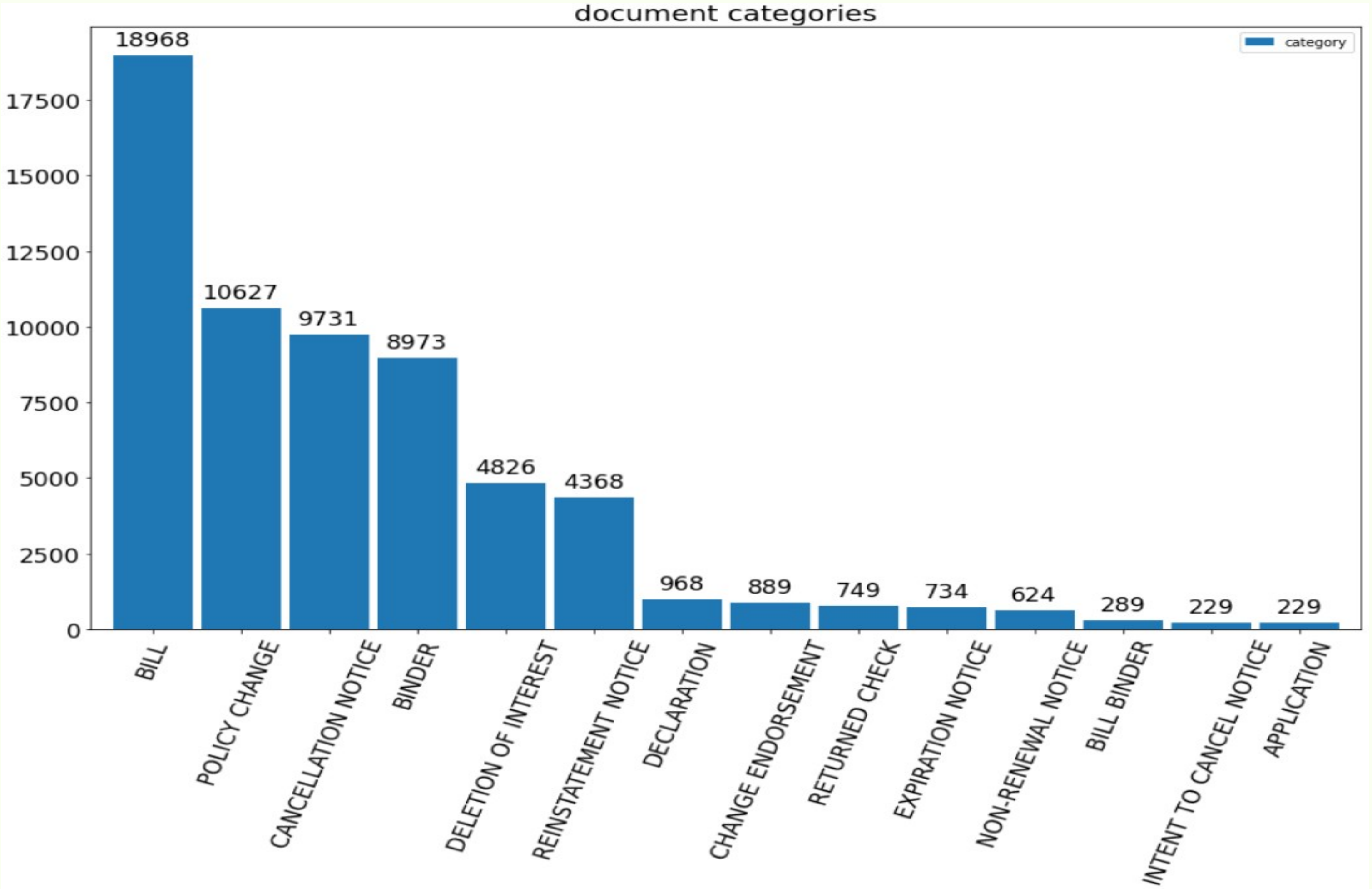
HeavyWater Machine Learning Problem

Solution by Mark Wilber

