CAPITOL

Classifying American Political Ideologies Through Online Language

Universita' degli Studi di Pavia
Bachelor degree in Artificial Intelligence
A.Y. 2024-45

by Brusati Lorenzo & Tsuhuy Ecaterina

TABLE OF CONTENTS

INTRODUCTION

1

Project Objective:

Classify American political ideologies by analysing tweets to distinguish between democratic and republican content.

2

Model Development:

Developed and evaluated models for binary classification using labeled political tweets focusing on text content to identify political affiliation.

3

Text Representation Approaches:

2 main text representation approaches were explored: TF-IDF vectorisation with traditional classifiers and pretrained word embeddings with LSTM networks.

DATASET

DATASET

ind...	date object	id int64	username object	text object
0 - ...	2021-04-19 ... 0% 2021-07-09 ... 0% 29358 others 100%	135168...		
0	2021-10-13 19:47:44	14483...	SenatorHassan	Happy th birthday to the @USN
1	2021-06-30 14:53:13	14102...	SenatorMenendez	The greatest generation's inves
2	2021-08-08 01:11:29	14241...	SenBillCassidy	Thanks to @SenTedCruz and @
3	2021-04-14 14:02:49	13823...	SenBlumenthal	/ To get lasting change we cant
4	2021-12-11 16:06:38	14697...	SenatorBraun	Today were celebrating years o
5	2021-05-11 17:18:50	13921...	SenJeffMerkley	The #ForthePeopleAct includes
6	2021-08-10 16:22:55	14251...	SenBlumenthal	Todays strong, bipartisan vote i
7	2021-08-12 01:13:26	14256...	SenatorHagerty	Supporting crime victims requir
8	2021-12-11 16:27:51	14697...	SenBillCassidy	We in Louisiana know how natu
9	2021-01-20 13:55:04	13518...	CoryBooker	Today we start anew.

Using 2 features:
tweets text and labels 0, 1 for Democrat and Republican

PoliticalTweets from Hugging Face:

