UNIVERSITY OF COPENHAGEN

Fake News Articles and How to Find Them

Final Project

Applied Machine Learning 2023

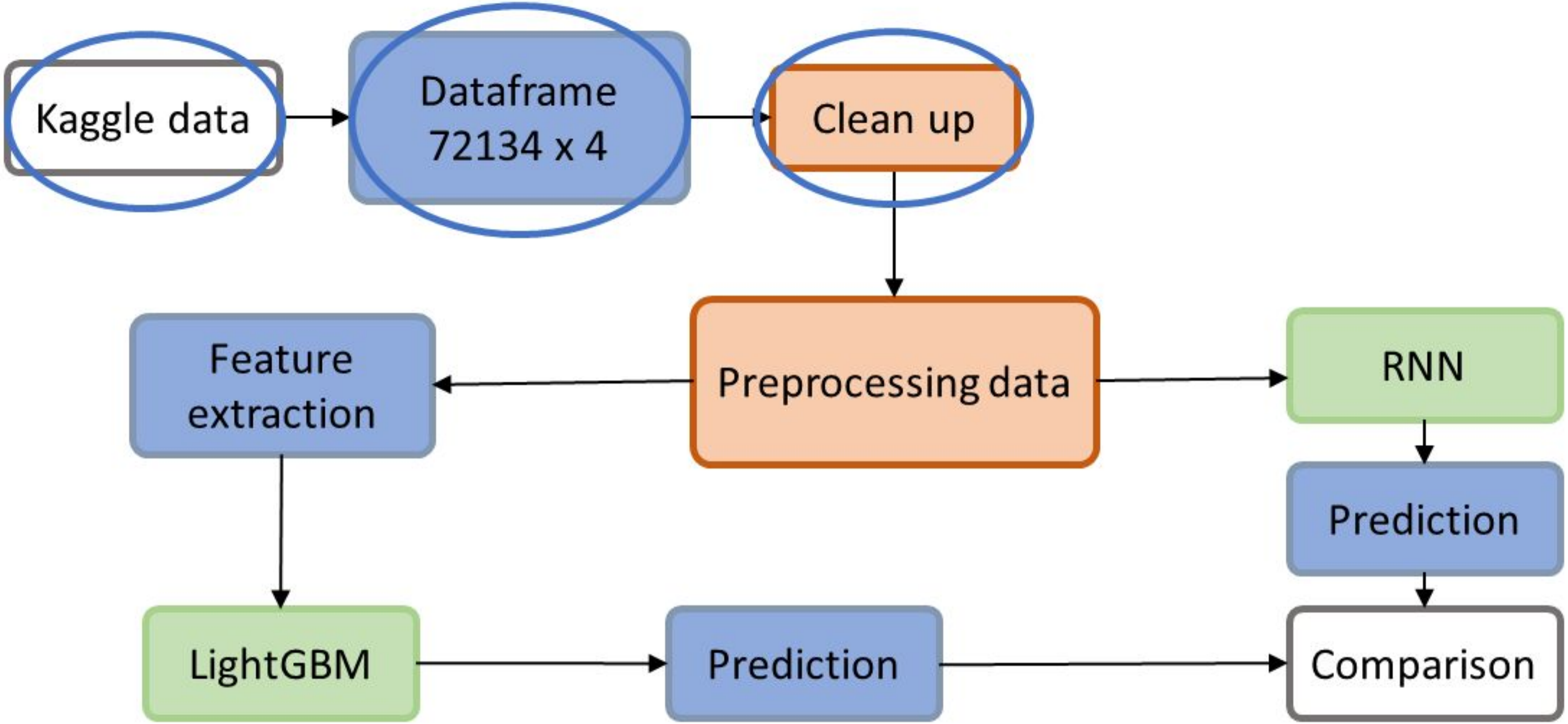
Marcus Gut, Rune Zeitzen, Magnus Oddershede and Brage Haldor Thomsen

Introduction and motivation

UNIVERSITY OF COPENHAGEN

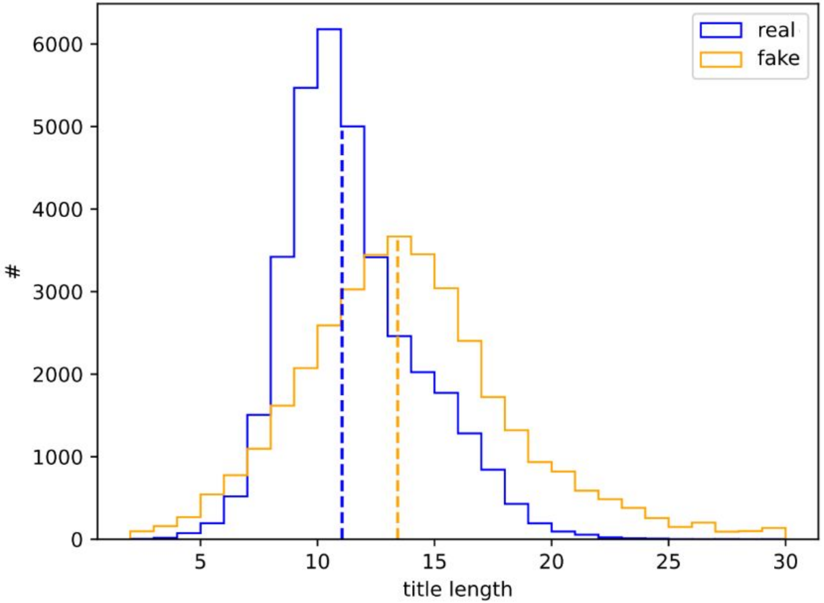
- Fake news has a documented effect on political beliefs (Ognyanova et al, 2020)
- Reuters and others spend resources on fact-checking
- Our task: Classification of fake news articles using Consumer grade hardware, utilising
  - LightGBM Gradient Boosted Decision Trees
  - Recurrent Neural Networks with Long Short-Term Memory (LSTM)

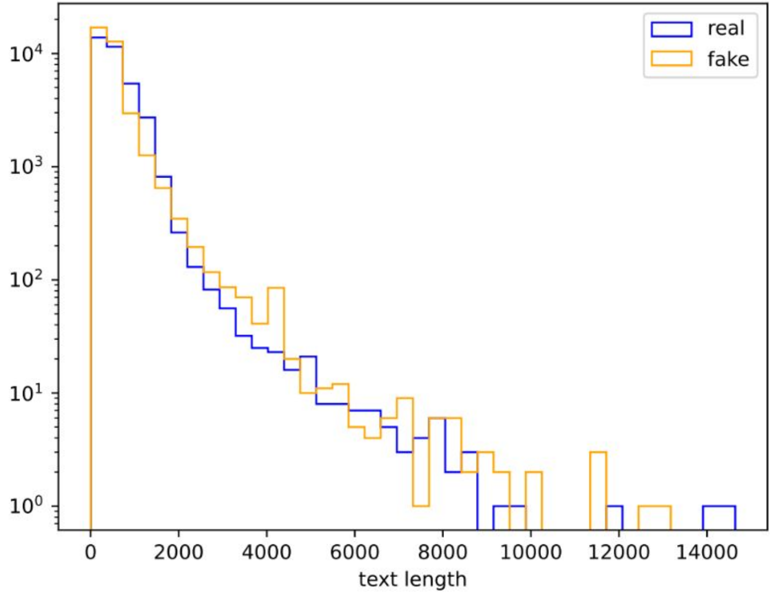
Outline


Kaggle data
Dataframe
72134 x 4
Clean up
Feature
extraction
Preprocessing data
RNN
Prediction
LightGBM
Prediction
Comparison

Dataset

- Very balanced - nearly 50/50 - thus making accuracy a performance measure
- Mix of long and short articles with no obvious difference
- Contains title, text and label





