FINAL PROJECT REPORT

Course:
DATS 6312 - Natural Language Processing

Instructor:
Dr. Amir Jafari

Authors:
Nusrat Nawshin, Kartik Jain, Siddharth Das

Semester:
Fall 2022

Date:
12th December 2022

CONTENTS

OVERVIEW

In this project, we work with a dataset of articles labeled as either sarcastic or non-sarcastic news. We have implemented various modeling strategies to classify the news headlines into two categories using models ranging from classical models incorporating TFIDF Vectorizer to complex transformer-based models in PyTorch and compared their performance. We have also trained a text summarizer on the body of the news articles and compared it with the original headline for the subset of sarcastic news stories. Finally, we documented our results and tried to get insights into our models.

INTRODUCTION AND DATA SOURCES

We obtained the dataset from Kaggle, News Headlines Dataset For Sarcasm Detection. There are a total of 55,328 news headlines with articles. Past studies in Sarcasm Detection mostly use Twitter datasets collected using hashtag-based supervision, but such datasets are noisy in terms of labels and language. To overcome the limitations related to noise in Twitter datasets, the authors collected this News Headlines dataset for Sarcasm Detection from two news websites. *TheOnion* produces sarcastic versions of current events, and collects all the headlines from News in Brief and News in Photos categories which are sarcastic. It collects real-world and non-sarcastic news headlines from *HuffPost* [1]. This dataset consists of three attributes:

- **is_sarcastic**: 1 if the record is sarcastic, else 0
- **headline**: the headline of the news article
- **article_link**: link to the original news article

	article_link	headline	is_sarcastic
0	https://www.huffingtonpost.com/entry/versace-b...	former versace store clerk sues over secret 'b...	0
1	https://www.huffingtonpost.com/entry/roseanne-...	the 'roseanne' revival catches up to our thorn...	0
2	https://local.theonion.com/mom-starting-to-fea...	mom starting to fear son's web series closest ...	1
3	https://politics.theonion.com/boehner-just-wan...	boehner just wants wife to listen, not come up...	1
4	https://www.huffingtonpost.com/entry/jk-rowlin...	j.k. rowling wishes snape happy birthday in th...	0

Figure 1: Original dataset

For summarization, we scraped the article bodies from the internet using the BeautifulSoup python package. The final dataset is in the cloud, so the process does not need to be repeated multiple times.

	article_link	headline	is_sarcastic	body
2	https://local.theonion.com/mom-starting-to-fea...	mom starting to fear son's web series closest ...	1	WHITE PLAINS, NY—With still no indication that...
3	https://politics.theonion.com/boehner-just-wan...	boehner just wants wife to listen, not come up...	1	—Amid the continuing debate over the upcoming ...
8	https://politics.theonion.com/top-snake-handle...	top snake handler leaves sinking huckabee camp...	1	LITTLE ROCK, AR—Dealing yet another blow to th...
15	https://entertainment.theonion.com/nuclear-bom...	nuclear bomb detonates during rehearsal for 's...	1	NEW YORK—In yet another setback for the $65 mi...
16	https://www.theonion.com/cosby-lawyer-asks-why...	cosby lawyer asks why accusers didn't come for...	1	LOS ANGELES—Responding to recent allegations t...

Figure 2: Web Scrapped dataset

DATA PREPROCESSING

We performed a few text-processing operations for the classification task. It included removing the numbers, lowering the characters, removing stop words, and lemmatization on the news headlines. This following regular expression removed numbers, punctuations, and other noisy characters from the headlines:

```
final_df['text'] = [re.sub(r'[^a-zA-Z]', ' ', str(sentence)) for sentence in final_df['text']]
```

The headline characters are then converted to lower-case to reduce the noise.

```
final_df['text'] = [sentence.lower() for sentence in final_df['text']]
```

Stop words are removed from the headlines using the NLTK package to remove low-level information and focus on the relevant tokens.

```
lst = []
for sentence in final_df['text']:
    lst.append([i for i in str(sentence).split() if i not in (stopwords.words('english'))])

final_df['text'] = [' '.join(sent) for sent in lst]
```

Finally, we lemmatized the headline texts.

```
wnl = nltk.WordNetLemmatizer()
lst = []
for sentence in final_df['text']:
    lst.append(' '.join(wnl.lemmatize(word) for word in sentence.split()))

final_df['text'] = lst
```

Lemmatization is a text normalization technique that converts words to their root mode. The context of the text may be helpful in our classification problem. Lemmatization performs better than stemming the word as it has higher accuracy at getting the root of the word. We used the NLTK WordNetLemmatizer method for lemmatization.

