NLP - Fake news detection

Simbiat Musa Georg F.K. Höhn

Ironhack

19 June 2025

Project overview

Given

- dataset of headlines annotated as `fake` (0) or `real` (1)

Aim

- classify news headlines in unseen data (using machine learning)

Deliverables

- Python pipeline (+ csv with model training results)
- csv with predictions for test set

Data overview

- 34152 rows of annotated data
  - fake news: 17572
  - real news: 16580
- relatively balanced

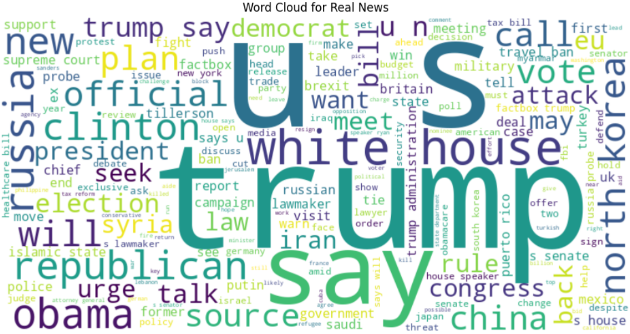

Figure 1: Word cloud for real headlines

Word cloud for data annotated as fake

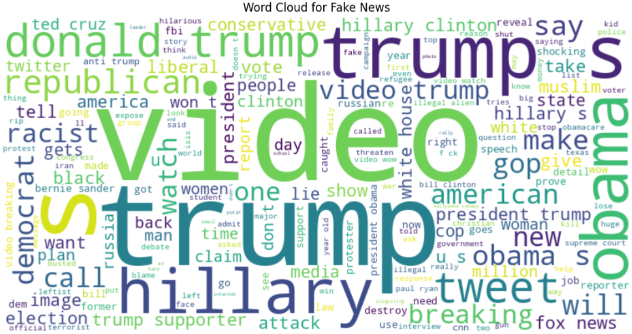

Figure 2: Word cloud for fake headlines

Methodology

Structure

- one main `.py` file for data exploration, model experimentation and output generation
- `helper.py` with functions for
  - cleaning strings
  - generate, print and save model evaluations (to csv)
  - removing stop words (not needed with TF-IDF vectoriser)
  - lemmatizer (currently not used)
  - huggingface pipeline for transformer models

General pipeline

- EDA
- clean data (generate new columns)
- train-test split (20% test size)
- vectorise train and test sets
- train (and tune?) models
- compare based on test accuracy
- run best model(s) on target data and save
- (run best models of different types on target data and calculate inter-annotator agreement)

Vectorisation

- TF-IDF
- GloVE (`glove-wiki-gigaword-100`)

ML-algorithms

- Logistic Regression (`sklearn.linear_model.LinearRegression`)
- Random Forest (`sklearn.ensemble.RandomForestClassifier`)
- KNN (`sklearn.neighbors.KNeighborsClassifier`)
- XGBoost (`xgboost.XGBClassifier`)

Regarding transformer models

- tried pipelines with
  1) `jy46604790/Fake-News-Bert-Detect`
  2) `omykhailiv/bert-fake-news-recognition`
- both performed abysmally: everything is fake, see result for 1
- over-sensitive? issue with our data pre-processing?

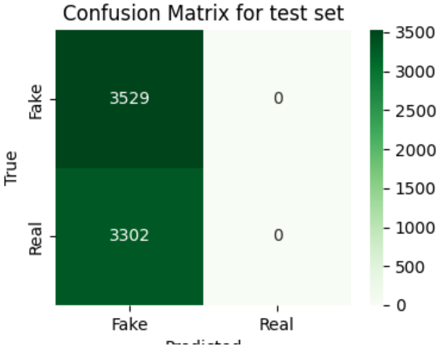

Figure 3: jy46604790/Fake-News-Bert-Detect

Training results

model_id	params	acc_train	accuracy
logreg_lemma	miter=500	0.9209	0.9133
xgb_lemma	est=500,depth=100,lr=0.3,α=0.1	0.9963	0.9130
logreg_final	miter=500	0.9172	0.9084
logreg_1000	miter=1000	0.9172	0.9084
xgb_final	est=500,depth=100,lr=0.3,α=0.1	0.9899	0.9065
xgb	est=500,depth=200,lr=0.07,α=0.1	0.9896	0.9037
xgb_2	est=200,depth=0,lr=0.1	0.9896	0.9026
rndforest_lemma	est=300,minsleaf=2	0.9409	0.9002
xgb_2	est=200,depth=50,lr=0.04	0.9407	0.8971
rndforest_final	est=300,minsleaf=2	0.9382	0.8949
xgb_1	defaults	0.9002	0.8807
logreg_glove	miter=500	0.8689	0.8703
rndforest_2	est=300,depth=30	0.8677	0.8418
xgb_2	est=10,depth=50,lr=0.04	0.8611	0.8371
knn_3	k=3	0.9154	0.8172
knn_5	k=5	0.8829	0.8037
knn_3_lemma	k=3	0.8901	0.7993
knn_10	k=10	0.8158	0.7804
omykhailiv	defaults	0.5140	0.5166
jy46604790	defaults	0.5140	0.5166