Dataset

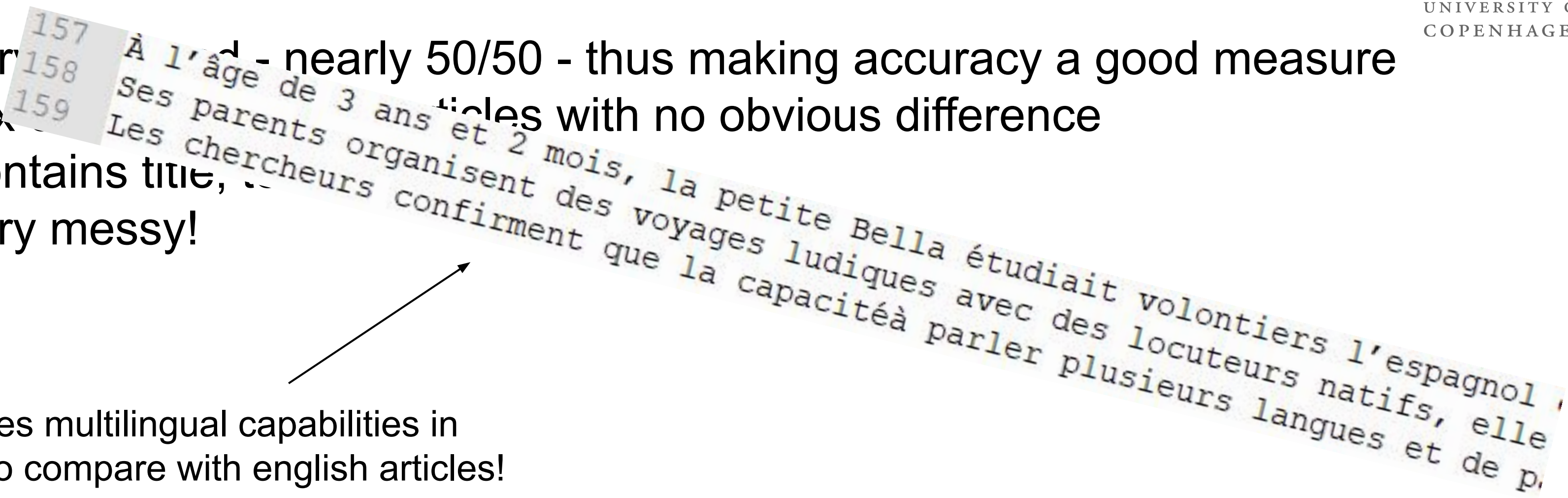
UNIVERSITY OF
COPENHAGEN

- Very balanced - nearly 50/50 - thus making accuracy a good measure
- Mix of long and short articles with no obvious difference
- Contains title, text and label
- Very messy - complicated dataset

Dataset

UNIVERSITY OF COPENHAGEN

- Ver… d - nearly 50/50 - thus making accuracy a good measure
- Mix… ticles with no obvious difference
- Contains title, …
- Very messy!

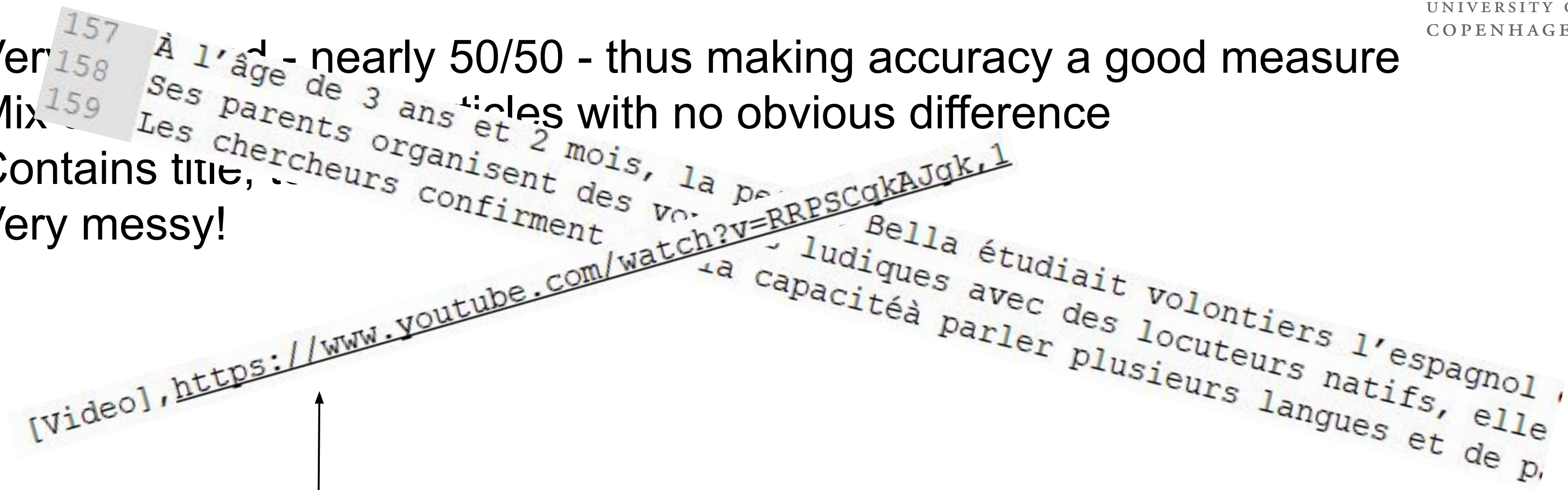

À l'âge de 3 ans et 2 mois, la petite Bella étudiait volontiers l'espagnol
Ses parents organisent des voyages ludiques avec des locuteurs natifs, elle
Les chercheurs confirment que la capacitéà parler plusieurs langues et de p

Requires multilingual capabilities in order to compare with english articles!

Dataset

UNIVERSITY OF COPENHAGEN

- Ver... - nearly 50/50 - thus making accuracy a good measure
- Mix... ...cles with no obvious difference
- Contains title, ...
- Very messy!

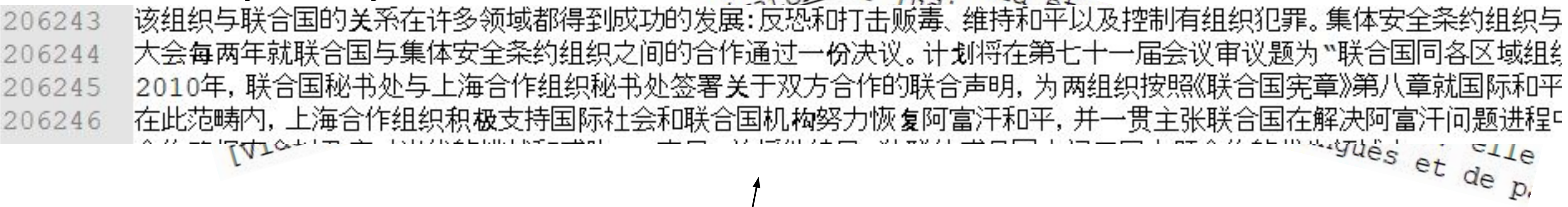


"Text" is literally just a link!

Dataset

UNIVERSITY OF COPENHAGEN

- Ver... nearly 50/50 - thus making accuracy a good measure
- Mix... icles with no obvious difference
- Contains title, ...
- Very messy!



Classifying this is probably beyond the scope of this project…

Cleanup

UNIVERSITY OF COPENHAGEN

- Average word length - takes care of link-only articles without removing actual articles using links

Cleanup

UNIVERSITY OF COPENHAGEN

- Average word length - takes care of link-only articles without removing actual articles using links
- Search for most common special char. in other languages: æøå, ç, ¿, various chinese and arabic symbols, ect.