CLASSIFICATION

Classical models

We use classical models like Random Forest Classifier and Multilayer Perceptron for the classification task. We also implemented text preprocessing on the dataset but discovered that the model gave better results on the raw vectorized text. We transform the headlines using the TFIDF vectorizer before feeding them into the model. Term frequency (TF) and Inverse document frequency (IDF) are based on the Bag of Words (BoW) model, which contains insights about the less occurring and higher occurring words in a document. However, it has a limitation as it does not capture the semantic meaning of the words [2]. After vectorization, we pass it to the Random Forest and MLP classifiers, both of them perform well with 100% train accuracy and 95% test accuracy.

```
Training RF classifier
RF Train Accuracy score: 0.9999774072567891
RF Train F1-score: 0.9999753530673108
RF Test Accuracy score: 0.959515633471896
RF Test F1-score: 0.9553873730332602

Training MLP classifier
MLP Train Accuracy score: 1.0
MLP Train F1-score: 1.0
MLP Test Accuracy score: 0.9602385685884692
MLP Test F1-score: 0.9562798092209858
```

Figure 3: Classical Models output

MLP model using GloVe embeddings

Multi-layer perceptron (MLP) is a supplement of a feed-forward neural network. It consists of the input layer, output layer, and hidden layer. In the input layer, we are using GloVe embedding vectors. GloVe is an unsupervised learning algorithm for obtaining vector representations for words. It trains on aggregated global word-word co-occurrence statistics from a corpus, and the resulting embedding showcases interesting linear substructures of the word vector space [3]. The embedding output is

then passed in a linear layer to get the output layer. We use the Adam optimizer with a learning rate of 0.001 with cross-entropy loss as the loss function.

```
class TextClassificationModel(nn.Module):
    def __init__(self, vocabs, embed_dim, num_classes, weights, trainable=False):
        super(TextClassificationModel, self).__init__()
        self.embedding = nn.EmbeddingBag(vocabs, embed_dim)
        self.embedding.load_state_dict({'weight': torch.FloatTensor(weights)})
        if trainable:
            self.embedding.weight.requires_grad = True
        else:
            self.embedding.weight.requires_grad = False
        self.fc = nn.Linear(embed_dim, num_classes)
        self.init_weights()

    def init_weights(self):
        initrange = 0.5
        self.embedding.weight.data.uniform_(-initrange, initrange)
        self.fc.weight.data.uniform_(-initrange, initrange)
        self.fc.bias.data.zero_()

    def forward(self, text, offsets):
        embedded = self.embedding(text, offsets)
        return self.fc(embedded)
```

Figure 4: MLP with GloVe embedding model

With batch size 64 and training the model up to 10 epochs it gives 0.99 training accuracy and 0.96 testing accuracy. We can also specify whether we want the model to be able to train on weights for words that are not recognized by the embedding to get a better score. If we freeze the weights, we get the following results:

```
| epoch  10 |   500/  692 batches | accuracy    0.652
-----------------------------------------------------------
| end of epoch  10 | time:  2.43s | valid accuracy    0.655
-----------------------------------------------------------
```

Figure 5: MLP+GloVe model output with frozen embedding weights

There are 25539 available words and 2680 unavailable words in the GloVe dictionary with reference to the vocabulary for our dataset. If we make the weights trainable, we get the following results:

```
-----------------------------------------------------------
| epoch  10 |   500/  692 batches | accuracy    0.996
-----------------------------------------------------------
| end of epoch  10 | time:  3.13s | valid accuracy    0.961
-----------------------------------------------------------
```