DATA ANALYSIS

49.3% vs 50.7%

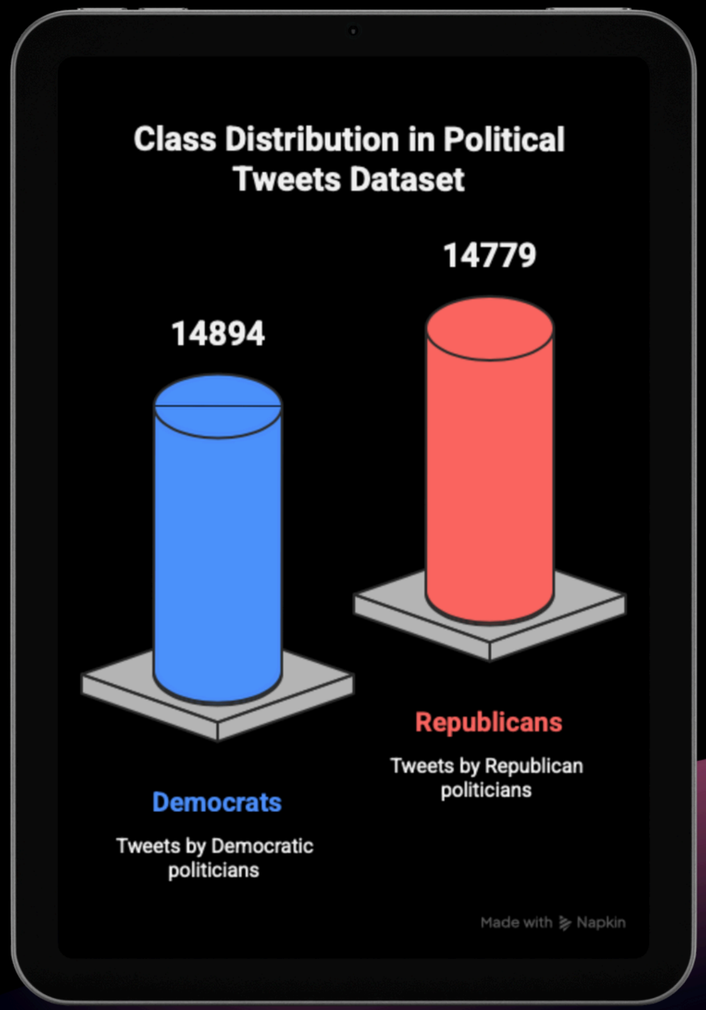
Class distribution

25-40 words

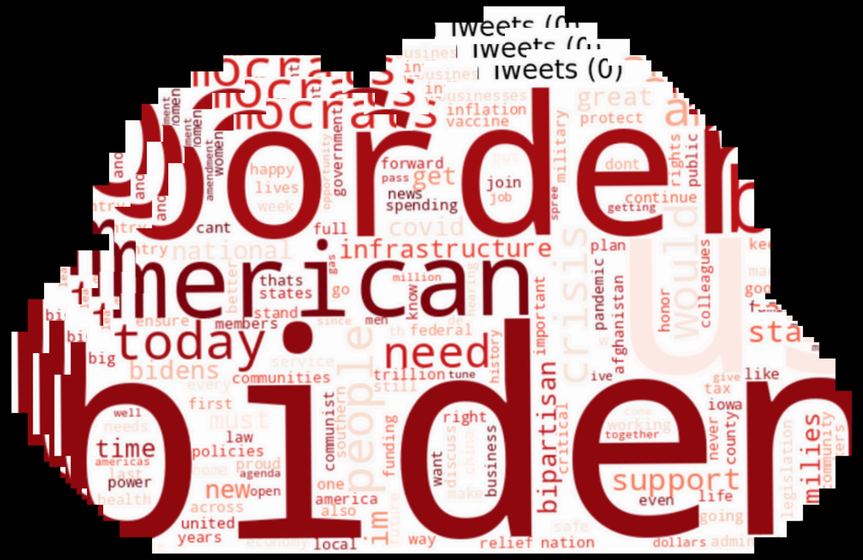
Word Count Skewness

214.82 chars

Average tweet length



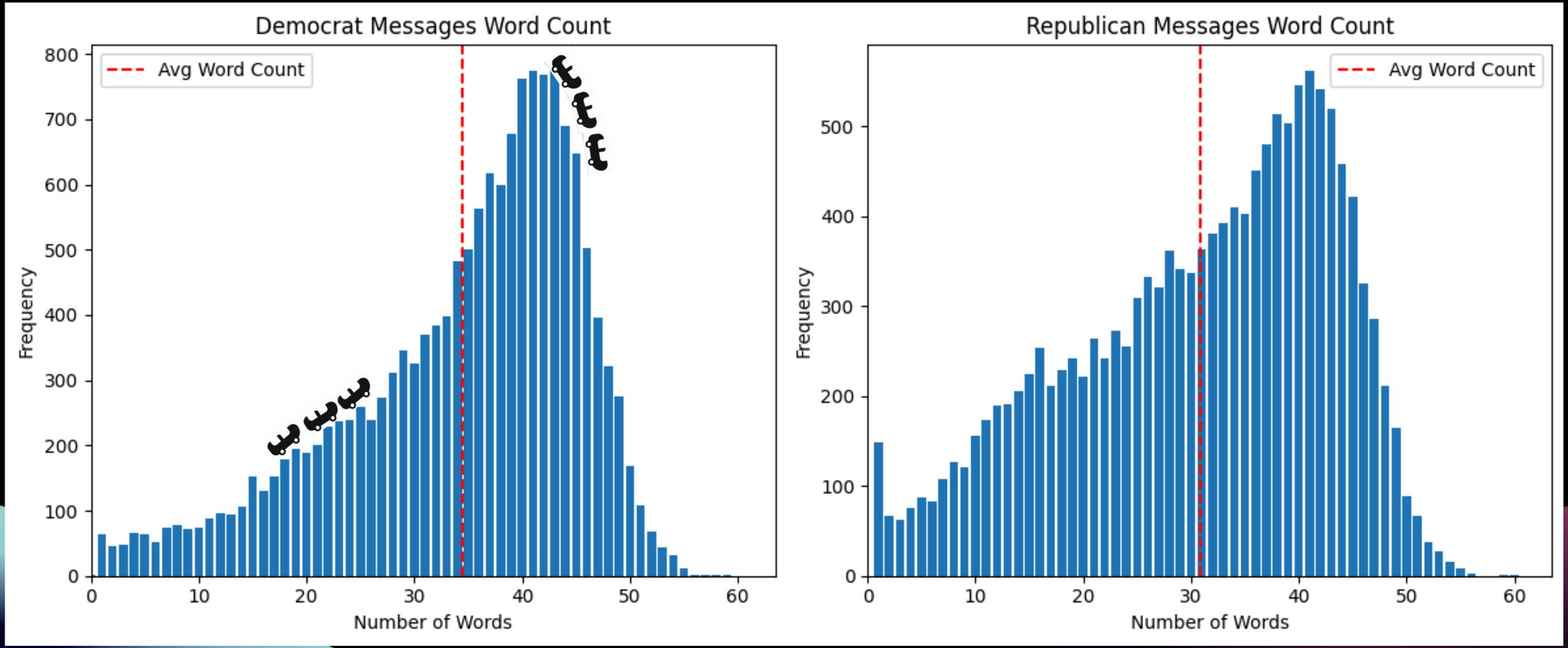
WORD CLOUD

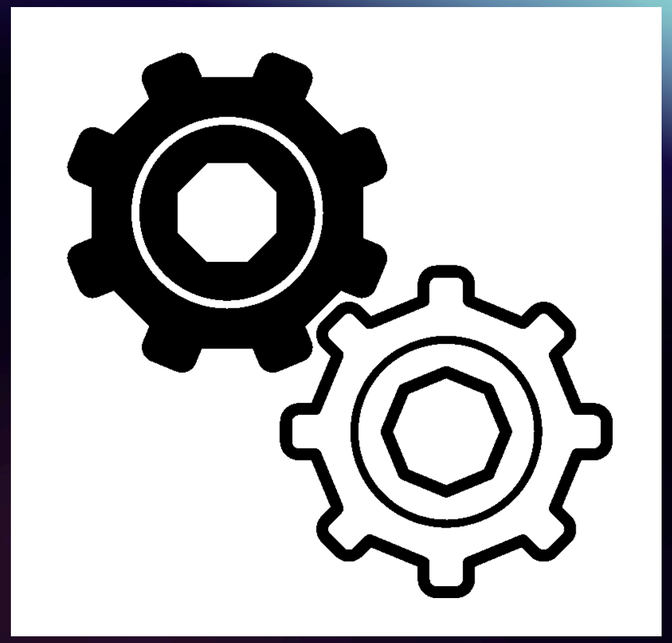


GOP.

DEMOCRATS

WORD COUNT


Democrat Messages Word Count
Avg Word Count
Frequency
Number of Words
0
100
200
300
400
500
600
700
800
0
10
20
30
40
50
60
Republican Messages Word Count
Avg Word Count
Frequency
Number of Words
0
100
200
300
400
500
0
10
20
30
40
50
60