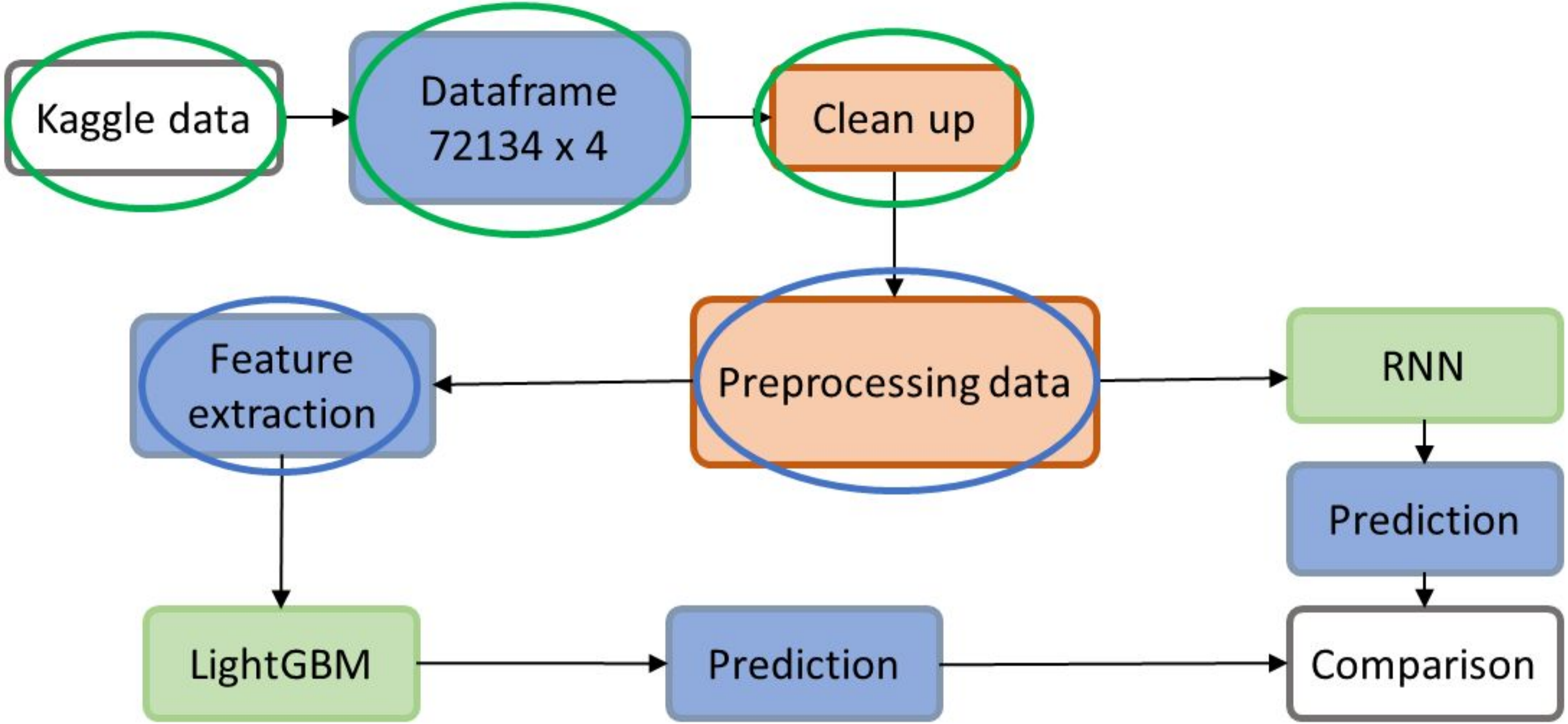
Cleanup

UNIVERSITY OF COPENHAGEN

- Average word length - takes care of link-only articles without removing actual articles using links
- Search for most common special char. in other languages: æøå, ç, ¿, various chinese and arabic symbols, ect.
- Remove formatting artifacts, double spaces, \n ect.

Outline

UNIVERSITY OF COPENHAGEN



Preprocessing

Geologist thinks she found a meteorite, but the geologist really found a rock!

[0, 1527, 45, 236, 5, 0, 32, 1, 28490, 219, 236, 5, 0]

Preprocessing

Geologist thinks she found a meteorite, but the geologist really found a rock!

[0, 1527, 45, 236, 5, 0, 32, 1, 28490, 219, 236, 5, 0]

0 is unknown

Preprocessing

Geologist thinks she found a meteorite, but the geologist really found a rock!

Space between special character

Uppercase to lowercase

Geologist thinks she found a meteorite , but the geologist really found a rock !

geologist thinks she found a meteorite, but the geologist really found a rock!

[0, 1635, 58, 270, 7, 0, 3, 38, 1, 19379, 250, 270, 7, 1665, 134]

[28490, 1527, 45, 236, 5, 0, 32, 1, 28490, 219, 236, 5, 0]

Preprocessing

Geologist thinks she found a meteorite, but the geologist really found a rock!

Space between special character

Geologist thinks she found a meteorite , but the geologist really found a rock !

[0, 1635, 58, 270, 7, 0, 3, 38, 1, 19379, 250, 270, 7, 1665, 134]

Uppercase to lowercase

geologist thinks she found a meteorite, but the geologist really found a rock!

[28490, 1527, 45, 236, 5, 0, 32, 1, 28490, 219, 236, 5, 0]

Preprocessing

Stop words removed

Geologist thinks she found a meteorite, but the geologist really found a rock!

↓

Geologist thinks found meteorite, geologist really found rock!

↓

[0, 1411, 136, 0, 29374, 119, 136, 0]

Special character to single word

One of the F*YoFlag organizers is called Sunshine.**

↓

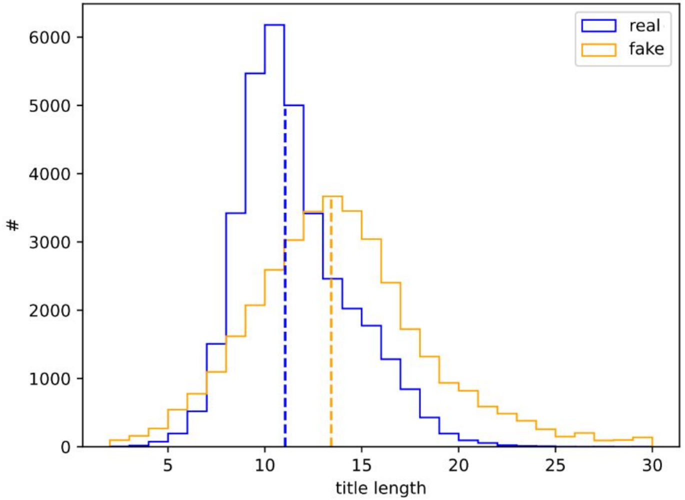
One of the wordwithasterisk organizers is called Sunshine.

↓

[0, 3, 1, 660, 5729, 8, 163, 0]

Feature extraction for LightGBM

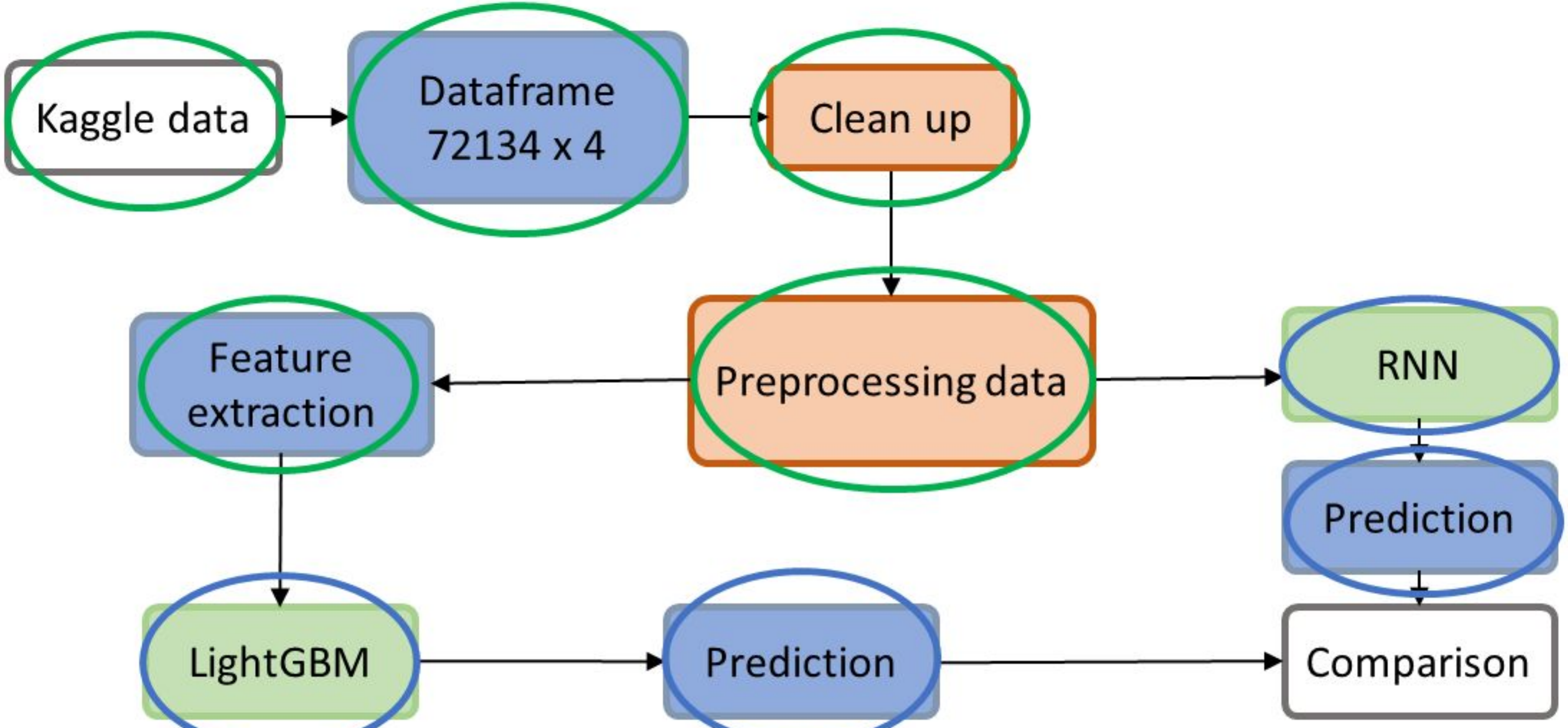
- Title and text length
- Average word length
- Entropy gain from splitting at a




```
final_words = ['bizarre', 'discovery', 'legislation', 'legislative', 'cataclysmic', 'event', 'inauguration',
               'unheard', 'earth-shattering', 'election', 'claim', 'bipartisan', 'unexplained', 'office',
               'inside', 'senate', 'supreme', 'blockbuster', 'unveiled', 'exposed']
```