Figure 6: MLP+GloVe model output with unfrozen embedding weights

RNN cells

Recurrent Neural Network(RNN) is a type of Neural Network where the output from the previous step is fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words [4]. We used RNN cells with the same layout as that of the MLP scripts and got similar results.

```python
class TextClassificationModel(nn.Module):
    def __init__(self, vocabs, embed_dim, hidden_dim, num_classes, weights, trainable=False):
        super(TextClassificationModel, self).__init__()
        self.embedding = nn.EmbeddingBag(vocabs, embed_dim)
        self.embedding.load_state_dict({'weight': torch.FloatTensor(weights)})
        if trainable:
            self.embedding.weight.requires_grad = True
        else:
            self.embedding.weight.requires_grad = False
        self.rnn = nn.RNN(embed_dim, hidden_dim)
        self.fc = nn.Linear(hidden_dim, num_classes)
        self.init_weights()

    def init_weights(self):
        initrange = 0.5
        self.embedding.weight.data.uniform_(-initrange, initrange)
        self.rnn.weight_ih_l0.data.uniform_(-initrange, initrange)
        self.rnn.weight_hh_l0.data.uniform_(-initrange, initrange)
        self.rnn.bias_ih_l0.data.uniform_(-initrange, initrange)
        self.rnn.bias_hh_l0.data.uniform_(-initrange, initrange)
        self.fc.weight.data.uniform_(-initrange, initrange)
        self.fc.bias.data.zero_()

    def forward(self, text, offsets):
        embedded = self.embedding(text, offsets)
        output, _ = self.rnn(embedded)
        return self.fc(output)
```

Figure 7: RNN with GloVe embedding model

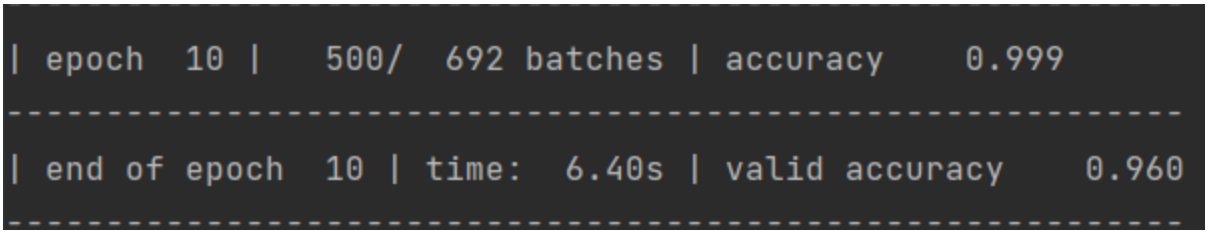


Figure 8: RNN with GloVe model output with unfrozen embedding weights

Transformers - BERT + MLP

BERT, which stands for Bidirectional Encoder Representations from Transformers, is based on transformers, a deep learning model in which every output element connects to every input element, and the weights between them are dynamically calculated based upon their connection, i.e., attention. [5] BERT base has 12 layers in the encoder with 768 hidden feedforward layers and 110 million parameters. This model takes the ‘CLS’ token as input first, followed by a sequence of words as input. Here CLS is a classification token. The inputs then pass to the above layers. [6] Each layer applies self-attention and passes the result through a feedforward network, after which it hands off to the next encoder. The model outputs a vector of hidden size 768, which is then passed in two linear layers with GELU as an activation function. We use the AdamW optimizer for optimization with learning rate 1e-6 with Cross Entropy Loss as a loss function as the target is binary. AdamW is a stochastic optimization method that modifies the typical implementation of weight decay in Adam by decoupling weight decay from the gradient update. [7] The model has a total of 108606338 trainable parameters.