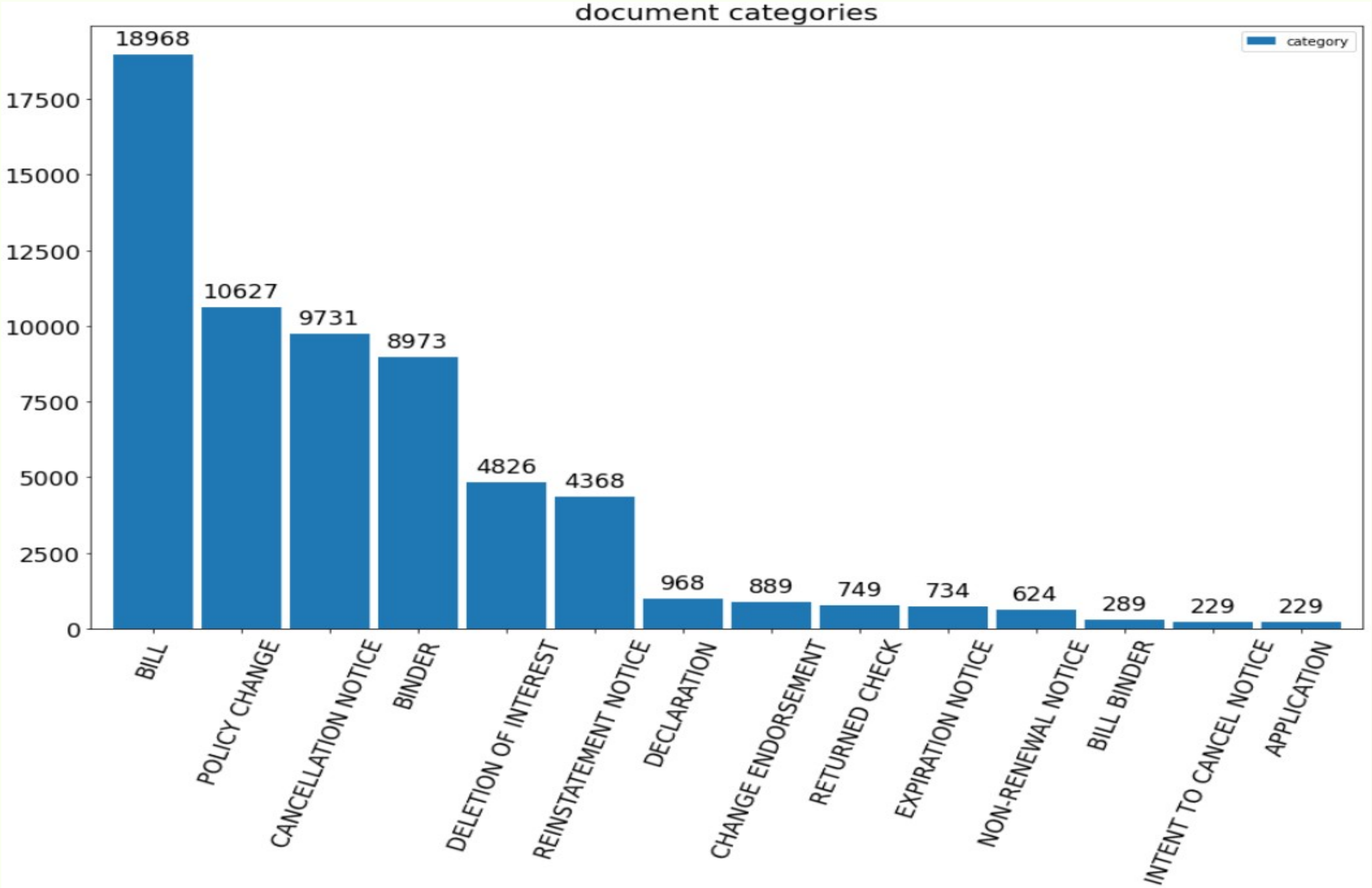
What we are dealing with

- 62 K documents, 14 categories



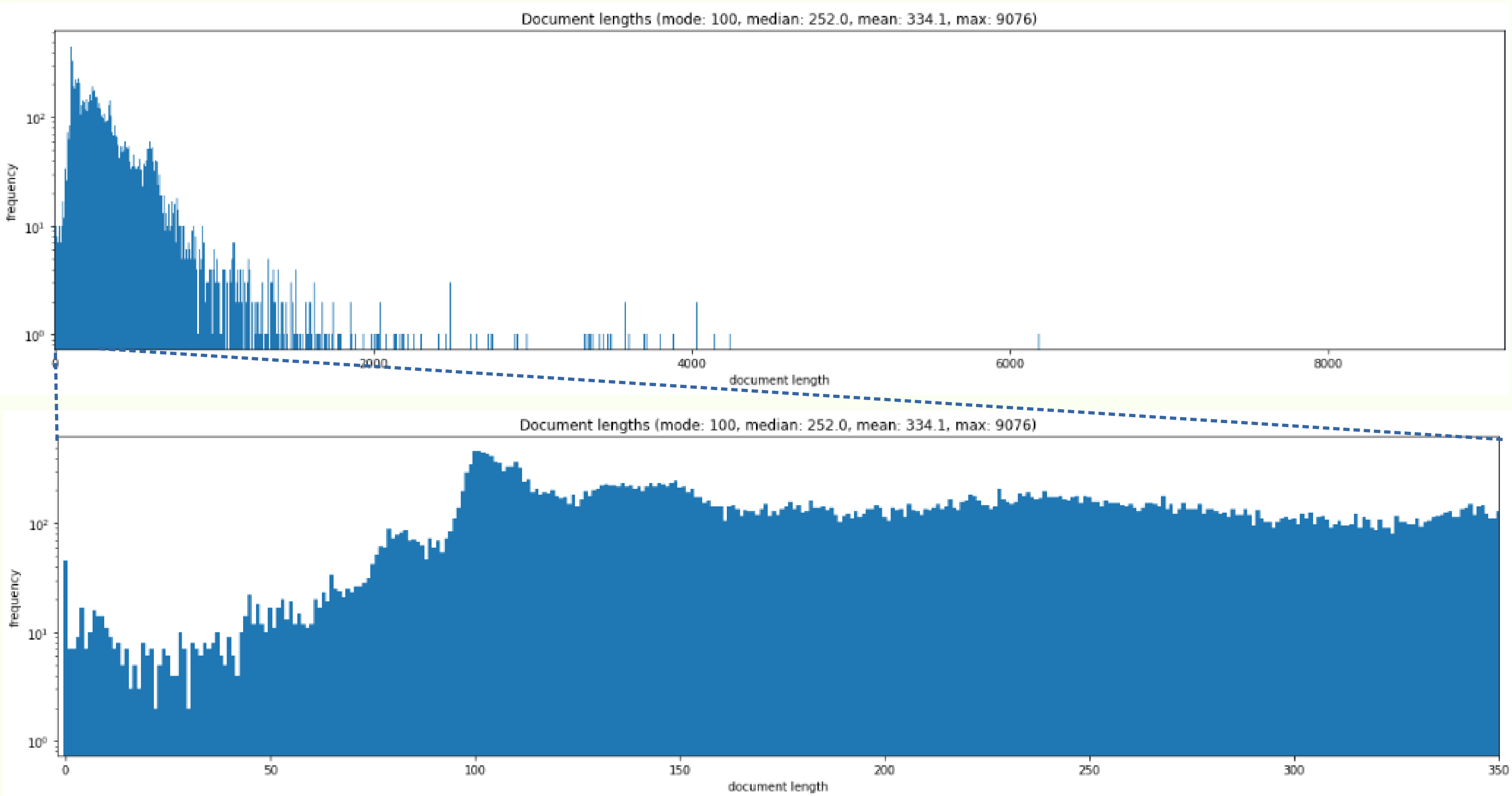
What we are dealing with

- 62 K documents, 14 categories
- unbalanced classes, spanning nearly 2 orders of magnitude