Outline

UNIVERSITY OF COPENHAGEN



LightGBM GBDT

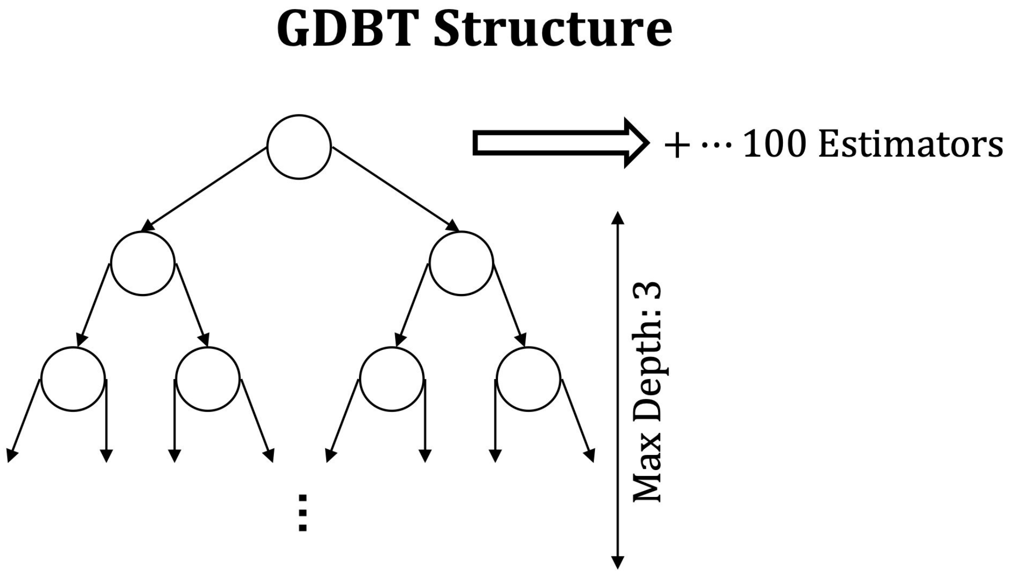
UNIVERSITY OF COPENHAGEN

Model Highlights

- Gradient Boosted Decision Tree
- Easy to set up - Fast training
- Performant on minimal preprocessi

HyperParams:

- max_depth: 3
- learning_rate: 0.01
- n_estimators: 100
- subsample: 0.8
- reg_alpha (L1): 0.1
- reg_lambda (L2): 0.1



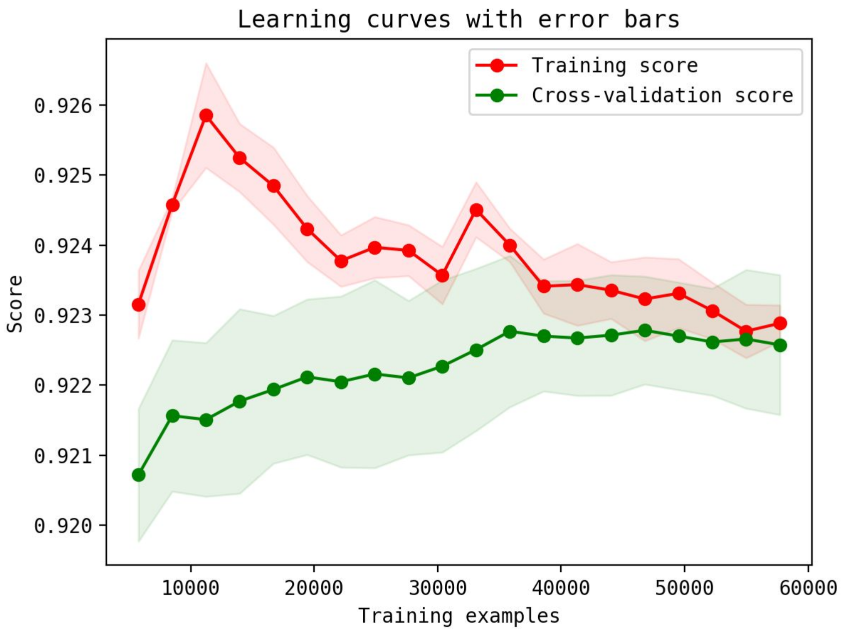
LightGBM

LightGBM GBDT
Combating overfitting

UNIVERSITY OF COPENHAGEN

Reduced model complexity

- Optimized using k-fold cross validation & learning curves
  - small k-fold cross validation spread indicating no overfitting
  - Convergent learning curves



LightGBM

Tensorflow LSTM

UNIVERSITY OF COPENHAGEN

Model Highlights

- Long Short-Term Memory (RNN)
- Naturally well suited for sequence data, specifically structured text
    - Handles and uses context through word ordering
    - Tackles unseen data well

TensorFlow

LSTM RNN Structure

Sequential Keras Implementation

Tokenised Text Sequence

↓

Embedding Layer

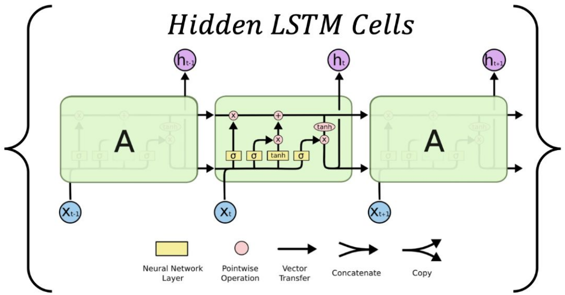
↓

Long Short-Term Memory Layer

↓

Dense Output Layer
Sigmoid Activation

0: Real
1: Fake



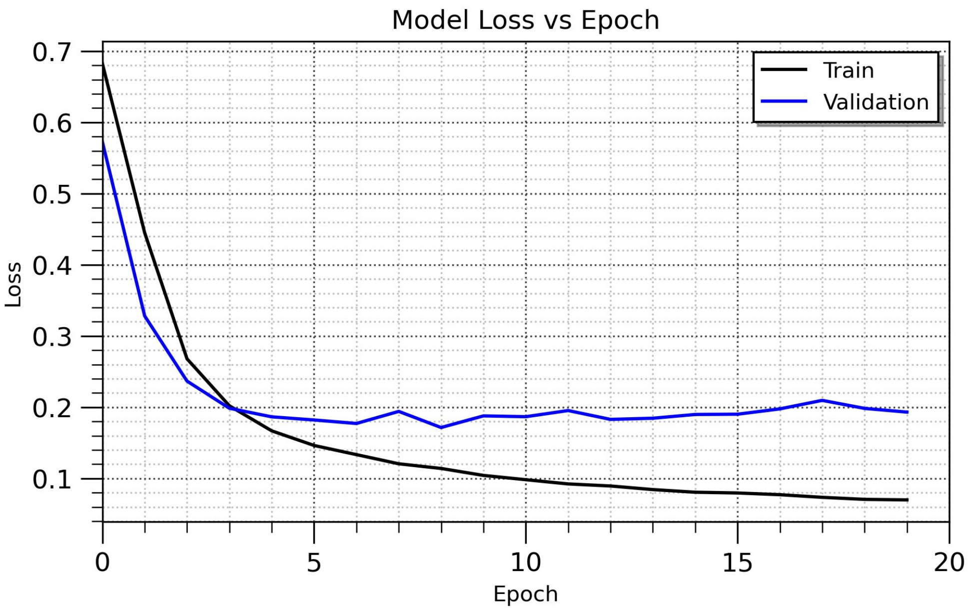
LSTM Hidden Cells Figure:
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