Model:

```python
def __init__(self, dropout=0.3):
    super(BertClassifier, self).__init__()

    self.bert = BertModel.from_pretrained('bert-base-cased')
    self.dropout = nn.Dropout(dropout)
    self.linear1 = nn.Linear(768, 384)
    self.gelu1 = nn.GELU()
    self.dropout = nn.Dropout(dropout)
    self.linear2 = nn.Linear(384, 2)
    self.gelu2 = nn.GELU()

def forward(self, input_id, mask):
    _, pooled_output = self.bert(input_ids=input_id, attention_mask=mask, return_dict=False)
    dropout_output = self.dropout(pooled_output)
    linear_output1 = self.linear1(dropout_output)
    first_layer = self.dropout(self.gelu1(linear_output1))
    linear_output = self.linear2(first_layer)
    final_layer = self.gelu2(linear_output)
    return final_layer
```

Figure 9: BERT-MLP classification model

After training it with batch size 8 and till 4 epochs, the training accuracy is 0.97 and the test accuracy is 0.94.

```
Training on cuda
100%|██████████| 5533/5533 [48:36<00:00,  1.90it/s]
Epochs: 1 | Train Loss:  0.063 | Train Accuracy:  0.755 | Val Loss:  0.050 | Val Accuracy:  0.834
100%|██████████| 5533/5533 [48:35<00:00,  1.90it/s]
Epochs: 2 | Train Loss:  0.038 | Train Accuracy:  0.890 | Val Loss:  0.039 | Val Accuracy:  0.888
100%|██████████| 5533/5533 [48:35<00:00,  1.90it/s]
Epochs: 3 | Train Loss:  0.021 | Train Accuracy:  0.952 | Val Loss:  0.031 | Val Accuracy:  0.923
100%|██████████| 5533/5533 [48:37<00:00,  1.90it/s]
Epochs: 4 | Train Loss:  0.011 | Train Accuracy:  0.979 | Val Loss:  0.027 | Val Accuracy:  0.936
Testing on cuda
Test Accuracy:  0.949
```

Figure 10: BERT-MLP classification model output

Transformers - RoBERTa + MLP

RoBERTa model is built on BERT and modifies key hyperparameters, removing the next-sentence pretraining objective and training with much larger mini-batches and learning rates. In BERT architecture, the masking is performed once during data preprocessing, resulting in a single static mask. Training data is duplicated and masked ten times to avoid using a single static mask, each time with a different mask strategy over 40 epochs, thus having four epochs with the same mask. This strategy is not the

same as dynamic masking, in which a different mask is generated every time we pass data into the model [7]. The model outputs a matrix of max sentence length 103 by hidden size 768. This gets passed through a max pooling layer that aggregates over the time dimension. It is then passed in a linear layer with LogSoftmax as an activation function. The Adam optimizer is used with learning rate 1e-5 with KL - Divergence as the loss function. Although it is a binary classification problem, the labels were one-hot encoded and compared with the maximum logit value. This returns a better score as seen from the validation accuracy.

Model:

```
def __init__(self, dropout=0.2):
    super(TransformerClassifier, self).__init__()
    self.trans = RobertaModel.from_pretrained('roberta-base')
    self.hidden_size = self.trans.config.hidden_size
    self.maxpool = nn.MaxPool1d(MAX_LEN)
    self.linear = nn.Linear(self.hidden_size, 2)
    self.log_softmax = nn.LogSoftmax(dim=1)


def forward(self, input_id, mask):
    encoded_layers = self.trans(input_ids=input_id, attention_mask=mask, return_dict=False)[0]
    encoded_layers = encoded_layers.permute(0, 2, 1)
    maxpool_output = self.maxpool(encoded_layers).squeeze(2)
    linear_output = self.linear(maxpool_output)
    linear_output = self.log_softmax(linear_output)
    return linear_output
```

Figure 11: RoBERTa-MLP classification model

After training it with batch size 32 for six epochs, the training accuracy is 0.99, and we obtained a test accuracy of 0.986. However, we were not able to reproduce our results and subsequent runs gave us a test accuracy of around 0.96.