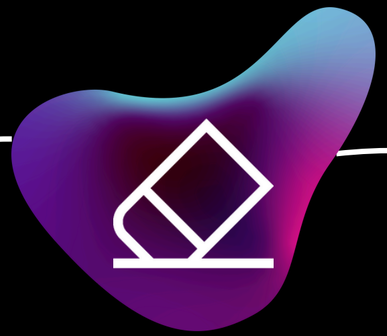


PREPROCESSING

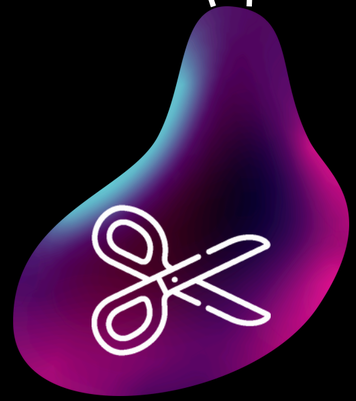
PREPROCESSING PIPELINE

Text normalisation

Hashtag preprocessing



Emoji removal

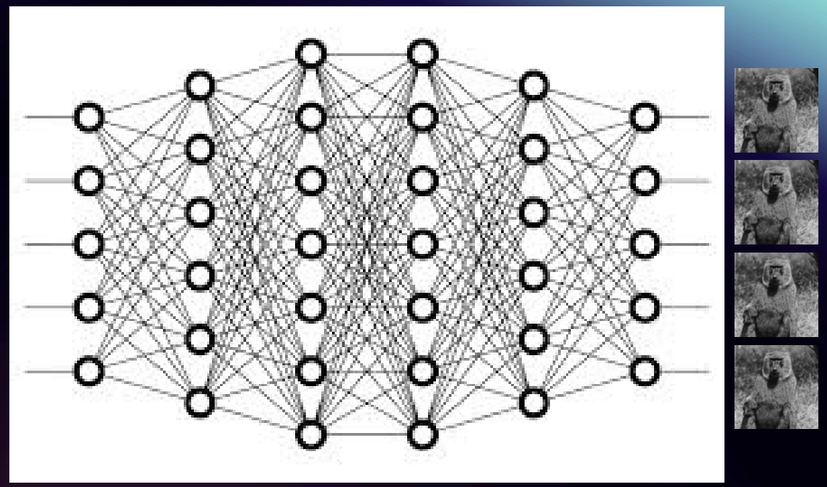


Regex cleaning

Stopwords filtering

PREPROCESSING PIPELINE

```python
def clean_text(text):
    text = str(text).lower()
    hashtags = re.findall(r"#\w+", text)
    for tag in hashtags:
        split_words = wordninja.split(tag.lstrip("#"))
        text = text.replace(tag, " ".join(split_words))
    text = emoji.replace_emoji(text, replace='')
    text = re.sub(r'\n', '', text)                                  # newline
    text = re.sub(r'@\w+', '', text)                                # menzioni (@user)
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)  # link
    text = re.sub(r'\w*\d\w*', '', text)                            # parole con numeri
    text = re.sub(r'[^\x00-\x7F]+', '', text)                       # caratteri n
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text) # punteggiat
    return text
```



MODELS AND ARCHITECTURES

FEATURE REPRESENTATION

TF-IDF vectorisation

Sparse vectors

No word order

No semantics

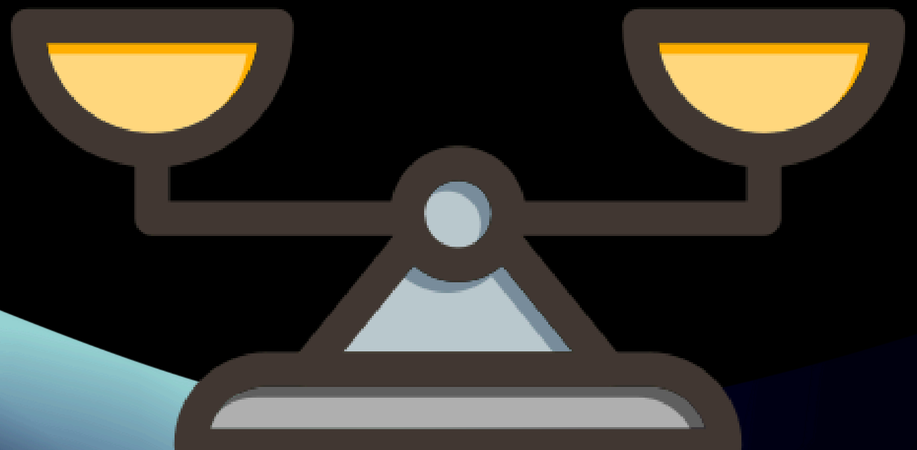
VS

Pretrained word embeddings

Dense vectors

Neural network architecture

Capture semantic relationships



MODEL ARCHITECTURES

TF-IDF	TF-IDF	GloVe	FastText
XGBoost	Logistic Regression	LSTM	LSTM

TF-IDF

$$w_{i,j} = tf_{i,j} \cdot \log\left(\frac{N}{df_i}\right)$$

- “The - is not a minus, it’s an hyphen”
- Count the number of occurrences in a document (x_train).
- Offset by the number of documents in the corpus that contain the word
- It's a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus

```python
X_train = list(df_train['text_cleaned'])
X_test = list(df_test['text_cleaned'])

# TF-IDF stands for:
# TF = Term Frequency: how often a word appears in a document.
# IDF = Inverse Document Frequency: how rare a word is across all documents.
# a high TF-IDF value = a term that's frequent in this document and rare elsewhere

vectorizer = CountVectorizer(vocabulary = vocab) # matrix of token counts
transformer = TfidfTransformer()

# TD-IDF for training data
X_train = vectorizer.fit_transform(X_train)
X_tfidf_train = transformer.fit_transform(X_train)

# TD-IDF for test data
X_test = vectorizer.transform(X_test)
X_tfidf_test = transformer.transform(X_test)

print(X_tfidf_train.shape, X_tfidf_test.shape)
```

TF-IDF

XGBoost

- Model that trains a fixed number of decision trees. Each tree is trained on a subset of the data.

```python
# XGBOOST MODEL

pipe = Pipeline([
    ('model', xgb.XGBClassifier(
        learning_rate=0.1,
        max_depth=11,
        n_estimators=80,
        use_label_encoder=False,
        eval_metric='auc'
    ))
])

pipe.fit(X_train, y_train)

y_pred_class_xgb = pipe.predict(X_test)
y_pred_train_xgb = pipe.predict(X_train)

print('Train: {}'.format(accuracy_score(y_train, y_pred_train_xgb)))
print('Test: {}'.format(accuracy_score(y_test, y_pred_class_xgb)))

acc_train_xgb = accuracy_score(y_train, y_pred_train_xgb)
acc_test_xgb = accuracy_score(y_test, y_pred_class_xgb)
report_xgb = classification_report(y_test, y_pred_class_xgb)

print("XGBoost classification report:")
print(report_xgb)

conf_xgb = confusion_matrix(y_test, y_pred_class_xgb)
print(conf_xgb)
```

