDDEN_UNITS = 4 * LSTM_UNITS

def build_model(embedding_matrix, normal_feats_shape = 11):
    # ----- Statements network input
    statements = Input(shape=(None,))

    x1 = Embedding(*embedding_matrix.shape, weights=[embedding_matrix], trainable=False)(statements)
    # x1 = SpatialDropout1D(0.2)(x1)
    # x1 = Bidirectional(CuDNNLSTM(LSTM_UNITS, return_sequences=True))(x1)
    x1 = Bidirectional(CuDNNLSTM(LSTM_UNITS, return_sequences=True))(x1)

    hidden1 = concatenate([
        GlobalMaxPooling1D()(x1),
        GlobalAveragePooling1D()(x1),
    ])

    hidden1 = add([hidden1, Dense(DENSE_HIDDEN_UNITS, activation='relu')(hidden1)])
    hidden1 = add([hidden1, Dense(DENSE_HIDDEN_UNITS, activation='relu')(hidden1)])

    # ----- Justifications network input
    justifications = Input(shape = (None, ))

    x2 = Embedding(*embedding_matrix.shape, weights=[embedding_matrix], trainable=False)(justifications)
    x2 = SpatialDropout1D(0.2)(x2)
    x2 = Bidirectional(CuDNNLSTM(LSTM_UNITS, return_sequences=True))(x2)
    x2 = Bidirectional(CuDNNLSTM(LSTM_UNITS, return_sequences=True))(x2)

    hidden2 = concatenate([
        GlobalMaxPooling1D()(x2),
        GlobalAveragePooling1D()(x2),
    ])

    hidden2 = add([hidden2, Dense(DENSE_HIDDEN_UNITS, activation='relu')(hidden2)])
    hidden2 = add([hidden2, Dense(DENSE_HIDDEN_UNITS, activation='relu')(hidden2)])

    # ----- Other features network input
    norm_input = Input(shape = (normal_feats_shape, ))
    x3 = Dense(8, activation = 'sigmoid')(norm_input)

    # ----- Connecting all 3 branches in one
    hidden = concatenate([hidden1, hidden2, x3])
    result = Dense(1, activation='sigmoid')(hidden)

    model = Model(inputs=[statements, justifications, norm_input], outputs=[result])
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics = [binary_accuracy])

    return model

```

## Modelling for multiclass target.

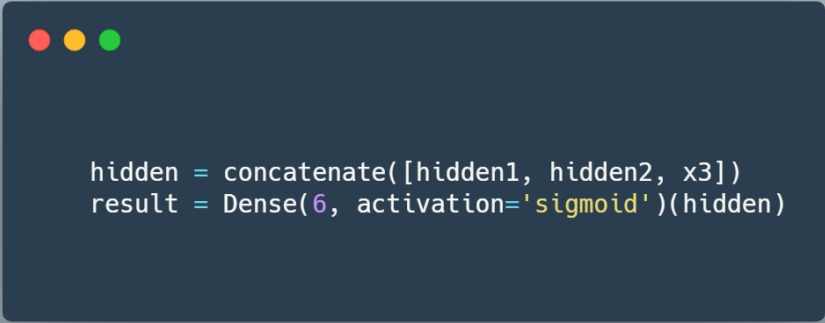
For multiclass problem the same model is being used, except the fact that I am using one hot encoded multiclass labels as the following:

```

from keras.utils import to_categorical
tr_st, tr_justi, tr_add, tr_label = tr['statement'], tr['justification'], tr.loc[:, 'subject':
'context'], to_categorical(tr.loc[:, 'label'])
val_st, val_justi, val_add, val_label = vl['statement'], vl['justification'], vl.loc[:, 'subject':
'context'], to_categorical(vl.loc[:, 'label'])
te_st, te_justi, te_add, te_label = te['statement'], te['justification'], te.loc[:, 'subject':
'context'], to_categorical(te.loc[:, 'label'])

```

The code for the model changes in the following way:



```
hidden = concatenate([hidden1, hidden2, x3])  
result = Dense(6, activation='sigmoid')(hidden)
```

So we are having now just a different dimension of the output.

#### **What has been tried but didn't work.**

I have also tried glove and paragram embeddings, their coverage for the justification and statement data was worse than for crawl embeddings, so I decided to stick to the later one.

I tried different number of epochs + different number of units for dense/LSTM layers, I compared their scores on validation and decided to stick to the values you could see now.

I have tried increasing/decreasing number of layers in the branches of neural networks , but the score was worse.

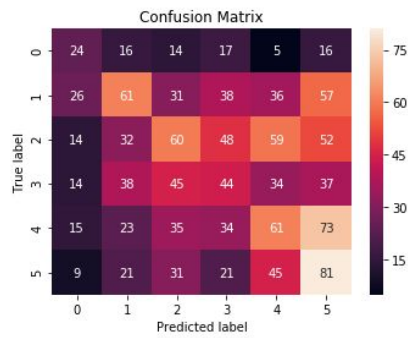
I have also tried adding some more regularization for the network, but the score was worse.

#### **Highest metrics achieved.**

Average ROC AUC score and Accuracy score for binary target are around 0.6, which means that the problem is a hard one for a model but there is some signal, that it was able to extract.

For the 6 class problem the accuracy was around 26-28%, which also means that there is some signal in the provided data.

Confusion matrix for multiclass problem:



### Libraries used.

The code was written in Python 3 with the usage of the following libraries:

Numpy, Pandas, Sklearn, nltk, keras, tensorflow, seaborn, matplotlib, wordcloud, gensim, pickle, colab, tqdm.

### Code sources.

<https://www.kaggle.com/christofhenkel/how-to-preprocessing-for-glove-part1-eda>

<https://www.kaggle.com/gpreda/jigsaw-eda>

<https://www.kaggle.com/thousandvoices/simple-lstm>