FAKE NEWS DETECTOR

NATURAL LANGUAGE PROCESSING

DESIREE MAESTRI
JAN DIRK
JONATHAN SADA

Summary

PROJECT CONTEXT

1. ENVIRONMENT SETUP
2. DATA LOADING AND PREPROCESSING
3. BASELINE MODEL
4. CLASSICAL MACHINE LEARNING MODEL EXPERIMENTATION
5. CURRENT BEST CLASSICAL ML MODEL
6. CLASSICAL MODEL OPTIMIZATION
7. TRANSFORMER-BASED EXPERIMENTATION
8. TRANSFER LEARNING
9. FINAL MODEL APPLICATION
10. CONCLUSION

Project context

TASK

In this project, you will put these skills into practice to identify whether a news headline is real or fake news.

Your goal is to build a classifier that is able to distinguish between the two.

DATASET

Once you have a classifier built, then use it to predict the labels for dataset/testing_data.csv. Generate a new file where the label 2 has been replaced by 0 (fake) or 1 (real) according to your model. Please respect the original file format, do not include extra columns, and respect the column separator.

DELIVERABLES

- Python Code
- Predictions
- Accuracy estimation
- Presentation

PROCESS

1. Environment Setup

TOOLS AND LIBRARIES

- pandas,
- sklearn,
- xgboost,
- transformers,
- hugginface,
- python

CUSTOM UTILITY FUNCTIONS

- helpers script
- transfer learning scripts

GLOBAL PARAMETERS

- warnings
- seed = 42

2. Data Loading and Preprocessing

2.1. INITIAL DATA INSPECTION
2.2. DATA CLEANING
2.3. DATA SPLITTING

- No code : HMTL, CSS, JS
- Removed all special characters and numbers
- X = cleaned data
- Split 20% training and testing

```
Original: trump is so obsessed he even has obama,s name coded into his website (images)
Cleaned:  trump is so obsessed he even has obama name coded into his website images
```

3. BASELINE MODEL

LEARNED IN CLASS

3. Baseline Model

	precision	recall	f1-score	support
0	0.78	0.90	0.83	3529
1	0.87	0.72	0.79	3302
accuracy			0.81	6831
macro avg	0.82	0.81	0.81	6831
weighted avg	0.82	0.81	0.81	6831

RandomForestClassifier

▼ Parameters

n_estimators	200
criterion	'entropy'
max_depth	None
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_features	'sqrt'
max_leaf_nodes	None
min_impurity_decrease	0.0
bootstrap	True
oob_score	False
n_jobs	-1
random_state	42
verbose	0
warm_start	False
class_weight	None
ccp_alpha	0.0
max_samples	None
monotonic_cst	None

4. CLASSICAL ML MODEL EXPERIMENTATION

4.1. Classical ML Model Experimentation

- KNeighbors
- LogisticRegression
- DecisionTree
- RandomForest
- AdaBoost
- XGBoost
- BernoulliNB