TF-IDF

Logistic Regression

- Supervised learning algorithm for classification. Models the probability that an input belongs to a class using the logistic (sigmoid) function.

```
# LOGISTIC REGRESSION MODEL

pipe = Pipeline([
    ('model', LogisticRegression(
        solver='lbfgs',
        max_iter=1000,
        multi_class='auto'
    ))])

pipe.fit(X_train, y_train)

y_pred_class_log = pipe.predict(X_test)
y_pred_train_log = pipe.predict(X_train)

print('Train: {}'.format(accuracy_score(y_train, y_pred_train_log)))
print('Test: {}'.format(accuracy_score(y_test, y_pred_class_log)))

acc_train_log = accuracy_score(y_train, y_pred_train_log)
acc_test_log = accuracy_score(y_test, y_pred_class_log)
report_log = classification_report(y_test, y_pred_class_log)

print("Logistic Regression classification report:")
print(report_log)

conf_log = confusion_matrix(y_test, y_pred_class_log)
print(conf_log)
```

GloVe

- Unsupervised learning algorithm for word embeddings. It builds vectors by aggregating global word-word co-occurrence statistics from a corpus.

glove.6B.100d.txt (347.12 MB)

This preview is truncated due to the large file size. Create a Notebook or download this file to see the full content. **Download** **Create Notebook**

```
the -0.038194 -0.24487 0.72812 -0.39961 0.083172 0.043953 -0.39141 0.3344 -0.57545 0.087459 0.28787 -0.06731 0.30906 -0.26384 -0.13231 -0.20757 0.33395 -0.33848 -0.31743 -0.48336 0.1464 -0.37304 0.34577 0.052041 0.44946 -0
, -0.10767 0.11053 0.59812 -0.54361 0.67396 0.10663 0.038867 0.35481 0.06351 -0.094189 0.15786 -0.81665 0.14172 0.21939 0.58505 -0.52158 0.22783 -0.16642 -0.68228 0.3587 0.42568 0.19021 0.91963 0.57555 0.46185 0.42363 -0.0
. -0.33979 0.20941 0.46348 -0.64792 -0.38377 0.038034 0.17127 0.15978 0.46619 -0.019169 0.41479 -0.34349 0.26872 0.04464 0.42131 -0.41032 0.15459 0.022239 -0.64653 0.25256 0.043136 -0.19445 0.46516 0.45651 0.68588 0.091295
of -0.1529 -0.24279 0.89837 0.16996 0.53516 0.48784 -0.58826 -0.17982 -1.3581 0.42541 0.15377 0.24215 0.13474 0.41193 0.67043 -0.56418 0.42985 -0.012183 -0.11677 0.31781 0.054177 -0.054273 0.35516 -0.30241 0.31434 -0.33846
to -0.1897 0.050024 0.19084 -0.049184 -0.089737 0.21006 -0.54952 0.098377 -0.20135 0.34241 -0.092677 0.161 -0.13268 -0.2816 0.18737 -0.42959 0.96039 0.13972 -1.0781 0.40518 0.50539 -0.55064 0.4844 0.38044 -0.0029055 -0.349
and -0.071953 0.23127 0.023731 -0.50638 0.33923 0.1959 -0.32043 0.18364 -0.18057 0.28963 0.20448 -0.5496 0.27399 0.58327 0.20468 -0.40228 0.19974 -0.070237 -0.88049 0.20485 0.14071 -0.1009 0.09449 0.36973 0.44554 0.28998 -
in 0.085703 -0.22201 0.16569 0.13373 0.38239 0.35401 0.01287 0.22461 -0.43817 0.50164 -0.35874 -0.34983 0.055156 0.69648 -0.17958 0.067926 0.39101 0.16039 -0.26635 -0.21138 0.53698 0.49379 0.9366 0.66902 0.21793 -0.46642 0
a -0.27086 0.044006 -0.02026 -0.17395 0.6444 0.71213 0.3551 0.47138 -0.29637 0.54427 -0.72294 -0.0047612 0.040611 0.043236 0.29729 0.10725 0.40156 -0.53662 0.033382 0.067396 0.64556 -0.085523 0.14103 0.094539 0.74947 -0.19
" -0.30457 -0.23645 0.17576 -0.72854 -0.28343 -0.2564 0.26587 0.025309 -0.074775 -0.3766 -0.057774 0.12159 0.34384 0.41928 -0.23236 -0.31547 0.60939 0.25117 -0.68667 0.70873 1.2162 -0.1824 -0.48442 -0.33445 0.30343 1.086 0
's 0.58854 -0.2025 0.73479 -0.68338 -0.19675 -0.1802 -0.39177 0.34172 -0.60561 0.63816 -0.26695 0.36486 -0.40379 -0.1134 -0.58718 0.2838 0.8025 -0.35303 0.30083 0.078935 0.44416 -0.45906 0.79294 0.50365 0.32805 0.28027 -0.
for -0.14401 0.32554 0.14257 -0.099227 0.72536 0.19321 -0.24188 0.20223 -0.89599 0.15215 0.035963 -0.59513 -0.051635 -0.014428 0.35475 -0.31859 0.76984 -0.087369 -0.24762 0.65059 -0.15138 -0.42703 0.18813 0.091562 0.15192
- -1.2557 0.61036 0.56793 -0.96596 -0.45249 -0.071696 0.57122 -0.31292 -0.43814 0.90622 0.06961 -0.053104 0.25029 0.27841 0.77724 0.26329 0.56874 -1.1171 -0.078268 -0.51317 0.8071 0.99214 0.22753 1.0847 0.88292 0.17221 -0.
that -0.093337 0.19043 0.68457 -0.41548 -0.22777 -0.11803 -0.095434 0.19613 0.17785 -0.020244 -0.055409 0.33867 0.79396 -0.047126 0.44281 -0.061266 0.20796 0.034094 -0.64751 0.35874 0.13936 -0.6831 0.25596 -0.12911 0.2608
on -0.21863 -0.42664 0.5196 0.0043103 0.58045 -0.10873 -0.37726 0.4566 -0.60627 -0.075773 0.11306 0.17703 0.1605 0.074514 0.63649 -0.078852 0.75268 -0.24962 -0.51628 -0.33348 0.66754 -0.34183 0.61316 0.31668 0.64846 -0.07
is -0.54264 0.41476 1.0322 -0.40244 0.46691 0.21816 -0.074864 0.47332 0.088996 -0.22079 -0.12808 -0.1144 0.50891 0.11568 0.028211 -0.3628 0.43823 0.047511 0.20282 0.49857 -0.10068 0.13269 0.16972 0.11653 0.31355 0.25713 0.