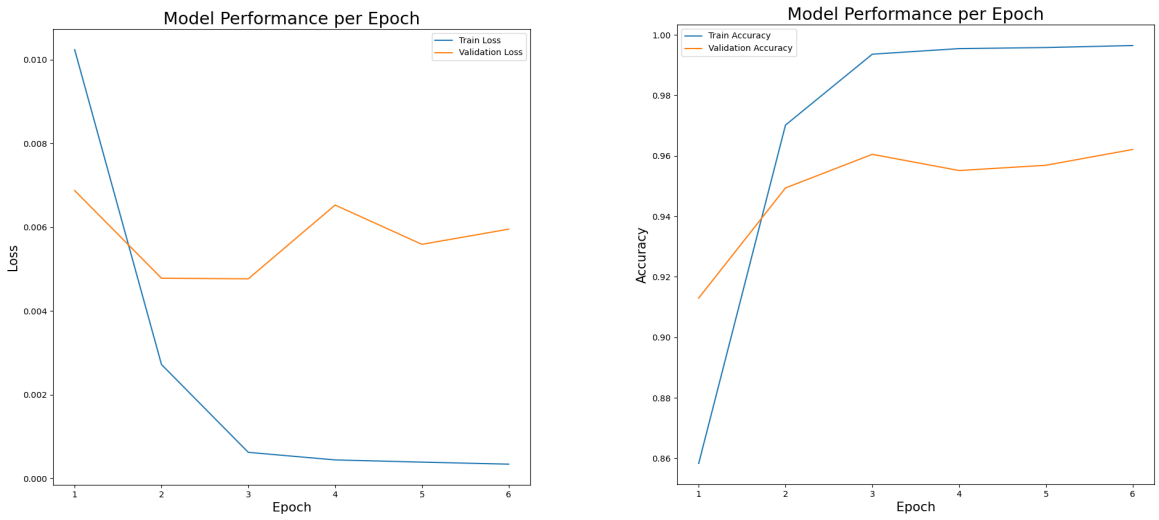


Figure 10: Loss and accuracy for Roberta model

Results

Classification Models Results:

Model	Optimizer	Epoch	Batch Size	max_len	Learning Rate	Train Accuracy	Test Accuracy
Random Forest	-	-	-	-	-	0.99	0.95
MLP	-	-	-	-	-	1.00	0.96
MLP + GloVe Embeddings	Adam	10	64	50	0.001	0.99	0.95
RNN + GloVe embeddings	Adam	10	64	50	0.001	0.99	0.96
BERT + MLP	AdamW	4	8	256	1e-6	0.97	0.94
RoBERTa	Adam	6	32	107	1e-5	0.99	0.962

SUMMARIZATION

Huggingface API - Pretrained models

- T5-small-headline-generator

T5 is an encoder-decoder model pre-trained on a multi-task mixture of unsupervised and supervised tasks and for which each task is converted into a text-to-text format. The input sequence is fed to the model using input IDs. It shifts the target sequence to the right, i.e., prepended by a start-sequence token, and feeds to the decoder using the decoder input IDs. In the teacher-forcing style, the target sequence is appended with the EOS token and corresponds to the labels. The PAD token is the start-sequence token. [8] We use the T5-small-headline-generator pretrained model for summarization, which is a t5-small model fine-tuned for headline generation. Sarcastic news article bodies are passed as the input to be summarized and compared to the news headlines. The batch size is 16, and it's trained to 25 epochs. We use the AdamW optimizer with a learning rate of 2e-5.