GLOBAL SETUP

DATA SOURCE

No special characters, numbers, single letters

split 80-20

4.2. Performance Summary

SETUP			TRAIN RESULTS				TEST RESULTS			
model	vectorizer	fit_time	accuracy_train	precision_train	recall_train	f1_train	accuracy_test	precision_test	recall_test	f1_test
KNeighborsClassifier	CountVectorizer	0.00078	0.60730	0.95502	0.20146	0.33273	0.57385	0.89655	0.13386	0.23294
LogisticRegression	**CountVectorizer**	**4.30651**	**0.98122**	**0.97620**	**0.98539**	**0.98077**	**0.94935**	**0.93962**	**0.95669**	**0.94808**
DecisionTreeClassifier	CountVectorizer	2.37056	1.0	1.0	1.0	1.0	0.87937	0.88170	0.86675	0.87416
RandomForestClassifier	CountVectorizer	1.10570	1.0	1.0	1.0	1.0	0.92973	0.93847	0.91460	0.92638
AdaBoostClassifier	CountVectorizer	0.54147	0.77351	0.69457	0.95308	0.80354	0.78232	0.70109	0.95821	0.80972
XGBClassifier	CountVectorizer	0.30768	0.91684	0.88067	0.95880	0.91808	0.90807	0.87451	0.94549	0.90861
BernoulliNB	CountVectorizer	0.00299	0.95271	0.93976	0.96453	0.95198	0.94481	0.93333	0.95397	0.94354
KNeighborsClassifier	TfidfVectorizer	0.00092	0.93038	0.91941	0.93907	0.92914	0.89431	0.87566	0.91066	0.89281
LogisticRegression	TfidfVectorizer	2.04005	0.96175	0.95297	0.96912	0.96098	0.94481	0.93180	0.95578	0.94364
DecisionTreeClassifier	TfidfVectorizer	2.21042	1.0	1.0	1.0	1.0	0.88333	0.87221	0.88886	0.88046
RandomForestClassifier	TfidfVectorizer	0.96668	1.0	1.0	1.0	1.0	0.93442	0.92445	0.94125	0.93277
AdaBoostClassifier	TfidfVectorizer	1.46798	0.79144	0.71554	0.94758	0.81537	0.79857	0.71996	0.95457	0.82083
XGBClassifier	TfidfVectorizer	2.97018	0.93247	0.90311	0.96453	0.93281	0.91802	0.88663	0.95215	0.91822
BernoulliNB	TfidfVectorizer	0.00307	0.95271	0.93976	0.96453	0.95198	0.94481	0.93333	0.95397	0.94354
KNeighborsClassifier	**HashingVectorizer**	**0.00087**	**0.92314**	**0.91659**	**0.92612**	**0.92133**	**0.87923**	**0.86893**	**0.88340**	**0.87611**
LogisticRegression	HashingVectorizer	7.91764	0.94835	0.93510	0.96039	0.94758	0.93442	0.91872	0.94821	0.93323
DecisionTreeClassifier	HashingVectorizer	9.16467	1.0	1.0	1.0	1.0	0.89299	0.88569	0.89400	0.88983
RandomForestClassifier	HashingVectorizer	83.24285	1.0	1.0	1.0	1.0	0.94862	0.93848	0.95639	0.94735
AdaBoostClassifier	HashingVectorizer	9.12883	0.80048	0.72882	0.93877	0.82058	0.80471	0.73251	0.93882	0.82294
XGBClassifier	HashingVectorizer	30.16289	0.93397	0.90339	0.96762	0.9344	0.92241	0.89286	0.95397	0.92240
BernoulliNB	HashingVectorizer	0.01897	0.76622	0.99370	0.52229	0.68470	0.75187	0.98904	0.49213	0.65723

4.3. Top-Performing Models x N-Grams

SETUP				TRAIN RESULTS				TEST RESULTS			
model	vectorizer	ngram_range	fit_time	accuracy_train	precision_train	recall_train	f1_train	accuracy_test	precision_test	recall_test	f1_test
LogisticRegression	CountVectorizer	(1, 1)	4.80231	0.98122	0.97620	0.98539	0.98077	0.94935	0.93962	0.95669	0.94808
LogisticRegression	**CountVectorizer**	**(1, 2)**	**4.44516**	**0.99824**	**0.99685**	**0.99955**	**0.99819**	**0.95286**	**0.94083**	**0.96305**	**0.95181**
LogisticRegression	CountVectorizer	(2, 2)	3.71909	0.99535	0.99163	0.99887	0.99524	0.90119	0.86496	0.94276	0.90219
LogisticRegression	CountVectorizer	(1, 3)	4.37791	0.99938	0.99872	1.0	0.99936	0.95023	0.93739	0.96124	0.94916
LogisticRegression	CountVectorizer	(2, 3)	4.48142	0.99795	0.99580	1.0	0.99790	0.89709	0.85652	0.94549	0.89881
LogisticRegression	CountVectorizer	(3, 3)	4.33108	0.99521	0.99023	1.0	0.99509	0.76958	0.68693	0.96154	0.80136
RandomForestClassifier	HashingVectorizer	(1, 1)	85.71581	1.0	1.0	1.0	1.0	0.94862	0.93848	0.95639	0.94735
RandomForestClassifier	HashingVectorizer	(1, 2)	58.55281	1.0	1.0	1.0	1.0	0.94598	0.93815	0.95094	0.94450
RandomForestClassifier	HashingVectorizer	(2, 2)	159.77131	0.99982	1.0	0.99962	0.99981	0.83399	0.91950	0.71956	0.80734
RandomForestClassifier	HashingVectorizer	(1, 3)	48.10105	1.0	1.0	1.0	1.0	0.94510	0.94335	0.94306	0.94321
RandomForestClassifier	HashingVectorizer	(2, 3)	138.37493	0.99982	1.0	0.99962	0.99981	0.82550	0.91766	0.70200	0.79547
RandomForestClassifier	HashingVectorizer	(3, 3)	213.39028	0.99908	1.0	0.99812	0.99906	0.63534	0.93369	0.26439	0.41208

