# CSE586.1 SPECIAL TOPICS (INFORMATION RETRIEVAL)

## Term Project Report

## OLID (Offensive Language Identification Dataset)

## Predicting the Type and Target of Offensive Posts in Social Media

Feyza Özkefe 20comp5008

**Abstract**

The OLID is a hierarchical dataset to identify the type and the target of offensive texts in social media. The dataset is collected on Twitter and publicly available. For each tweet, there are three levels of labels: (A) Offensive/Not-Offensive, (B) Targeted-Insult/Untargeted, (C) Individual/Group/Other. The relationship between them is hierarchical. If a tweet is offensive, it can have a target or no target. If it is offensive to a specific target, the target can be an individual, a group, or some other objects. This dataset is used in the OffensEval-2020 competition in SemEval-2020.

Multilingual Offensive Language Identification in Social Media, I propose two different techniques to compare results and get better representations and better leverage the information in the hierarchical dataset. The first of the models used is SVM and the second is bidirectional LSTM.

First of all, some pre-processes were made on the data set in common in both methods. These include removing "@USER" usernames from tweets (because there are too many of them in tweets), Replaced ampersands (&) with 'and', removed emojis,punctuation and etc.

### 1st Model – Support Vector Machines

#### Task A

A pipeline was created for the svm model and the TfidfVectorizer method was used as the vectorizer. Changed some parameters in TfidfVectorizer, one of them being ngram_range, unigram and bigram.

```
# SVM Model

svm_model = SVC(kernel="linear")
pipeline_svm = Pipeline([('vectorizer', TfidfVectorizer(sublinear_tf=True, min_df=5,
                                                         norm='l2', encoding='latin-1',
                                                         ngram_range=(1, 2), stop_words='english')),
                         ('classifier', svm_model)])


pipeline_svm.fit(X_train, y_train)
y_pred = pipeline_svm.predict(X_test)
```

**Fig:** SVM Model

```
model accuracy: 0.8267441860465117

              precision    recall  f1-score   support

           0       0.83      0.96      0.89       620
           1       0.81      0.49      0.61       240

    accuracy                           0.83       860
   macro avg       0.82      0.72      0.75       860
weighted avg       0.83      0.83      0.81       860
```

**Fig:** Task A results with SVM

The model predicts offensive tweets with less success than non-offensive tweets.

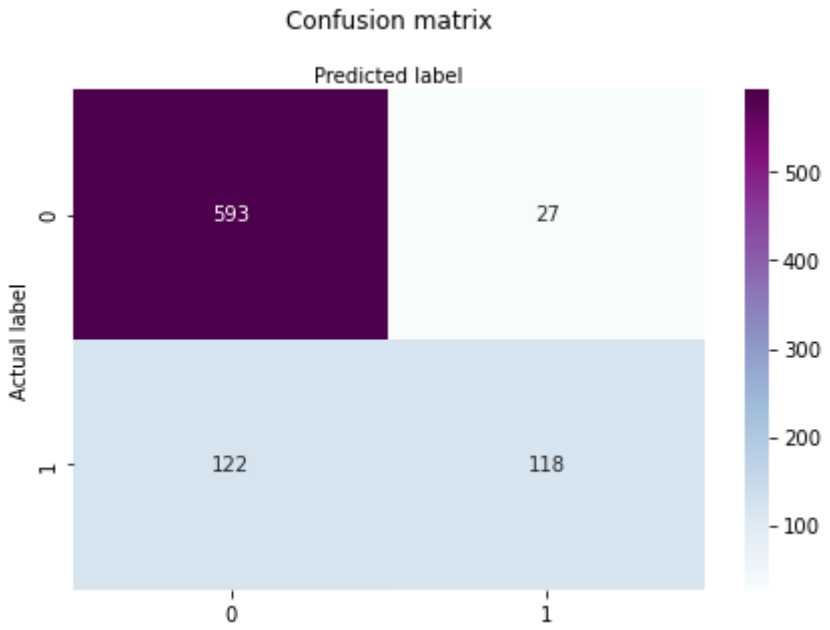


**Fig:** SVM Model Task A Confusion matrix

Here, OFF class is 1 and NOT class 0 is converted to binary values with label encoder. The confusion matrix of each task are plotted as follows.

**Task B**

The prediction process was continued with the values estimated as OFF for Task B. The task in this section is to make a new prediction as TIN or UNT with tweets predicted as OFF.