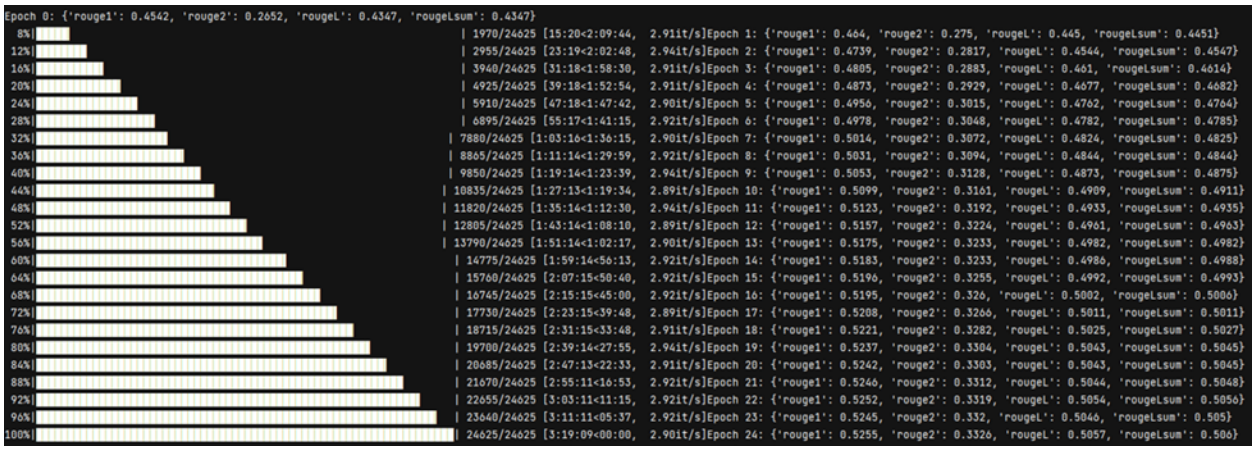


Figure 11: Summarization T5-small-headline-generator scores

The summarization gives all ROUGE scores of more than 50. ROUGH stands for Recall-Oriented Understudy for Gisting Evaluation which is essentially a set of metrics

for evaluating the automatic summarization of texts. [10] This task works by comparing the automatically produced summary against a set of reference summaries i.e., the news headlines. As our target headlines are just one sentence, we use lead-1 baseline to set the baseline scores instead of lead-3. It returns the first line of the text as the summary.

Baseline Scores on validation set:

Rouge1 : 0.305	RougeL : 0.280
Rouge2 : 0.151	RougeLsum : 0.280

Final ROUGE scores:

Rouge1 = 0.5255	RougeL = 0.5057
Rouge2 = 0.3326	RougeLsum = 0.506

Summarized Outputs:

Example 1:

Article: SAN FRANCISCO—In an effort to stop the spread of the dangerous negativity as quickly as possible, agents from the Department of Labor's Emergency Response Team on Friday reportedly sealed off the toxic workplace environment at tech incubator Molloy Capital. We've begun the hazardous but absolutely necessary task of cleaning up the site's dangerous backbiting, unwanted sexual advances, and gross managerial incompetence said Labor Department special agent Angela Lasker, who noted that prior to their arrival, a steady stream of inappropriate comments had been seeping into the water-cooler conversation and were threatening to poison the entire company-wide intranet. While we believe we've contained most of the professional and psychological damage, it's still a very hostile, unsafe area. We honestly don't think it will be able to sustain new hires for the foreseeable future, or at least until the pervasive low morale has had a chance to dissipate. Lasker went on to say, however, that the communications department would likely never again be suitable for human employment.

Headline: department of labor response team seals off toxic workplace environment

Summarized Headline: department of labor seals off toxic workplace environment

Example 2:

Article: DENTON, TX—Stressing that they were there solely to purchase gasoline and use the bathroom, if necessary, area dad Mike Whitcomb clarified while pulling into a travel plaza Thursday that this was not a food stop. We're here to get gas and that's it, Whitcomb said emphatically, adding that his three children were welcome to get out and stretch their legs, but they had better be back in their seats and buckled up by the time he finished filling the tank because he wasn't waiting around. I want to be back on the road in five minutes. If you're hungry, you can have one of the apples your mom brought. Sources later confirmed area mother Debra Whitcomb had okayed one bag of Chex Mix for everyone to share.

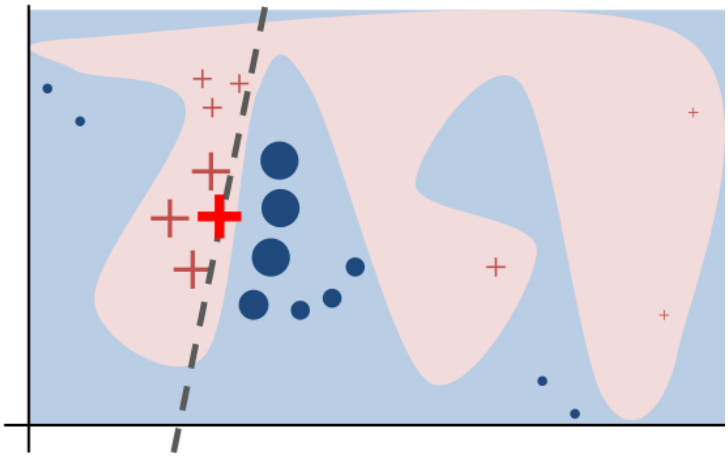
Headline: dad clarifies this not a food stop

Summarized Headline: area dad clarifies this not a food stop

EXPLAINABILITY

What are explanations?