- best performing: logistic regression model with lemmatized dataset
  - max-iterations did not seem to make a difference
- xgb and RandomForest very close by
  - but: high risk of overfitting
  - longer training times/higher complexity
- knn not performing very well
  - using lemmatized dataset actually leads to drop in accuracy

Confusion matrices LogReg

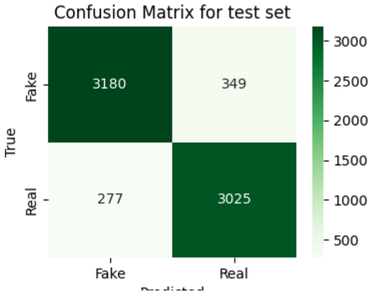

Figure 4: Confusion matrix for logistic regression model

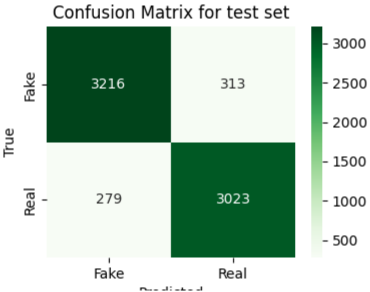

Figure 5: LogReg lemmatized

Confusion matrices XGBoost

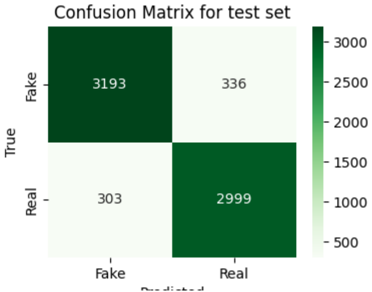

Figure 6: XGB model

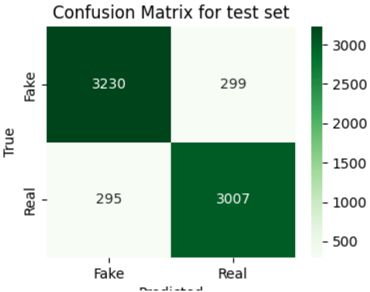

Figure 7: XGB lemmatized

Confusion matrices RandomForest

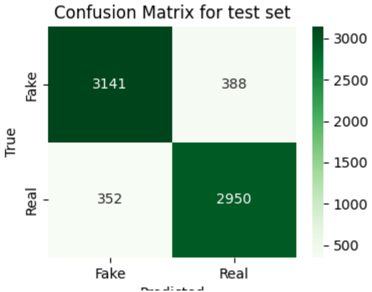

Figure 8: RandomForest

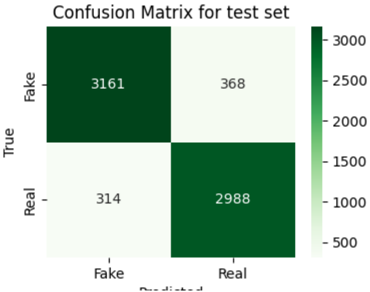

Figure 9: RandomForest lemmatized

Confusion matrices KNN

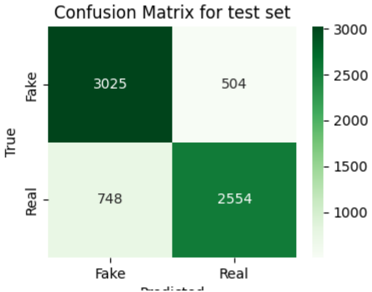

Figure 10: KNN, k=3

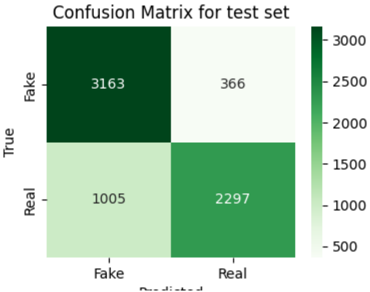

Figure 11: KNN lemmatized

Conclusion

- logistic regression model offers best performance

Collaboration

- using py files to avoid notebook consistency issues with git
- `# %%`: useful VS Code option for generating jupyter-like cells

Outlook/reflection

- further experimentation with vectoriser settings
- actually implement lemmatisation (oops)
- some confusion in our raw performance results.csv
  - some RandomForest models seemed to perform better than final model, but probably due to earlier mistakes in preprocessing?
  - lesson: also note changes to preprocessing or cleanly reset logging files

PS

- for annotation of the test set, the xgb model falls into the same issue as the transformers: everything is 0
- comparing the other three (Logistic Regression, Random Forest, KNN) yields an inter-annotator agreement Cohen's κ value of 0.32, which is 'fair agreement'

Thanks for your attention!