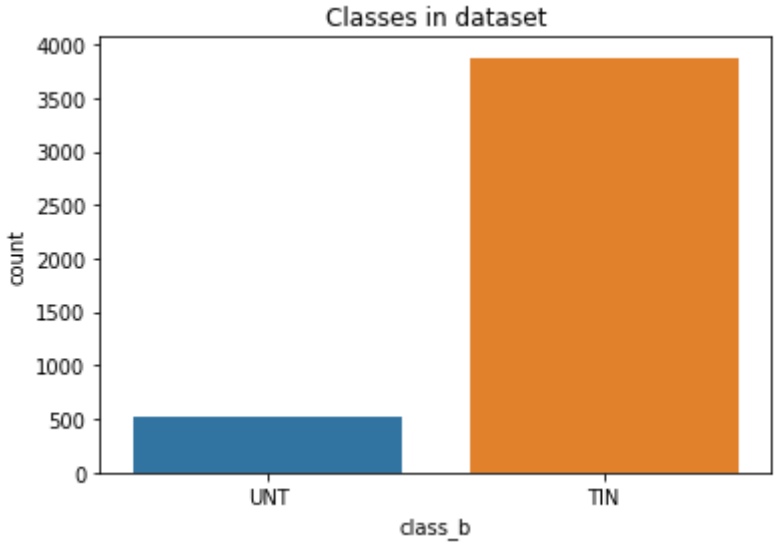


**Fig:** Task B Train Data

A rather imbalanced situation is observed in the data set. When the model is run with this data set, it cannot find the results of the UNT(0) class of the model. For this, undersampling process is applied on TIN(1) class.

```
model accuracy: 0.7622950819672131

              precision    recall  f1-score   support

           0       0.89      0.81      0.85        99
           1       0.41      0.57      0.47        23

    accuracy                           0.76       122
   macro avg       0.65      0.69      0.66       122
weighted avg       0.80      0.76      0.78       122
```

**Fig:** Task B Results with SVM

**Task C**

In task C, the values belonging to the TIN class are classified as IND, OTH, GRP. Accuracy precision recall and f1 scores were found by comparing the values found with the retest set. Likewise, a more balanced data set was obtained by applying undersampling and oversampling techniques for the imbalances in the C set. then training the model was done with this dataset.

```
model accuracy: 0.6265060240963856

              precision    recall  f1-score   support

           0       0.61      0.70      0.65        33
           1       0.64      0.68      0.66        37
           2       0.67      0.31      0.42        13

    accuracy                           0.63        83
   macro avg       0.64      0.56      0.58        83
weighted avg       0.63      0.63      0.62        83
```

**Fig:** Task C results with SVM

**2nd Model – Bidirectional LSTM Model**

The same preprocessing processes were applied in the second model. In addition, the following preprocessing processes were applied.

```
# TOKENIZER
max_features = 10000
embedding_dim = 128
max_len=500

tokenizer=Tokenizer(num_words=max_features,oov_token='</OOV>')
tokenizer.fit_on_texts(X_train.values)
dic=tokenizer.word_index
#print(dic)
```

Tokenizer method allows to vectorize a text corpus, by turning each text into either a sequence of integers (each integer being the index of a token in a dictionary) or into a vector where the coefficient for each token could be binary, based on word count, based on tf-idf...

Fit on text method updates internal vocabulary based on a list of texts. In the case where texts contains lists, we assume each entry of the lists to be a token. Required before using texts_to_sequences.

Finally, train validation and test sets were converted to 2d numpy array of sequences.

```
# TRAIN
X_train_seq = tokenizer.texts_to_sequences(X_train.values)
X_train_pad = pad_sequences(X_train_seq, maxlen=max_len)

print("train data tensor:" ,X_train_pad.shape)
```

The model was created sequentially. An embedding layer stores one vector per word. When called, it converts the sequences of word indices to sequences of vectors. These vectors are trainable. After training (on enough data), words with similar meanings often have similar vectors.

A recurrent neural network (RNN) processes sequence input by iterating through the elements. RNNs pass the outputs from one timestep to their input on the next timestep. Bidirectional wrapper can also be used with an RNN layer. This propagates the input forward and backwards through the RNN layer and then concatenates the final output. The main advantage of a bidirectional RNN is that the signal from the beginning of the input doesn't need to be processed all the way through every timestep to affect the output. The main disadvantage of a bidirectional RNN is that you can't efficiently stream predictions as words are being added to the end.