5. CURRENT TOP CLASSICAL MODEL

DATA SOURCE	Cleaned 80-20 split
MODEL	LogisticRegression
FEATURE VECTORIZATION	CountVectorizer
N-GRAM CONFIGURATION	1, 2
RESULTS	Accuracy (Train) 0.98 Recall (Train) 0.98 Accuracy (test) 0.94 Recall (Test) 0.95

6. TOP CLASSICAL MODEL OPTIMIZATION

6.1. Model Quality Validation

- After removing the words iterativelly based on the coeficient of the words we see a decrease in accuracy (0.844 by keeping only words with coeficient inbetween -1 and 1)
- The recall stays relevant in between 0.937 and 0.963

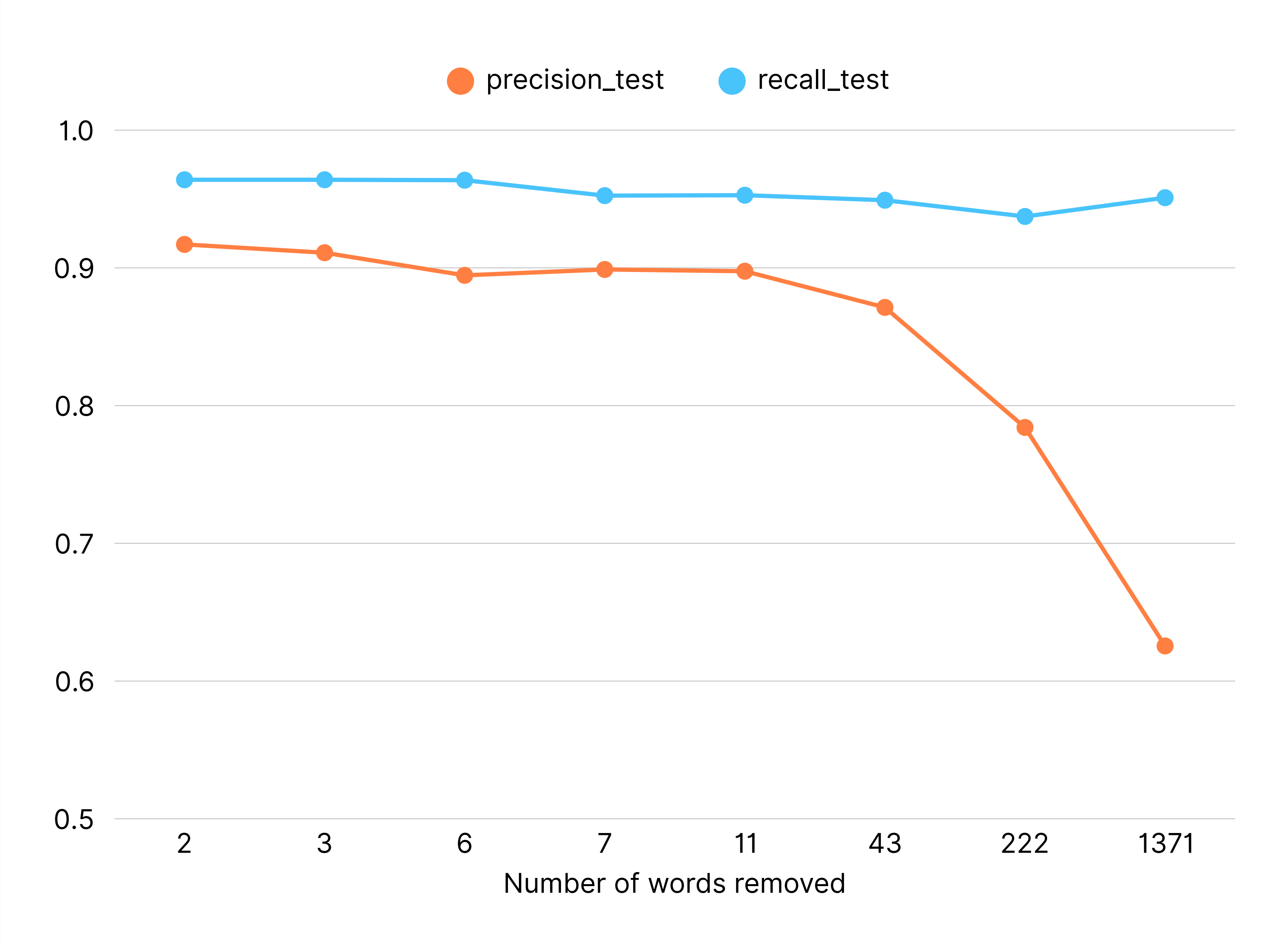
SETUP			TRAIN RESULTS				TEST RESULTS			
coef_thld_rm	num_words_rm	prop_words_rm	accuracy_train	precision_train	recall_train	f1_train	accuracy_test	precision_test	recall_test	f1_test
4	**2**	0.000011832779	0.998243109695	0.996845425867	0.999548124717	0.998194945848	**0.940418679549**	0.917026793431	**0.963961235614**	0.939908460062
3.5	**3**	0.000017749168	0.998243109695	0.996845425867	0.999548124717	0.998194945848	**0.937051676182**	0.910990269032	**0.963961235614**	0.936727486756
3	**6**	0.000035498337	0.998243109695	0.996845425867	0.999548124717	0.998194945848	**0.927536231884**	0.894574079280	**0.963658388855**	0.927832045487
2.5	**7**	0.000041414727	0.998243109695	0.996845425867	0.999548124717	0.998194945848	**0.925193968672**	0.898828236639	**0.952453058752**	0.924863990589