TensorFlow LSTM

Combating overtraining

UNIVERSITY OF COPENHAGEN

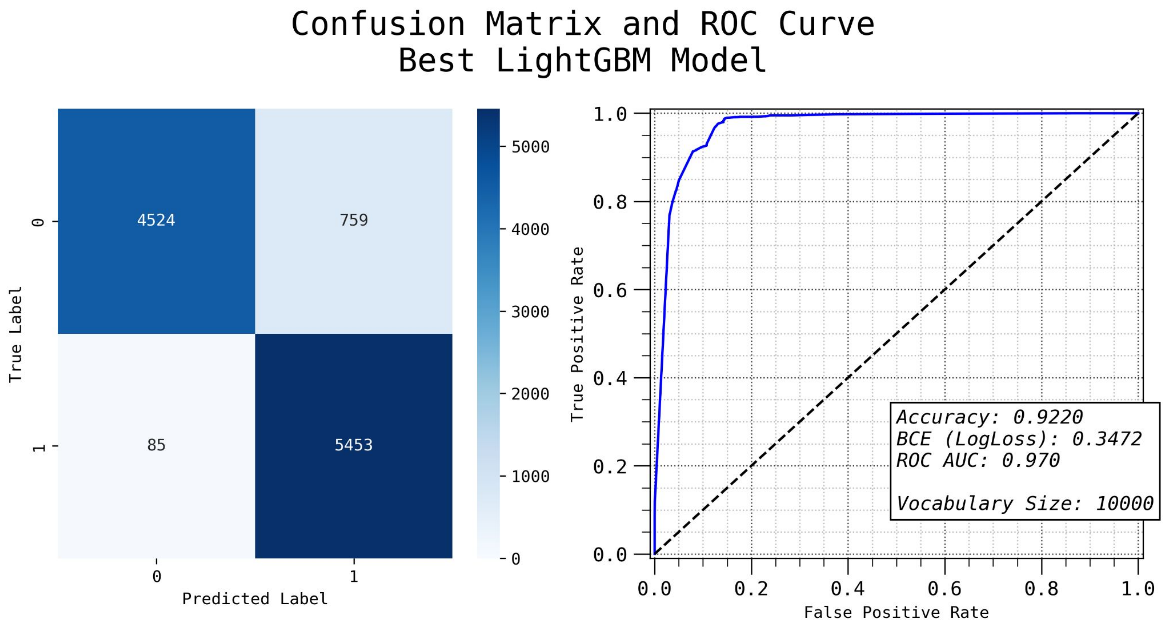
- Model structure and batch size optimised for efficient training
- Input & Recurrent dropout to combat overtraining
- Batchsize: 2048 (Most Stable)
- Training time ~ 20-30 minutes



TensorFlow

Results and Performance
LightGBM GBDT

- Acceptable metrics
- False positive is dominant error for default cut (p=0.5)
- A very clean cut is not possible (ROC curve)
- Extremely high true positive rate (true fake classified as fake) is attainable



Preprocessing: lowercase, spaces, stopwords, cleanup

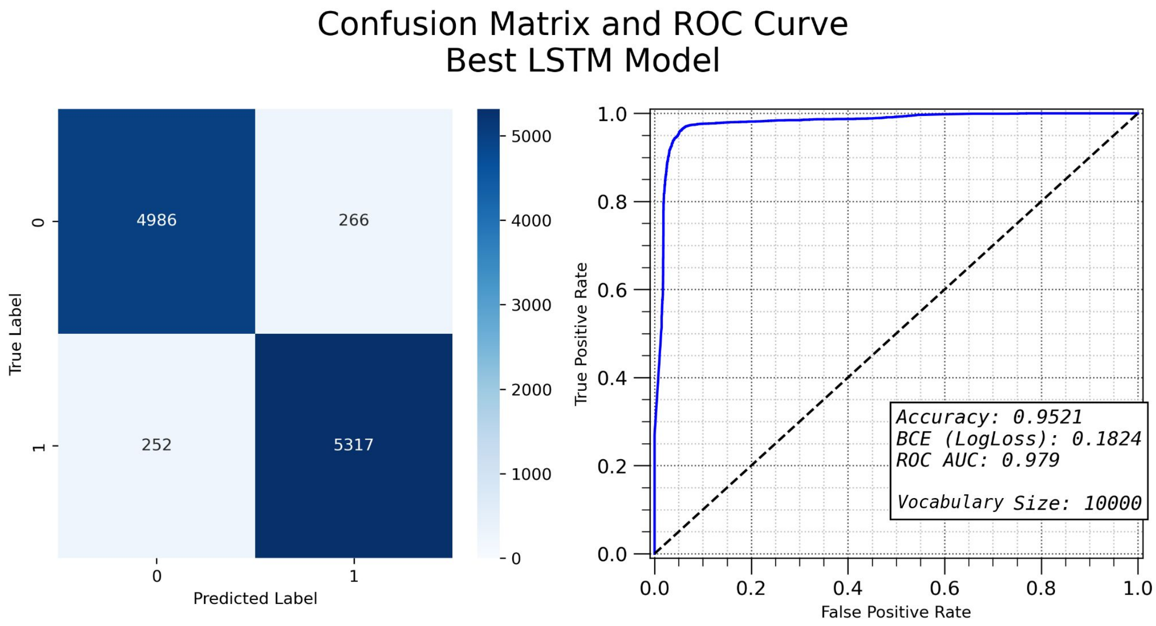
LightGBM

UNIVERSITY OF COPENHAGEN

Results and Performance
TensorFlow LSTM RNN

UNIVERSITY OF COPENHAGEN

- Better metrics than LGBM
- False positive and false negative are balanced
- Good separation, but one type of error cannot be excluded



Preprocessing: lowercase, spaces, cleanup

TensorFlow

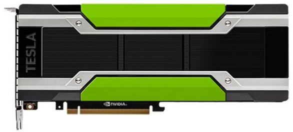
Trying BERT on GPUs

BERT: Bidirectional Encoder Representation of Transformers

Using a pretrained BERT Tokenizer: 'bert-base-uncased'.

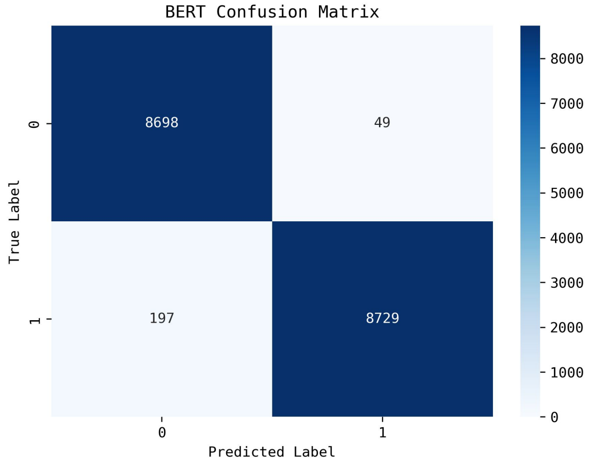
Fine tuned using our dataset, run on Kaggle NVIDIA P100 GPU.

Yielded high accuracy of about 98% on test data.



NVIDIA P100 GPU
Training time: 1.5 hours

Google AI



Preprocessing: lowercase, spaces, cleanup

UNIVERSITY OF COPENHAGEN

Vocabulary size and preprocessing

UNIVERSITY OF COPENHAGEN

Some observations:

- Preprocessing improved performance to a certain extent
  - Clean, lowercase, and spaces improve performance
  - Find and replace ” (f**k → wordwithasterisk) impairs performance
- Preprocessing becomes irrelevant when vocabulary becomes large

Conclusion

- ➔ Well performing NLP classification tasks can be executed on personal computers - no need for HPC resources.
  - ◆ RNN-LSTM: good metrics and general separation
  - ◆ LGBM-GBDT: Very pure “fake” classification is possible
- ➔ Preprocessing and feature extraction improves performance to a certain extent.
- ➔ Transformer based models, BERT, offer improved performance at the cost of computational time.

Predictions on current NEWS

Using the LSTM model, our trained model labelled this **BBC** article as ***Real***.

Donald Trump is 'toast' if indictment correct, William Barr says

8 hours ago

Indictments of Donald Trump

GETTY IMAGES

William Barr was once one of Mr Trump's staunchest allies but has been critical of him since leaving office

BBC https://www.bbc.com/news/world-us-canada-65875898

The same model flagged this article from **The Onion** as ***Fake.***

Trump Takes Out Full-Page Newspaper Ad Calling For Death Penalty For Himself

Published April 4, 2023 | Alerts

https://www.theonion.com/trump-takes-out-full-page-newspaper-ad-calling-for-deat-1850299979

the ONION® America's Finest News Source.

Further work

➔ Bidirectional, peephole & coupled forget/input gate LSTMs or even GRU (Gated Recurrent Unit) for 1 update unit

➔ Generate fake news articles

- Using trained LSTM
- Using transformer based model

➔ General Fake News Detection Software

- Using our model to flag fake news, utilising a webscraber

UNIVERSITY OF
COPENHAGEN

Appendix

Github Repository:

https://github.com/Chrowian/Final_Project_GutQuaadeHaldorZeitzen.git

Dataset:

https://www.kaggle.com/datasets/saurabhshahane/fake-news-classification

STATEMENT

All authors contributed equally to all parts of the project, both in developing key ideas for investigation, code to preprocess and run the Machine Learning algorithms and subsequent analysis of the results.