After the RNN has converted the sequence to a single vector the two layers.Dense do some final processing, and convert from this vector representation to a single logit as the classification output.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 500, 128)          1280000
_________________________________________________________________
bidirectional (Bidirectional (None, 256)               263168
_________________________________________________________________
dense (Dense)                (None, 128)               32896
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 1,576,193
Trainable params: 1,576,193
Non-trainable params: 0
_________________________________________________________________
```

BinaryAccuracy was used as the accuracy metric for the first two tasks, and CategoricalAccuracy was used for the last task. When compiling the model, binary_crossentropy for the first two models and categorical_crossentropy for the last task. 'Man' was used in each model as the optimizer. The results of the models are shown below.

**Task A**

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.81 | 0.85 | 0.83 | 620 |
| 1 | 0.56 | 0.50 | 0.52 | 240 |
| accuracy | | | 0.75 | 860 |
| macro avg | 0.68 | 0.67 | 0.68 | 860 |
| weighted avg | 0.74 | 0.75 | 0.74 | 860 |

**Fig:** Task A Results with LSTM

**Task B**

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.89 | 0.79 | 0.84 | 105 |
| 1 | 0.24 | 0.41 | 0.30 | 17 |
| accuracy | | | 0.74 | 122 |
| macro avg | 0.57 | 0.60 | 0.57 | 122 |
| weighted avg | 0.80 | 0.74 | 0.76 | 122 |

**Fig:** Task B Results with LSTM

**Task C**

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.44 | 0.75 | 0.55 | 32 |
| 1 | 0.55 | 0.44 | 0.49 | 39 |
| 2 | 0.00 | 0.00 | 0.00 | 15 |
| accuracy | | | 0.48 | 86 |
| macro avg | 0.33 | 0.40 | 0.35 | 86 |
| weighted avg | 0.41 | 0.48 | 0.43 | 86 |

**Fig:** Task C Results with LSTM

**Comparison of the Two Models**

Comparisons were made with the macro average F1 scores of each model.

When we make a comparison between the two models, it is observed that the SVM model gives a more successful result.

| **Model** | **Task A** | **Task B** | **Task C** |
|---|---|---|---|
| **SVM** | 0.83 | 0.76 | 0.63 |
| **LSTM** | 0.75 | 0.74 | 0.48 |

Especially in the third task, we see that the svm model achieves a much more successful result. When the confusion matrices are compared, we notice that the LSTM model does not make any predictions for OTH class 0, on the contrary, SVM was able to make a prediction for this class.

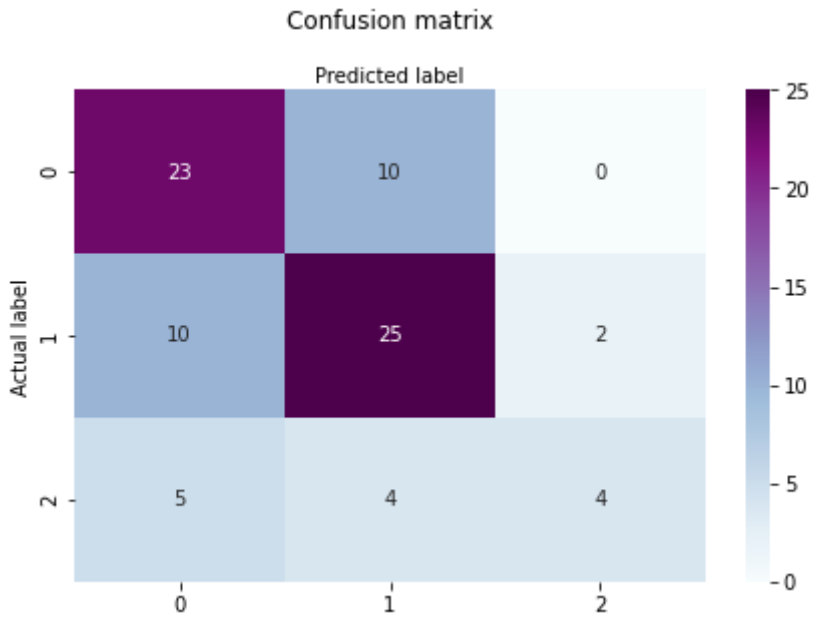


**Fig:** Task C confusion matrix with SVM

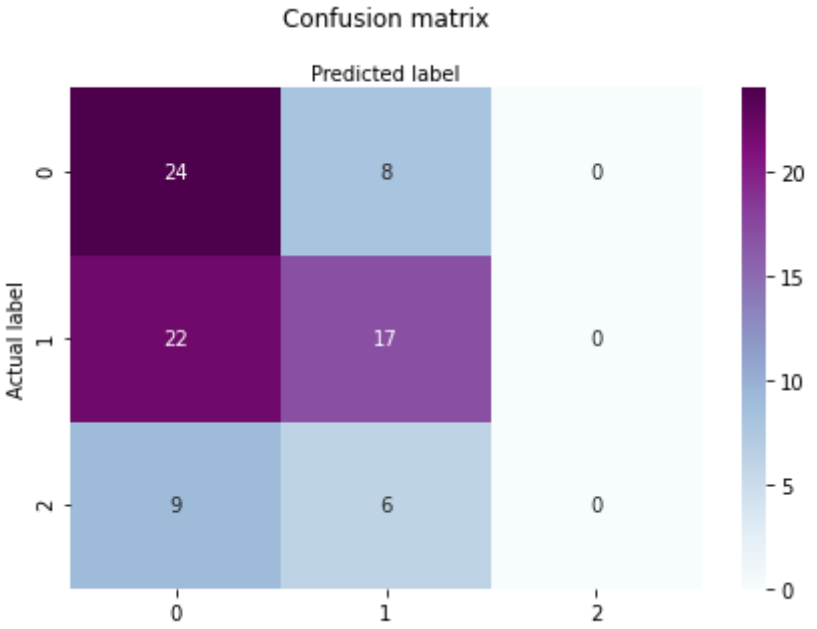


**Fig:** Task C confusion matrix with LSTM