2	**11**	0.000065080285	0.998243109695	0.996845425867	0.999548124717	0.998194945848	**0.924608402869**	0.897574893009	**0.952755905511**	0.924342588511
1.5	**43**	0.000254404752	0.998243109695	0.996845425867	0.999548124717	0.998194945848	**0.907626994583**	0.871281623575	**0.949121744397**	0.908537469198
1	**222**	0.001313438487	0.998243109695	0.996845425867	0.999548124717	0.998194945848	**0.844971453667**	0.784139853052	**0.937310720775**	0.853910884259
0.5	**1371**	0.008111370117	0.998243109695	0.996845425867	0.999548124717	0.998194945848	**0.701068657590**	0.625498007968	**0.950938824954**	0.754626291756

	word	coef
49321	factbox	3.182219
127033	says	2.709662
157549	urges	2.001461
147551	tillerson	1.741403
82476	lawmakers	1.733302
141728	talks	1.730247
26569	china	1.727194
47074	eu	1.723963
168852	zimbabwe	1.693987
136309	spokesman	1.622523

	word	coef
158962	video	-4.307516
19445	breaking	-4.102125
60141	gop	-3.593094
66407	hillary	-3.400183
79005	just	-3.086532
117247	racist	-2.433639
38121	dem	-2.108554
69683	huge	-2.043482
167531	wow	-1.976108
18423	bombshell	-1.911474

6.1. Model Quality Validation

As we remove more words, the precision decreases; however, the recall metric remains high. This indicates that the model can make relevant predictions even without the words that have higher coefficients in the model.