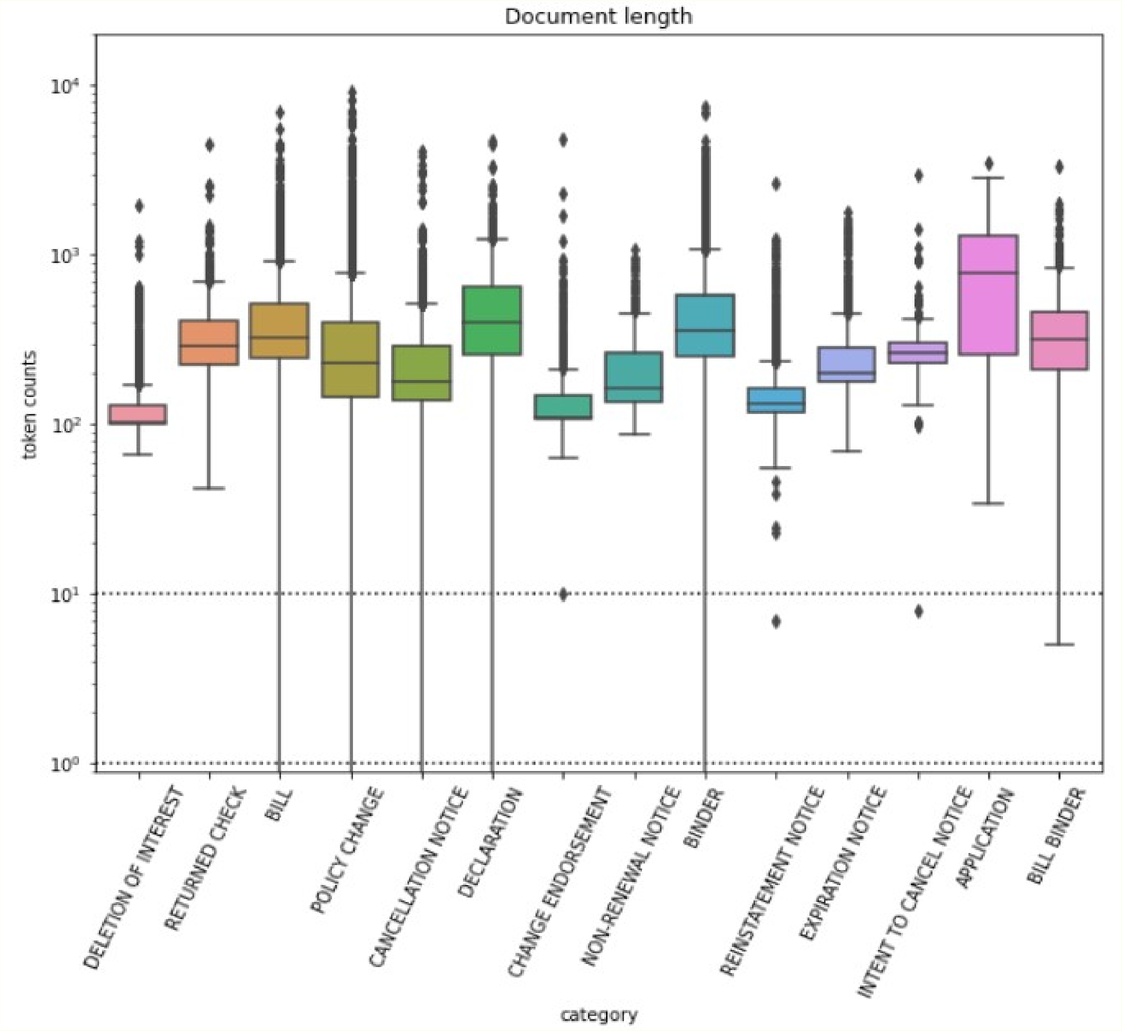
What we are dealing with

- document lengths spanning 0–9076 tokens (mode: 100, median: 252, mean: 334.1)



What we are dealing with

- document lengths vary widely by category, but few are shorter than 10 tokens



What we are dealing with

- 1,037,933 *unique* tokens!