Intuitively, an explanation is a local linear approximation of the model's behaviour. While the model may be very complex globally, it is easier to approximate it around the vicinity of a particular instance. While treating the model as a black box, we perturb the instance we want to explain and learn a sparse linear model around it, as an explanation. The figure below illustrates the intuition for this procedure. The model's decision function is represented by the blue/pink background, and is clearly nonlinear. The bright red cross is the instance being explained (let's call it X). We sample instances around X, and weight them according to their proximity to X (weight here is indicated by size). We then learn a linear model (dashed line) that approximates the model well in the vicinity of X, but not necessarily globally



We use the LIME package to interpret the model.

Lime is able to explain any black box classifier, with two or more classes. All we require is that the classifier implements a function that takes in raw text or a numpy array and outputs a probability for each class. Support for scikit-learn classifiers is built-in.

Here are a few examples of LIME model explanability

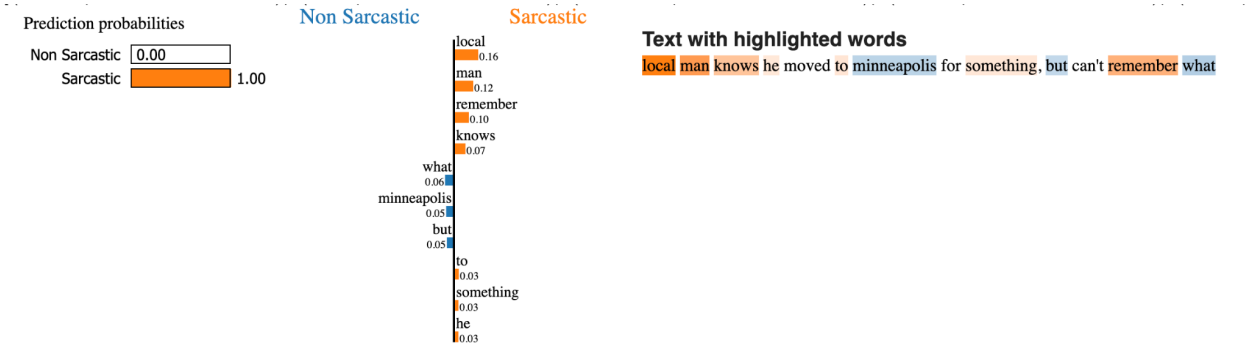


Fig1

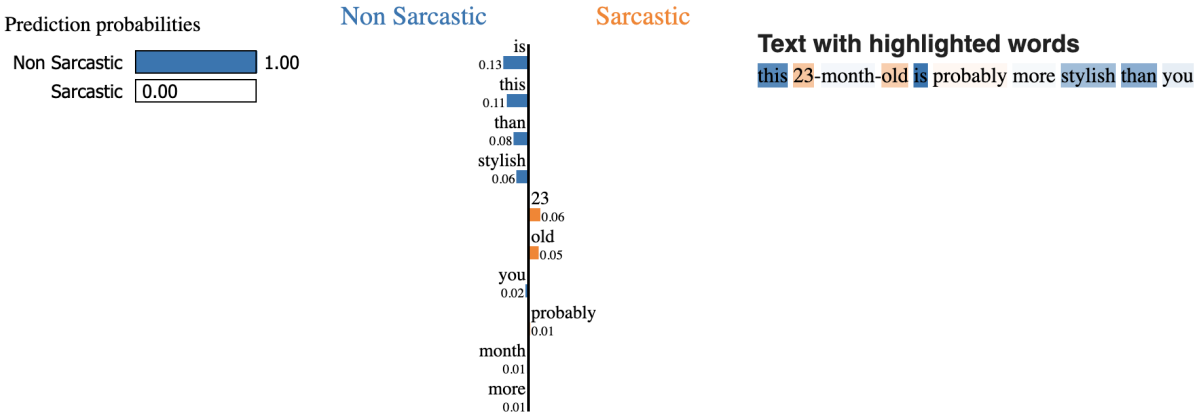


Fig2

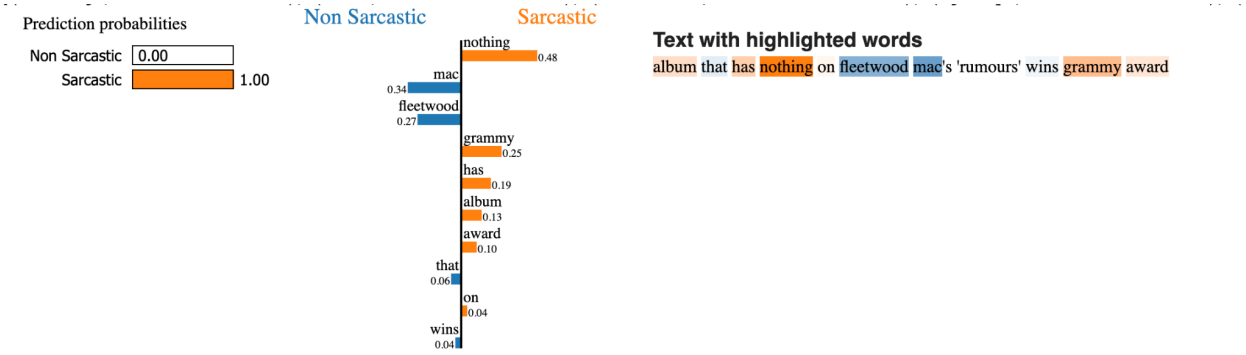


Fig3

Negative (blue) words indicate Non Sarcastic, while positive (Orange) words indicate Sarcastic. The way to interpret the weights by applying them to the prediction probabilities. For example, if we remove the words nothing and granny from the document, we expect the classifier to predict Sarcastic with probability 1-0.48-0.25

CONCLUSION