Cleaning Data

UNIVERSITY OF COPENHAGEN

Removal of empty strings, NaN's, non-english articles, only hyperlinks, were coded. In the end, approx. 70000 articles were left.

```
df = df.apply(lambda x: x.str.strip() if x.dtype == "object" else x)
df.replace(to_replace='', value=pd.NA, inplace=True)
df.dropna(inplace=True)
```

```
non_english_mask = df['text'].astype(str).str.contains(pattern, regex=True)
df_new = df[~non_english_mask.values]
```

```
final_drop_idx = avg_word_len.index.to_series()[avg_word_len < 3]
final_drop_idx_2 = avg_word_len.index.to_series()[avg_word_len > 10]
df_new.drop(final_drop_idx)
df_new.drop(final_drop_idx_2)
return df_new
```

Preprocessing

UNIVERSITY OF COPENHAGEN

Pandas dataframes are used in this project. With dataframes finding and replacing things are quick and easy.

```
def preprocess_space(dataframe):
    dataframe = dataframe.str.replace(special_chars, r' \1 ', regex=True)
    return dataframe
```

```
def lowercase(string):
    return str(string).lower()
```

```
def remove_stop_words(text):
    stop_words = set(stopwords.words('english'))
    filtered_tokens = [word for word in text.split() if word.lower() not in stop_words]
    return ' '.join(filtered_tokens)
```

Tokenization

UNIVERSITY OF COPENHAGEN

To turn words into tokens we first create a vocabulary of words from the articles.

This is done by counting how often each words appear and then giving the number of words you want in your vocabulary.

The tokenizer used were implemented using Keras, as this package was optimised for faster runtime.

```
# Initialize tokenizer
tokenizer = Tokenizer(num_words=vocab_size, filters='')

# Extract columns
titles = df[title_column]
texts = df[text_column]

# Fit the tokenizer on the texts
tokenizer.fit_on_texts(pd.concat([titles, texts]))

# Convert texts to sequences
sequences_titles = tokenizer.texts_to_sequences(titles)
sequences_texts = tokenizer.texts_to_sequences(texts)

# Pad sequences
padded_titles = pad_sequences(sequences_titles, maxlen=max_length, padding='post')
padded_texts = pad_sequences(sequences_texts, maxlen=max_length, padding='post')
```

This implementation found our own vocabulary

```
def get_vocab(X, n_words=40000, num_articles = 10000):
    """
    :return: vocabulary dictionary of words and their corresponding indices
    """
    senlist = X.values.tolist()[0:num_articles]
    all_words = list(chain(*[i.lower().split() for i in senlist]))
    words, count = np.unique(all_words, return_counts=True)
    idxs = np.argsort(count)[-n_words:]
    vocab = ['<UNK>'] + list(words[idxs][::-1])
    vocab_d = {vocab[i]: i for i in range(len(vocab))}
    return vocab_d
```

Tokenization

UNIVERSITY OF COPENHAGEN

This vocabulary assigns an integer from 0 to “number of words in vocabulary” to each word. So the most common word, usually “the”, is given integer 1 and so on.

If a word is not in the vocabulary it will be given the integer 0 and described as “UNK”, meaning its an unknown word.

From this vocabulary every word is replaced with is corresponding integer, thereby tokenizing the text.

Entropy splitting 1

UNIVERSITY OF COPENHAGEN

A list of 150 words related to American politics (a significant portion of articles seemed to be related to American politics, with “Trump” being present in approx. 50% of all articles) or “article-like” lingo (exclusive, breaking, interview, ect.) are counted in all articles, separated into four counts: occurring in fake or real articles, and not occurring in fake or real articles. From this, the entropy gain (based on Shannon entropy) is calculated from a potential split at this word. The chosen words are in the main presentation. See next slide for entropy calculations.

```
def search_substrings(row, index):
    for j, substring in enumerate(substrings):
        for column_name in X.columns:
            if substring in row[column_name]:
                bool_matrix[index, j] = 1

X.apply(lambda row: search_substrings(row, X.index.get_loc(row.name)), axis=1)
```

Entropy splitting 2

```python
def calculate_entropy(fake_occ, real_occ, fake_non_occ, real_non_occ):
    total_fake = fake_occ + fake_non_occ
    total_real = real_occ + real_non_occ
    total = total_fake + total_real
    p_fake_with_word = fake_occ / total
    p_real_with_word = real_occ / total

    entropy_parent = -(p_fake_with_word * np.log2(p_fake_with_word) + p_real_with_word * np.log2(p_real_with_word))
    entropy_parent[np.isnan(entropy_parent)] = 0  # Set NaN values to 0

    return entropy_parent


def calculate_entropy_gain(fake_occ, real_occ, fake_non_occ, real_non_occ):
    entropy_parent = calculate_entropy(fake_occ, real_occ, fake_non_occ, real_non_occ)

    total_fake = fake_occ + fake_non_occ
    total_real = real_occ + real_non_occ
    total = total_fake + total_real
    p_fake = total_fake / total
    p_real = total_real / total

    entropy_children = p_fake * calculate_entropy(fake_occ, fake_non_occ, fake_non_occ, real_non_occ) + \
                       p_real * calculate_entropy(real_occ, real_non_occ, fake_non_occ, real_non_occ)
    entropy_children[np.isnan(entropy_children)] = 0  # Set NaN values to 0

    entropy_gain = entropy_parent - entropy_children
    return entropy_gain
```

UNIVERSITY OF COPENHAGEN

LightGBM GBDT Setup

Setup using the LightGBM classifier class LGBMClassifier().

Key Hyperparamters:

n_estimators: 100, max_depth: 3, learning_rate: 0.01, subsample: 0.8, reg_alpha: 0.1, reg_lambda: 0.1

Initial experiments hinted at overfitting, and k-fold cross validation and learning curve divergence confirmed this. A Hyperparameter optimisation was carried out, reducing the complexity, and leading to satisfactory spread on k-fold cross validation experiments, and convergence in learning curves.

LightGBM GBDT Setup

UNIVERSITY OF COPENHAGEN

Performance of the LightGBM classifier was evaluated based of several indicators. The accuracy of the predicted labels on the test data, the binary cross entropy (LogLoss), and the ROC curve. While accuracy normally isn't always a good indicator for classification problems, we have an even data set, meaning the same number of fake and real articles, so that accuracy should be a good measure.

LightGBM

TensorFlow LSTM Setup

Implemented using the TenserFlow Keras API. The model built is a sequential model, where layers are added as needed. The simple model used for some of the results is implemented as shown by the figure in the presentation:

```
model = Sequential()
model.add(Embedding(max_features, 32))
model.add(LSTM(32, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid', kernel_regularizer=regularizers.l2(0.01)))
```

TensorFlow

TensorFlow LSTM Setup

UNIVERSITY OF COPENHAGEN

Although having only one LSTM layer may seem overly simple, in reality, as this is a type of RNN, the LSTM layer does many computations, and in that sense “adds” more “layers” to the model. It is not correct to call these time steps, x-1, x, x+1, layers, but they can be imagined as such. The model, even with this simple setup, was able to handle the complex data structures, and as such, it was not important to add more layers, which only would have made the model slower to train. The sequence length of the inputs to the model dictated the number of “timesteps” in the LSTM, typically we used between 300 and 400, with zeropadding on the end so that all inputs had the same shape (This is vital for LSTM RNNs).

TensorFlow

TensorFlow LSTM Setup

UNIVERSITY OF COPENHAGEN

The model structure and batchsize used was optimised to have a good tradeoff between stable model behaviour, i.e., semi-smooth converging loss functions, and training time. It was found that a batchsize of 2048 gave the most stable loss functions, while the reduction of units in the LSTM layer yielded faster training, while not costing much in performance. Increasing LSTM units, or adding another LSTM layer, only yielded accuracy scores that were slightly higher, ~ 0.96, at the cost of vastly increased computational time.

TensorFlow

BERT Classifier Setup

UNIVERSITY OF
COPENHAGEN

We used the pretrained model BERT to try and see whether or not a Transformer powered model would perform exceptionally well on the dataset. This was, as seen in the presentation, the case. Setting up the model was done using the TensorFlow Keras implementation version of the TFBertForSequenceClassification forward method. A high dropout rate was used to avoid overtraining and overfitting.

```
from transformers import BertTokenizer, TFBertForSequenceClassification, BertConfig

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)

config = BertConfig.from_pretrained('bert-base-uncased', num_labels=2)

config.hidden_dropout_prob = 0.35

model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased', config=config)
```

Google AI

BERT Classifier Setup

UNIVERSITY OF COPENHAGEN

As the BERT classifier model is a very large pretrained model, it may come as no surprise, that finetuning the pretrained model was time consuming. It was estimated, that the total training time for the model would have come close to 24 hours on our fastest available laptop (M2pro Macbook), which was by no means feasible. However, luckily, Kaggle offers 30 hours of GPU usage for its users per week, provided they have registered with a phone number. This allowed for much faster training times, in the order of 1 hour. The outputted, finetuned model, fills nearly 500MB, and is therefore a very heavy and complicated model.

Google AI

General Training Comments

UNIVERSITY OF COPENHAGEN

It is important to stress, that in the datasets were appropriately split, in such a way, to keep the testing dataset away from the model, while the training and validation datasets are used to train the model. Only when evaluating performance is the test dataset used.

Extended results

The next slides include more of the results, for varying levels of preprocessing, and vocabulary size. These results are found using the models described previously.