was 0.13717 -0.54287 0.19419 -0.29953 0.17545 0.084672 0.67752 0.098295 -0.035611 0.21334 0.51663 0.20687 0.44082 -0.33655 0.56025 -0.6879 0.51957 -0.21258 -0.52708 -0.12249 0.33099 0.026448 0.59067 0.0065469 0.45405 -0.3
said -0.13128 -0.452 0.043399 -0.99798 -0.21053 -0.95868 -0.24609 0.48413 0.18178 0.475 -0.22305 0.30064 0.43496 -0.3605 0.20245 -0.52594 -0.34708 0.0075873 -1.0497 0.18673 0.57369 0.43814 0.098659 0.3877 -0.2258 0.41911 0
with -0.43608 0.39104 0.51657 -0.13861 0.2929 0.50723 -0.012544 0.22948 -0.6316 0.21199 -0.018043 -0.39364 0.74164 0.30221 0.51792 -0.25191 0.25373 -0.65184 -0.42963 0.0093622 0.023334 -0.39245 0.34948 0.21217 0.7346 -0.21
he 0.1225 -0.058833 0.23658 -0.28877 -0.028181 0.31524 0.070229 0.16447 -0.027623 0.25214 0.21174 -0.059674 0.36133 0.13607 0.18755 -0.1487 0.31315 0.13368 -0.59703 -0.030161 0.080656 0.26162 -0.055924 -0.35351 0.34722 -0.
as -0.32721 0.096446 0.34244 -0.44327 0.30535 -0.042016 -0.071235 -0.31036 -0.22557 -0.181 -0.29088 -0.61542 0.29751 0.030491 0.41504 -0.51489 0.68628 -0.020302 -0.18486 0.31605 0.59472 -0.2147 0.29256 0.43262 0.35466 -0.2
it -0.30664 0.16821 0.98511 -0.33606 -0.2416 0.16186 -0.053496 0.4301 0.57342 -0.071569 0.36101 0.26729 0.27789 -0.072268 0.13838 -0.26714 0.12999 0.22949 -0.18311 0.50163 0.44921 -0.020821 0.42642 -0.068762 0.40337 0.0951
by -0.20875 -0.1174 0.26478 -0.28339 0.19584 0.7446 -0.03887 0.028499 -0.44252 -0.30426 0.27133 -0.51907 0.52183 -0.76648 0.28043 -0.48344 -0.15626 -0.49705 -0.51024 -0.03652 0.20579 -0.6136 0.46388 0.73497 0.66813 -0.4443
at 0.1766 0.093851 0.24351 0.44313 -0.39037 0.12524 -0.19918 0.59855 -0.82035 0.28006 0.54231 0.023079 0.12837 -0.044489 0.3837 -0.75659 0.40254 -0.4462 -0.81599 -0.0091513 0.65219 -0.043656 0.54919 -0.16696 0.73028 -0.207
( 0.19247 0.36617 0.52301 -0.79857 -0.2592 0.18267 0.19564 0.83148 -0.67636 -0.84648 1.4429 -0.84978 -0.023986 1.328 0.74061 0.039546 0.61659 -0.075604 -0.59537 0.69163 0.71303 0.016798 0.57518 0.94396 0.38447 -0.095771 0.
) -0.13797 0.27084 0.84036 -0.45668 -0.49429 0.35777 0.077772 0.42481 0.0076481 -0.50942 1.4008 -0.79993 0.053011 1.2054 0.3783 0.0842 0.91317 -0.35173 -0.30452 0.65606 0.50428 0.074796 0.31149 0.81957 0.40216 0.10982 0.3
from 0.30731 0.24737 0.68231 -0.52367 0.44853 0.42044 0.0002514 0.15265 -0.61363 0.22631 0.083071 0.070425 0.017683 0.56807 1.0067 -0.46206 0.44524 -0.50984 -0.42985 0.19935 0.22729 0.51662 0.56282 0.41282 0.17742 -0.1569
his 0.12883 -0.82209 0.27438 -0.069014 0.17989 0.72605 -0.15112 0.0085541 -0.95122 0.77243 -0.28375 0.28329 0.14825 -0.01223 -0.019267 -0.03446 0.31506 -0.16639 -0.013435 -0.0020459 0.064905 -0.20989 0.12524 0.3523 0.6404
`` 0.16478 0.17071 0.62111 -1.2101 -0.84063 0.21893 0.48123 -0.15044 0.36701 -0.20857 -0.23385 0.019356 -0.045098 0.18001 0.11995 -0.25622 -0.026299 0.28473 -0.91322 0.59811 0.30248 0.27973 0.11444 -0.073628 0.88137 1.0633
'' 0.092672 0.20241 0.69394 -0.58775 -0.097297 0.045522 -0.14156 0.30736 -0.35448 -0.20612 -0.21092 -0.0026685 -0.11537 0.052913 0.02988 0.0067036 0.47268 0.44669 -0.35419 0.70959 0.6984 0.42713 -0.40276 -0.37443 0.7434 0.
an -0.4214 -0.18797 0.46241 -0.17605 0.36212 0.36701 0.27924 0.14634 -0.054227 0.45834 0.065416 -0.33725 0.067595 -0.36316 0.50302 -0.010361 0.72826 -0.17564 -0.33996 0.072864 0.64481 -0.23908 0.38383 0.13858 1.0994 -0.246
be -0.46953 0.38432 0.54833 -0.63401 0.010133 0.11364 0.10612 0.58529 0.032302 -0.12274 0.030265 0.52662 1.0398 -0.082143 0.19118 -0.83784 0.50763 0.44488 -0.72604 0.036893 0.24211 -0.28878 0.33657 0.13656 0.14579 -0.13221
has 0.093736 0.56152 0.48364 -0.45987 0.56967 -0.1694 0.018687 0.45529 0.065615 0.25181 -0.14251 0.10532 0.77865 0.1428 -0.08114 -0.069555 0.32433 0.019611 -0.15608 0.22235 0.35559 0.14713 0.19156 0.2803 0.27691 -0.2067 -0
are -0.51533 0.83186 0.22457 -0.73865 0.18718 0.26021 -0.42564 0.67121 -0.31084 -0.61275 0.089526 -0.24011 1.1878 0.67609 -0.022885 -0.92533 0.071174 0.38837 -0.42924 0.37144 0.32671 0.43141 0.87495 0.34009 -0.23189 -0.411
have 0.15711 0.65606 0.0021149 -0.65144 -0.28427 -0.20369 -0.077596 0.40798 -0.03447 -0.1639 -0.21597 0.34178 1.196 0.33639 -0.21076 -0.56015 0.1507 0.34912 -0.97128 0.18152 0.74488 0.20029 0.66986 0.16085 -0.0013507 -0.5
but -0.057078 0.39874 0.68861 -0.68151 -0.45503 0.2008 0.17974 0.053648 0.43762 -0.026725 0.13383 -0.0078137 0.42207 -0.31801 0.18065 -0.35307 -0.30929 0.04066 -0.48854 0.3791 0.47955 -0.041942 0.40894 0.12419 0.40096 0.15
```