What we are dealing with

- 1,037,933 *unique* tokens!
- contrasts with: entire English language

Problem vocabulary exceeds that of OED:

> Oxford Dictionary has 273,000 headwords; 171,476 of them being in current use, 47,156 being obsolete words and around 9,500 derivative words included as subentries. The dictionary contains 157,000 combinations and derivatives in bold type, and 169,000 phrases and combinations in bold italic type, making a total of over 600,000 word-forms. There is one count that puts the English vocabulary at about 1 million words — but that count presumably includes words such as Latin species names, prefixed and suffixed words, scientific terminology, jargon, foreign words of extremely limited English use and technical acronyms.

What we are dealing with

- 1,037,933 *unique* tokens!
- contrasts with: entire English language

Problem vocabulary exceeds that of OED:

> Oxford Dictionary has 273,000 headwords; 171,476 of them being in current use, 47,156 being obsolete words and around 9,500 derivative words included as subentries. The dictionary contains 157,000 combinations and derivatives in bold type, and 169,000 phrases and combinations in bold italic type, making a total of over 600,000 word-forms. There is one count that puts the English vocabulary at about 1 million words — but that count presumably includes words such as Latin species names, prefixed and suffixed words, scientific terminology, jargon, foreign words of extremely limited English use and technical acronyms.

⇨ *very* unlikely ∃ so much variation in the lexicon of mortgages and loans!

What we are dealing with

- consider terms occurring with *lowest* frequencies

tf	rank	# ≥ rank	frac ≥ rank
6	77189	960745	0.925632
5	88316	949618	0.914912
4	103088	934846	0.900680
3	128487	909447	0.876209
2	172658	865276	0.833652
1	300995	736939	0.710006

- explanation: most of the tokens are "uninformative" (garbage)

What we are dealing with

- Consider terms occurring with lowest frequencies

tf	rank	# ≥ rank	frac ≥ rank
6	77189	960745	0.925632
5	88316	949618	0.914912
4	103088	934846	0.900680
3	128487	909447	0.876209
2	172658	865276	0.833652
1	300995	736939	0.710006

- explanation: most of the tokens are “uninformative” (garbage)
  - 71% of tokens only appear once,

What we are dealing with

- Consider terms occurring with lowest frequencies

tf	rank	# ≥ rank	frac ≥ rank
6	77189	960745	0.925632
5	88316	949618	0.914912
4	103088	934846	0.900680
3	128487	909447	0.876209
2	172658	865276	0.833652
1	300995	736939	0.710006

- <u>explanation</u>: most of the tokens are “uninformative” (garbage)
    - 71% of tokens only appear once, <u>92.6% occur 6 × or fewer</u>

What we are dealing with

- Consider terms occurring with lowest frequencies

tf	rank	# ≥ rank	frac ≥ rank
6	77189	960745	0.925632
5	88316	949618	0.914912
4	103088	934846	0.900680
3	128487	909447	0.876209
2	172658	865276	0.833652
1	300995	736939	0.710006

- explanation: most of the tokens are “uninformative” (garbage)
  - 71% of tokens only appear once, 92.6% occur 6 × or fewer
  - a small fraction are names (of humans, businesses), special codes

What we are dealing with

- Consider terms occurring with lowest frequencies

tf	rank	# ≥ rank	frac ≥ rank
6	77189	960745	0.925632
5	88316	949618	0.914912
4	103088	934846	0.900680
3	128487	909447	0.876209
2	172658	865276	0.833652
1	300995	736939	0.710006

- explanation: most of the tokens are “uninformative” (garbage)
  - 71% of tokens only appear once, 92.6% occur 6 × or fewer
  - a small fraction are names (of humans, businesses), special codes
  - if each numerical value has a token (rather than each digit), that could account for many (but not all), supposing an average of 10 unique values in each of document, 62 k documents, with some repetition

What we are dealing with

- Consider terms occurring with lowest frequencies

tf	rank	# ≥ rank	frac ≥ rank
6	77189	960745	0.925632
5	88316	949618	0.914912
4	103088	934846	0.900680
3	128487	909447	0.876209
2	172658	865276	0.833652
1	300995	736939	0.710006