General Trends

- Pre processing improves performance to a certain extent
  - Data cleanup, lowercase, spaces improves performance
  - “Find and replace” (f**k → wordwithasterisk) impairs performance
  - It was found, above a certain threshold, that preprocessing returns diminished, once vocabulary size was sufficiently large.
- Very good separation is possible for RNN-LSTM
- Very high true positive rate is possible for LGBM tree based solution
- Models are robust when vocabulary size is decreased

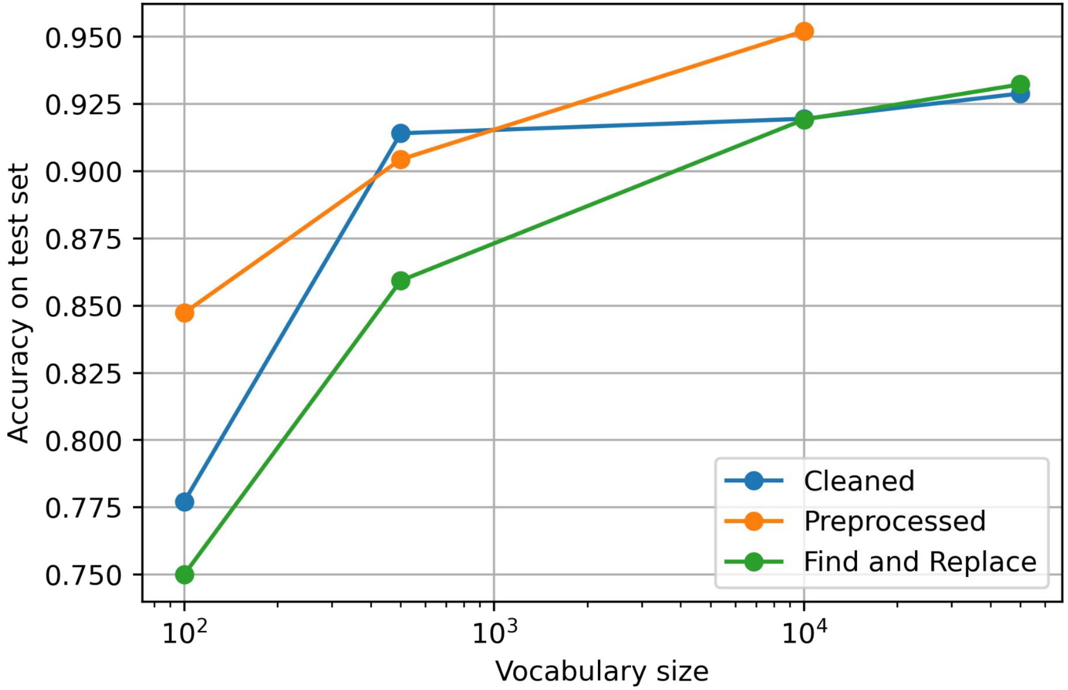
Extended results

The fact that our true negative rate cannot be very close to 1 (which is the case for the true positive rate) means that we cannot separate articles into two classes and be sure that one class contains only real articles. On the other hand, we can make sure that one class contains only fake news articles.

We believe that this is due to the fact that some fake articles are very well written and therefore are difficult to recognize. On the other hand, real news articles have a minimum level of writing, and the poorly written articles are therefore easy to classify as fake news.

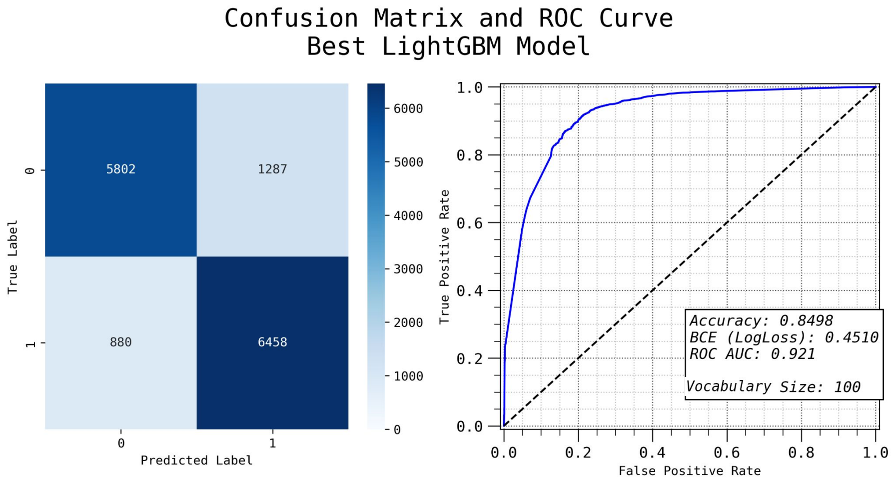
Accuracy vs vocabulary size

UNIVERSITY OF
COPENHAGEN



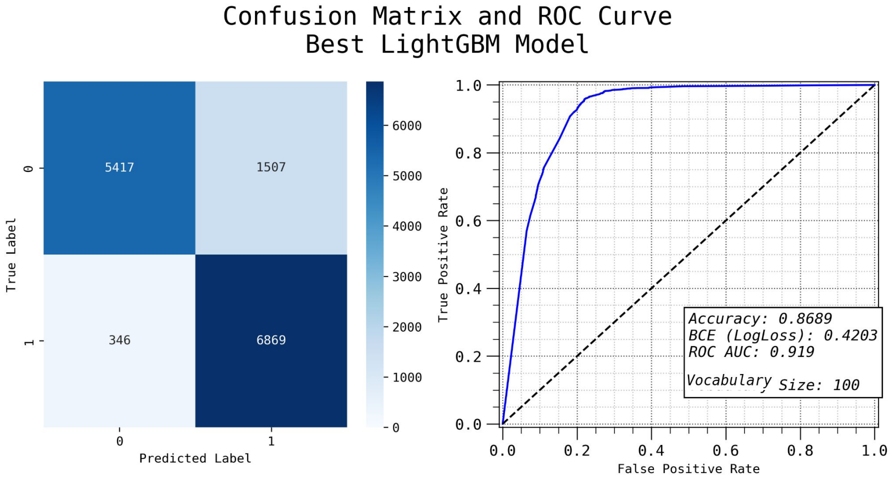
TensorFlow

LightGBM - Low Vocab Size



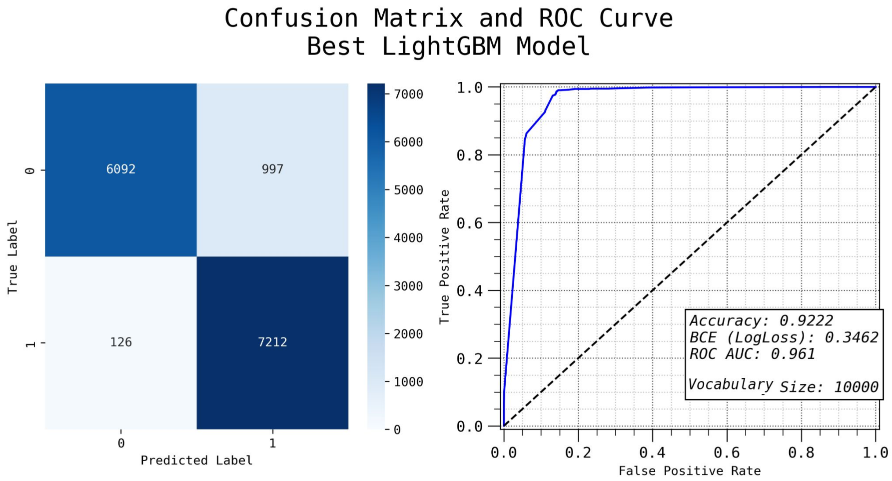
No preprocessing applied yielding a good accuracy nonetheless