FastText

- Word embedding model by Facebook. Extends Word2Vec by representing words as bags of character n-grams, capturing subword information.

$$h_w = \sum_{g \in w} x_g$$

mang erai ange
man ang era gera
nge ger rai nger

Character n-grams

+

mangerai

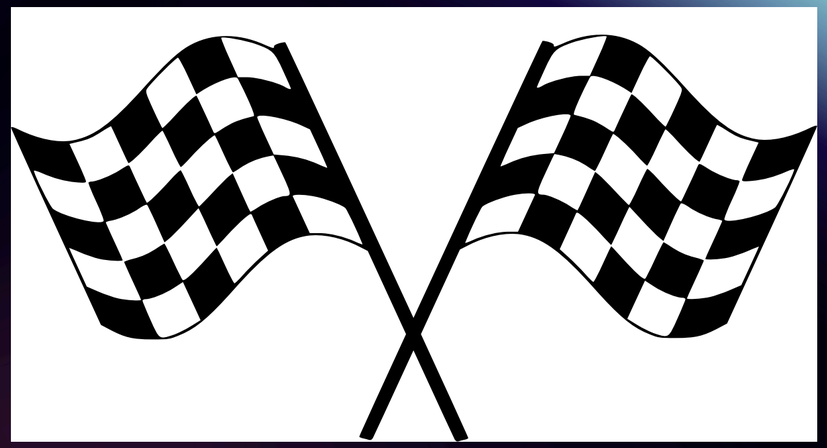
Word itself

Glove or FastText

LSTM

- Subtype of Recurrent Neural Networks (RNN). Used to recognize patterns in data sequences.
- Take as input also the embedding matrix