By comparing the results of our experiments we found that among all the models RoBERTA model is best at predicting the sarcastic news headline with 99% and 98% training and testing accuracy respectively. Along with that the classical models are also performing very well as well. Among all the tested models, BERT pretraing followed by MLP model is performing the least at the training and testing. For sarcastic headline predictions by applying text summarization technique using Huggingface T5-small-headline-generator, which is specifically trained on news headlines dataset is predicting sarcastic headlines with 50% ROUGE scores. In the future work, we intend to improve the architecture to give results faster. Limitation on the computing power restricted our attempts on better model building by modeling on higher hyperparameters like batch size, optimizers and bigger transfer models.

PERCENTAGE OF CODE WRITTEN

The codes were referred from Hugging Face official sites and documentations. Approximately 70% of the code was referred from the internet and modified according to the needs of our project. The remaining 30% was done to create helper functions or automating certain processes like downloading pretrained models, etc.

REFERENCES

[1] https://www.kaggle.com/datasets/rmisra/news-headlines-dataset-for-sarcasm-detection

[2] Text Vectorization: Term Frequency — Inverse Document Frequency (TFIDF) | by Vaibhav Jayaswal | Towards Data Science

[3] https://nlp.stanford.edu/projects/glove/

[4] https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

[5] What is BERT (Language Model) and How Does It Work?

[6] https://www.geeksforgeeks.org/explanation-of-bert-model-nlp/

[7] https://paperswithcode.com/method/adamw

[8] https://www.geeksforgeeks.org/overview-of-roberta-model/

[9] An intro to ROUGE, and how to use it to evaluate summaries (freecodecamp.org)

[10] JulesBelveze/t5-small-headline-generator · Hugging Face

[11] https://huggingface.co/course/chapter7/5?fw=pt

[12] https://github.com/amir-jafari/NLP