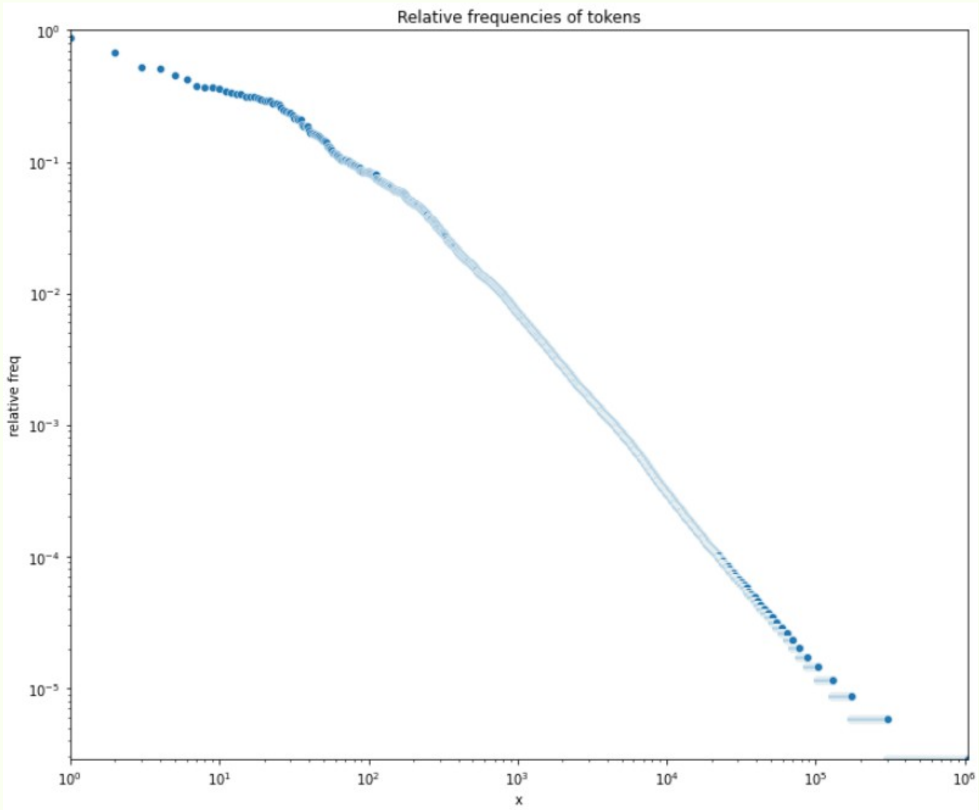
- explanation: most of the tokens are “uninformative” (garbage)
  - 71% of tokens only appear once, 92.6% occur 6 × or fewer
  - A small fraction are names (of humans, businesses), special codes
  - if each numerical value has a token (rather than each digit), that could account for many (but not all), supposing an average of 10 unique values in each of document, 62 k documents, with some repetition

⇨ *speculation:* rarely occurring terms are bogus, due to scan / OCR noise

⇨ smudges create nonsense terms

What we are dealing with

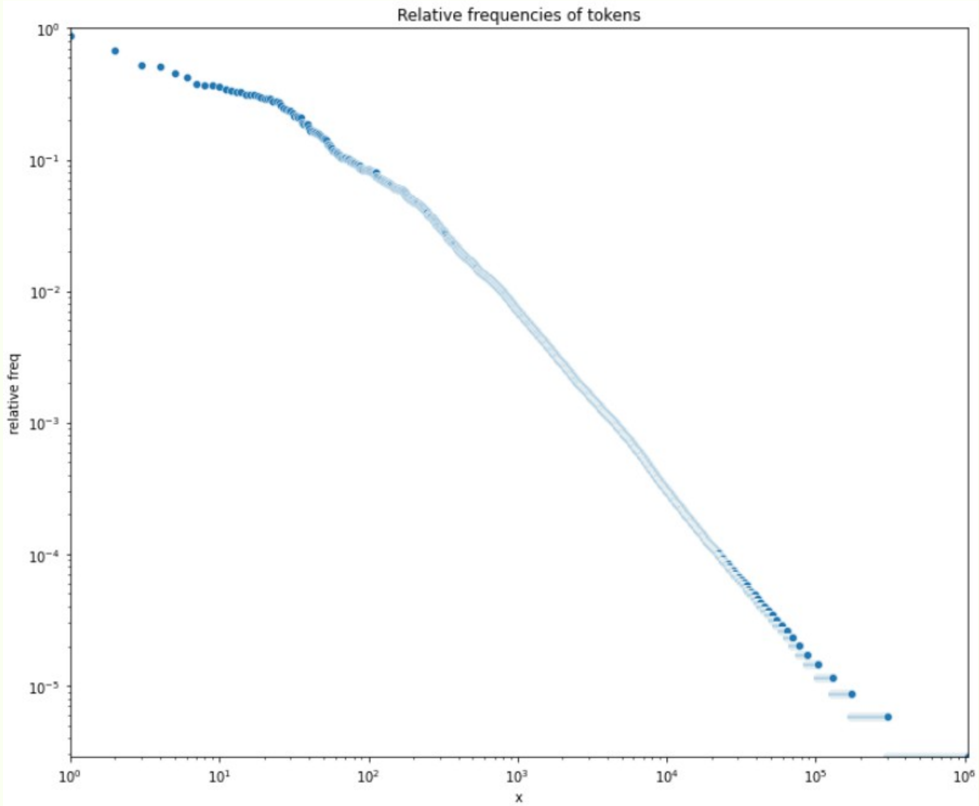
- Most frequent terms don't follow Zipf's relation