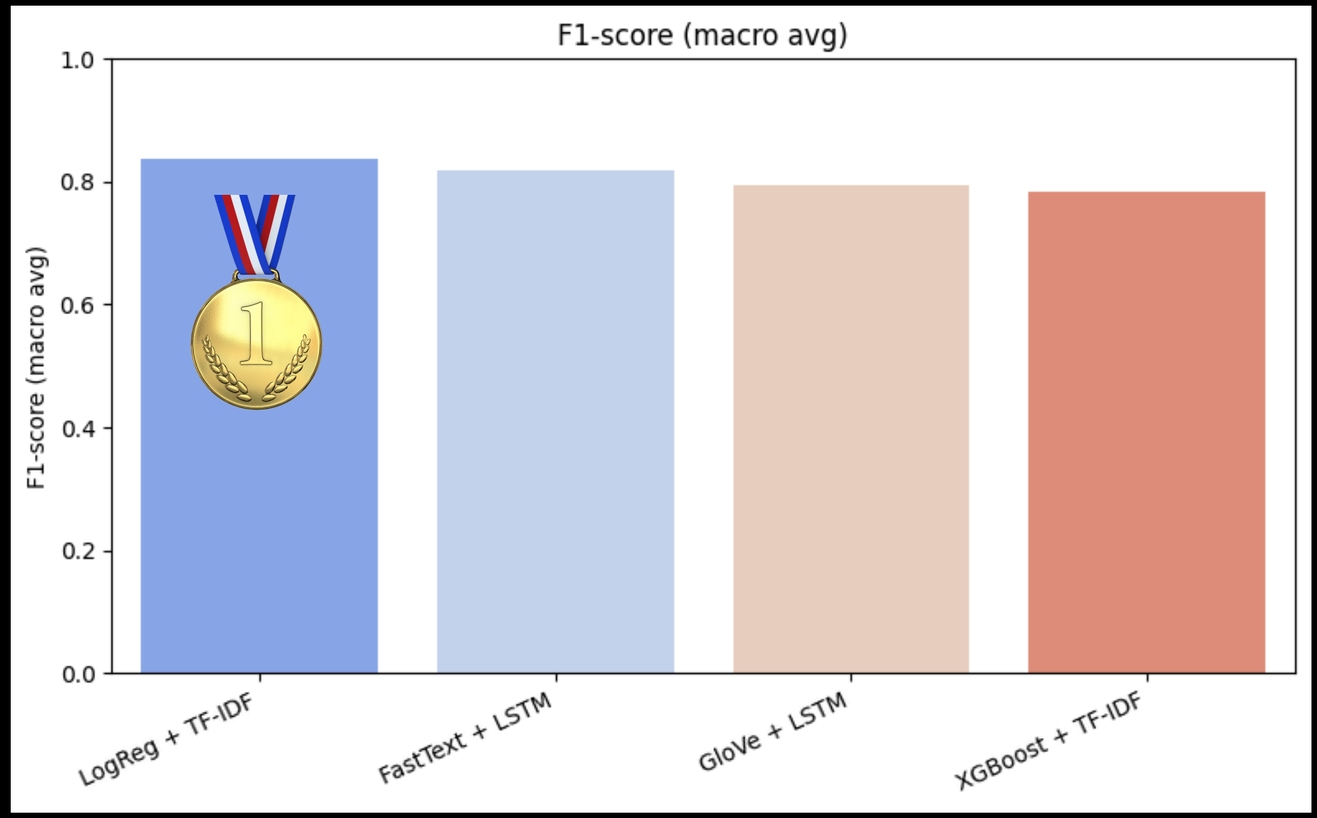
```python
def create_model(embedding_matrix,
                 LOSS = 'binary_crossentropy',
                 learning_rate=0.01,
                 hidden_act='relu',
                 dropout_rate=0.2):

    text_input = Input(shape=(max_len,), name='text_input')

    # Embedding layer suitable for both GloVe and FastText
    embedding = Embedding(input_dim=vocab_size,
                          output_dim=embedding_matrix.shape[1],
                          weights=[embedding_matrix],
                          input_length=max_len,
                          trainable=False,
                          name='embedding')(text_input)
    # RNN layer
    lstm = LSTM(units=128,
                activation='tanh',
                recurrent_activation='sigmoid',
                kernel_initializer='glorot_uniform',
                return_sequences=False,
                name='lstm')(embedding)

    # Dense layers with dropout
    x = Dense(64, activation=hidden_act, kernel_initializer='he_uniform', kernel_regularizer='l2')(lstm)
    x = Dropout(dropout_rate)(x)

    x = Dense(32, activation=hidden_act, kernel_initializer='he_uniform', kernel_regularizer='l2')(x)
    x = Dropout(dropout_rate)(x)

    # Output layer for binary classification
    output = Dense(1, activation='sigmoid', name='output')(x)

    model = Model(inputs=text_input, outputs=output)
    model.compile(
        optimizer=Adam(learning_rate=learning_rate),
        loss=LOSS,
        metrics=['accuracy']
    )

    return model
```



RESULTS

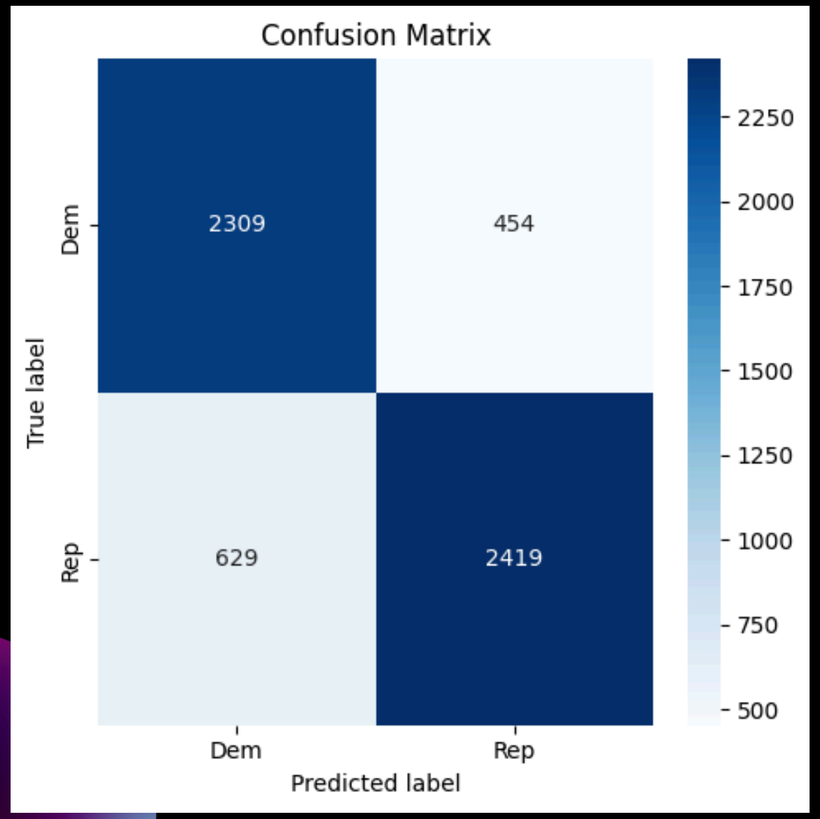
AND THE WINNER IS...

TF-IDF + Logistic Regression!