Preprocessing applied yielding higher accuracy



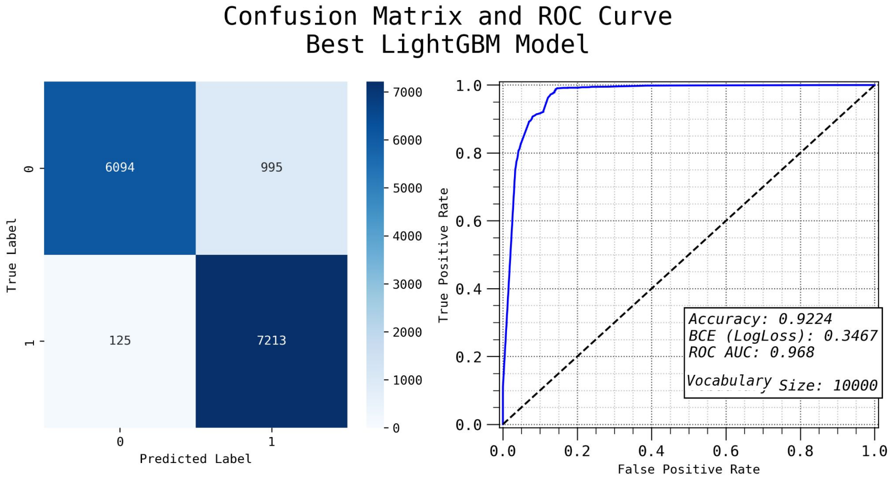
UNIVERSITY OF COPENHAGEN

LightGBM - Large Vocab Size

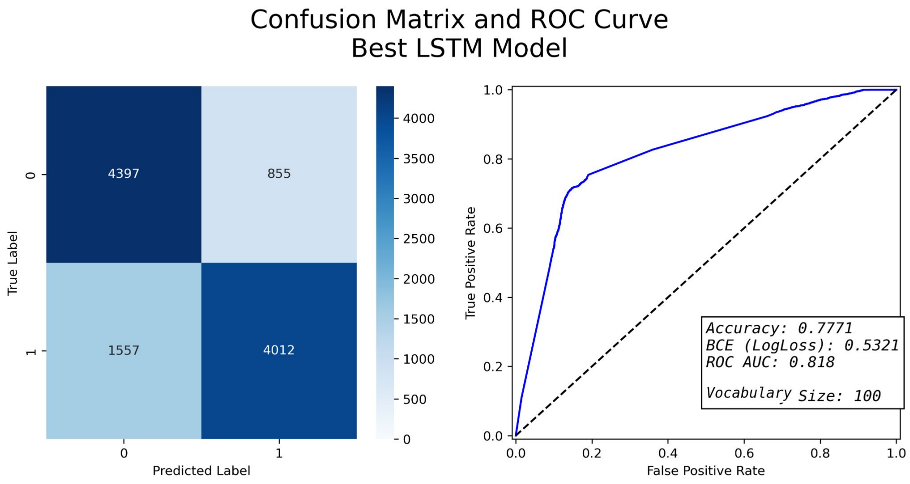


No preprocessing applied

Preprocessing applied

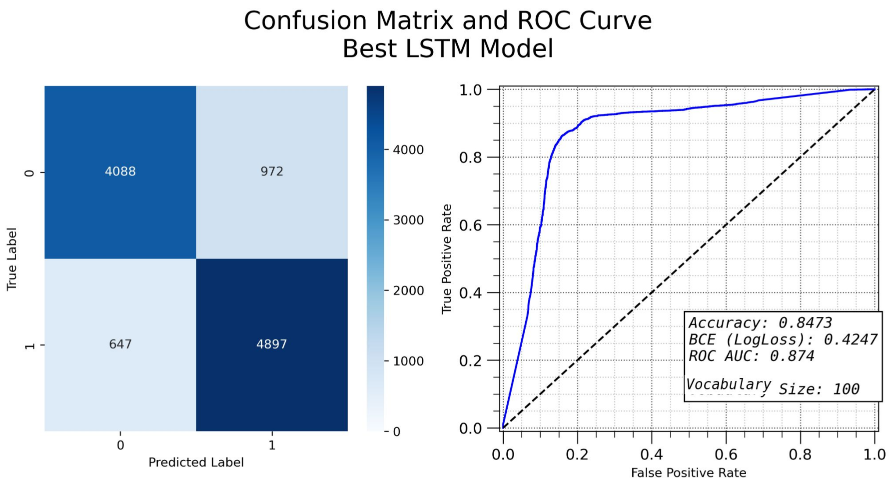


TensorFlow - Low Vocab Size



No preprocessing applied yielding a decent result

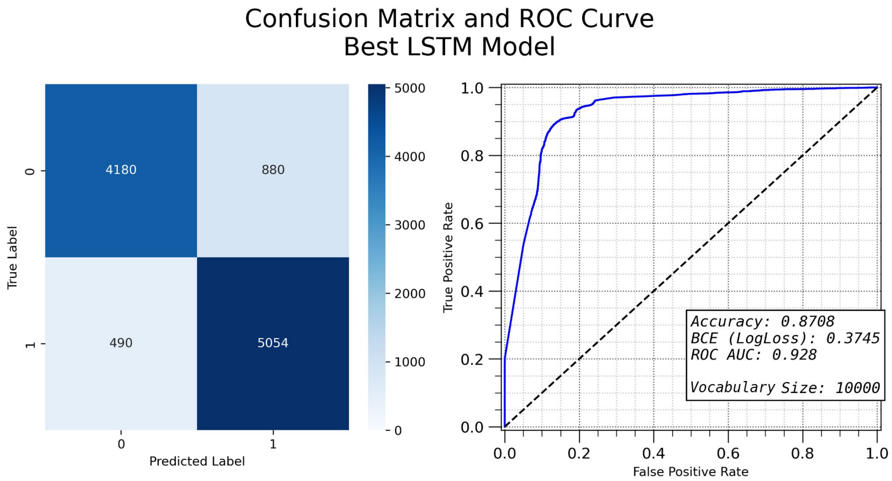
Preprocessing applied yielding higher accuracy



TensorFlow

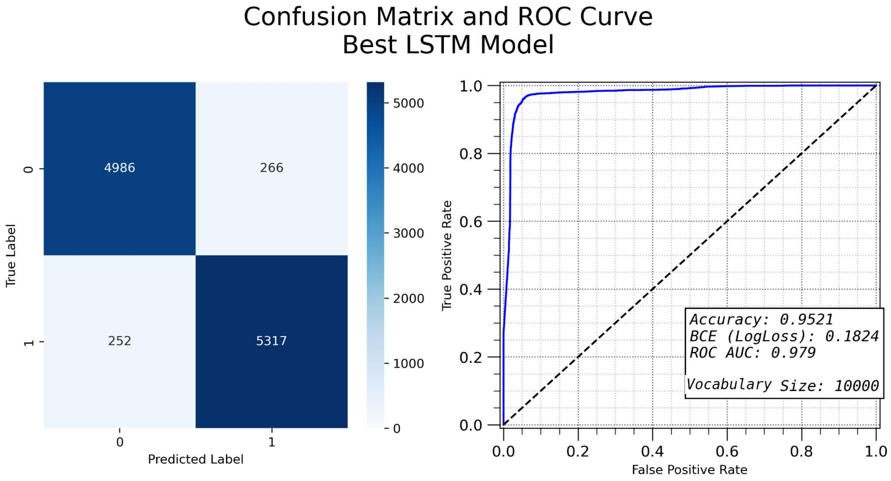
UNIVERSITY OF COPENHAGEN

TensorFlow - Large Vocab Size



No preprocessing applied yielding a good accuracy nonetheless

Preprocessing applied yielding higher accuracy



Real & Real Fake News

To analyse the two articles found on the internet, we had to first save the models in a dataframe, that was then passed through the same preprocessing and tokenizer functions that the model training data had gone through, so the consistency was there.

The model used to assess the validity of the two articles was a TensorFlow LSTM, which was structured and run in the same way as the previously described models. The model used had an accuracy of 93%, and was preprocessed. The articles went through the same preprocessing.

```
Classification of the Articles: 0: Real, 1: Fake

Article 1 (BBC):        [0.] ([0.04853711])
Article 2 (The Onion): [1.] ([0.93275005])
```

The two articles:
https://www.bbc.com/news/world-us-canada-65875898
https://www.theonion.com/trump-takes-out-full-page-newspaper-ad-calling-for-deat-1850299979

Real & Real Fake News

UNIVERSITY OF COPENHAGEN

We observed, that testing the model on a news article, which had little to do with politics, such as the attached news article, the model was more unsure how to classify them. This comes as no surprise, and it has more to say about out training data, than our model. A quick overview of the articles in our dataset also reveals, that the dataset mostly contains articles related to politics, and therefore, the model knows how to distinguish fake political articles from real.

This is the type of article, that the classifier had a harder time classifying.

https://www.theonion.com/god-still-little-pissed-off-every-time-human-takes-bite-1850524056

References

Katherine Ognyanova, David Lazer, Ronald E. Robertson, Christo Wilson: Misinformation in action: Fake news exposure is linked to lower trust in media, higher trust in government when your side is in power, The Harvard Kennedy School Misinformation Review, Vol 1.4 (2020)

Reuters Fact Check: Fact Check Housing Violation Notice in College Bathroom Prank Resurfaces (2021)

https://towardsdatascience.com/choosing-the-right-hyperparameters-for-a-simple-lstm-using-keras-f8e9ed76f046

https://towardsdatascience.com/lstm-by-example-using-tensorflow-feb0c1968537

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

https://towardsdatascience.com/long-short-term-memory-lstm-in-keras-2b5749e953ac

UNIVERSITY OF COPENHAGEN