6.2. Alternative Text Cleaning

SETUP				TRAIN RESULTS				TEST RESULTS			
model	vectorizer	fit_time	cleaning	accuracy_train	precision_train	recall_train	f1_train	accuracy_test	precision_test	recall_test	f1_test
LogisticRegression	CountVectorizer	4.65329480171203	less_cleaning	0.998426119	0.997219926	0.999548124	0.998382668	0.952715561	0.94028968371	0.963355542	0.951682872
LogisticRegression	CountVectorizer	4.53271770477294	clean_serie	0.998243109	0.996845425	0.999548124	0.998194945	0.952861952	0.94082840236	0.963052695	0.951810835
LogisticRegression	CountVectorizer	4.93542718887329	no_cleanning	0.998426119	0.997219926	0.999548124	0.998382668	0.952569169	0.93975191966	0.963658388	0.951555023

```
'americans once elected a president after he was accused of raping a 13-year-old girl'
```

7. TRANSFORMER-BASED MODELS EXPERIMENTATION

7.3. Performance summary

model	accuracy	precision	recall	f1
mrm8488/bert-tiny-finetuned-fake-news-detection	0.465214335910049	0.474819068126084	0.957599517490953	0.634851453476748
jy46604790/Fake-News-Bert-Detect	0.652319044272663	0.983162217659137	0.288781664656212	0.446433566433566
yasmine-11/distilbert_fake_news	0.484425899491233	0.485064695009242	0.988175930109956	0.650714144019043

7.3. Performance summary

model	accuracy	precision	recall	f1
mrm8488/bert-tiny-finetuned-fake-news-detection	0.465214335910049	0.474819068126084	0.957599517490953	0.634851453476748
jy46604790/Fake-News-Bert-Detect	0.652319044272663	0.983162217659137	0.288781664656212	0.446433566433566
yasmine-11/distilbert_fake_news	0.484425899491233	0.485064695009242	0.988175930109956	0.650714144019043

8. TRANSFER LEARNING EVALUATION

8.1. Transfer learning configuration

DATA SOURCE	`Cleaned + 80-20 split`	
MODELS	**`distilbert-base-uncased`**	**`bert-base-uncased`**
EVALUATED ON	`eval_recall`	

RESULTS	`{'eval_loss': 0.07611262053251266,`
	`'eval_accuracy': 0.9817010686575904,`
	`'eval_recall': 0.9766807995154452,`
	`'eval_runtime': 49.1314,`
	`'eval_samples_per_second': 139.035,`
	`'eval_steps_per_second': 8.691,`
	`'epoch': 3.0}`
		`{'eval_loss': 0.09613175690174103,`
		`'eval_accuracy': 0.9822866344605475,`
		`'eval_recall': 0.9887946698970321,`
		`'eval_runtime': 27.6307,`
		`'eval_samples_per_second': 247.225,`
		`'eval_steps_per_second': 15.454,`
		`'epoch': 3.0}`

9. FINAL MODEL APPLICATION

9.1. Champion Classical Model

No special characters, numbers, single letters

LogisticRegression

CountVectorizer

N-gram: 1, 2

TRAIN

ACCURACY: **0.998**

RECALL: **0.999**

TEST

ACCURACY: **0.952**

RECALL: **0.963**

We have evaluated that False Negatives are the most expensive result for this use case, so we decided to **focus on Recall**.

A Newspapper use or model to filter the news to publish.

- Publishing a Fake New would damage the newspapper credibility.
- Not publish a Real New would not damage the newspapper.

Conclusions

CHALLENGES

- Not setting-up a common base environment
- Collaboration tools
- Try the models on other data and make it work for that too
- Different experiments simultaneously make it harder to structure the work and compile the code

LEARNINGS

- Learned about transfer learning
- How to validate the model, by generalizing it
- How to optimize experiments with different models to deep dive only into the best one
- How classical models and pre-treined models work and what is best for each type of dataset

FUTURE WORK

- Test Lemmatization
- Test Stop Words Removal
- Hyperparametter tunning
- Train the model on spam emails from different languages to improve global accuracy.
- Use real-world email examples to better prepare the model for specific industries.
- Regularly check and update the model as spammers change their tactics over time.
- Include extra info like sender name or time of day to help the model detect spam.
- Test how well the model handles brand-new types of spam without retraining it first.

THANKYOU

QUESTIONS?