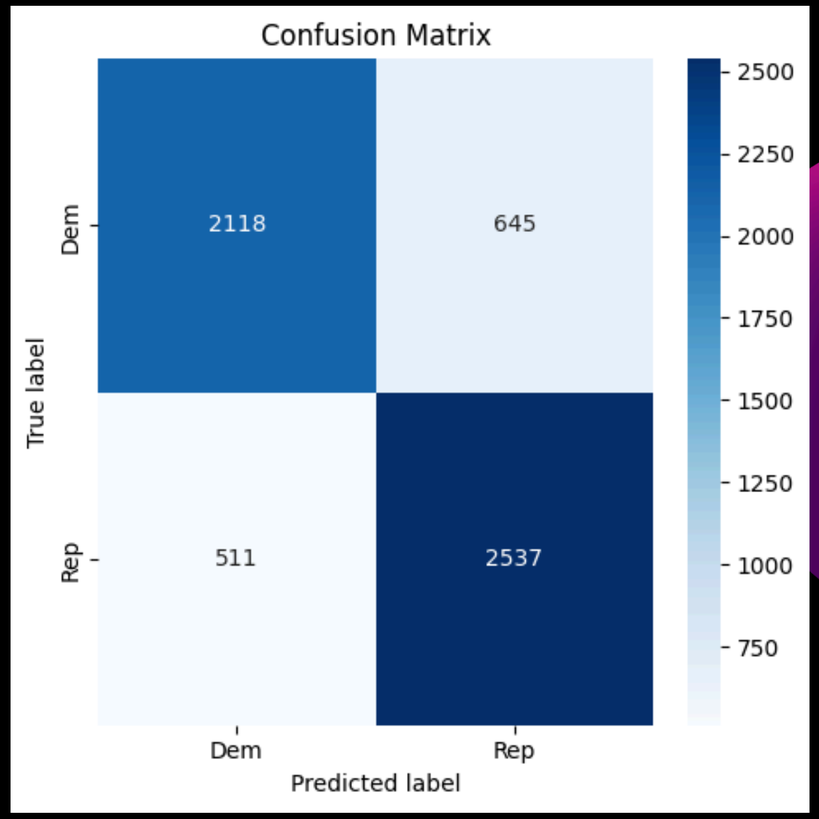


...2nd and 3rd place...

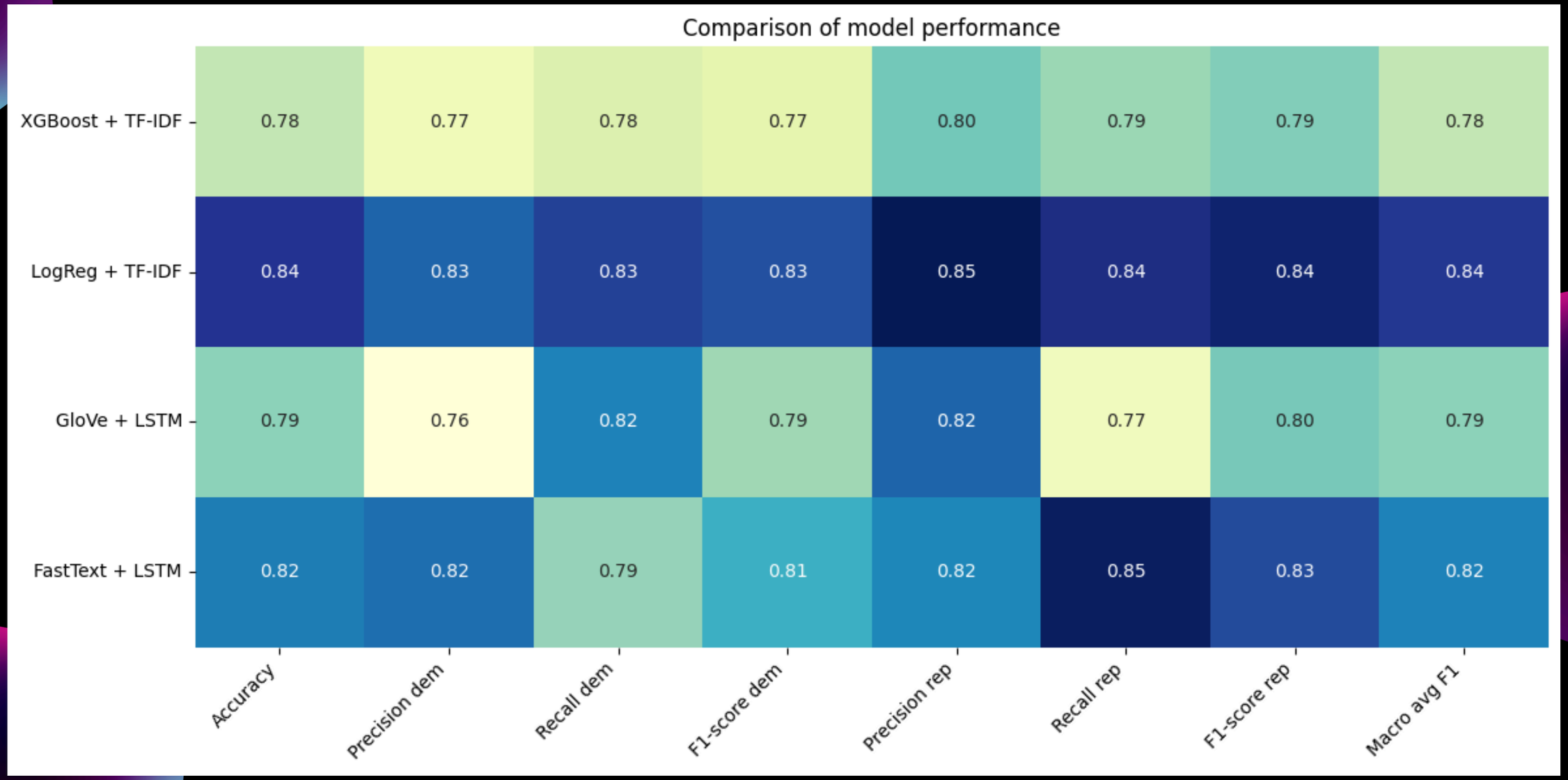
FastText + LSTM



GloVe + LSTM



RESULTS AND ANALYSIS



SOURCES

GLOVE:
HTTPS://NLP.STANFORD.EDU/PROJECTS/GLOVE/

HUGGINGFACE:
HTTPS://HUGGINGFACE.CO/DATASETS/JACOBVS/POLITICALTWEETS

CANVA:
HTTPS://WWW.CANVA.COM

DEEPNOTE:
HTTPS://DEEPNOTE.COM

THANK YOU FOR THE ATTENTION!