- first ~25 tokens frequency declines weakly vs Zipf

What we are dealing with

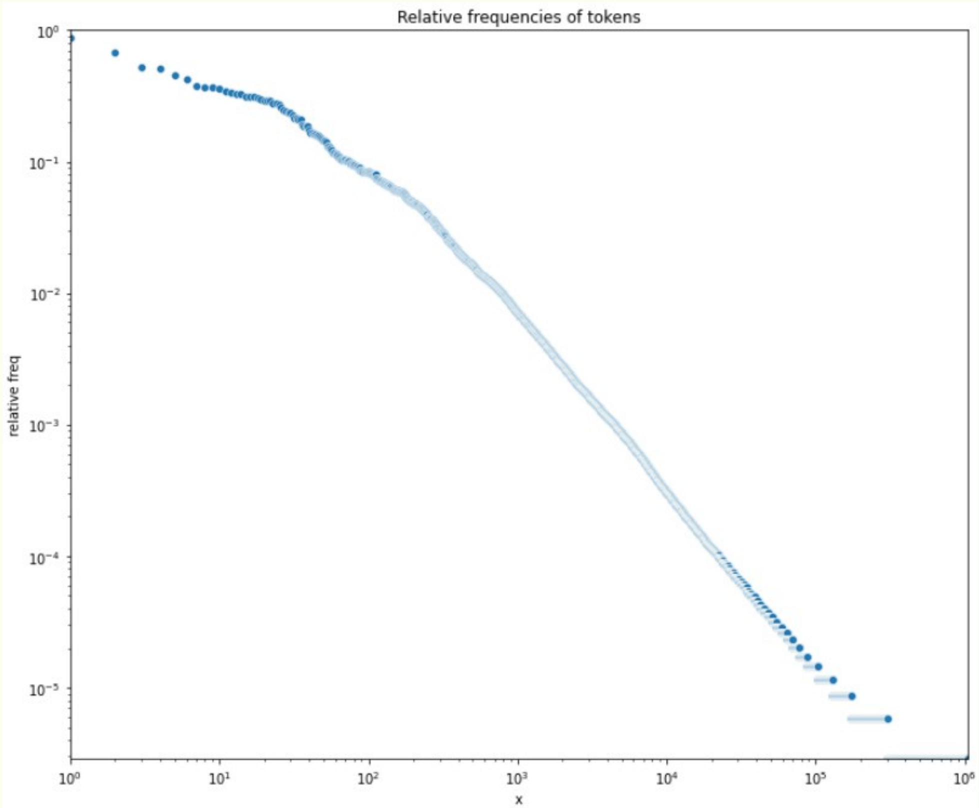
- Most frequent terms don't follow Zipf's relation



- first ~25 tokens frequency declines weakly vs Zipf (25th ranked should be 4×10^{-2})

What we are dealing with

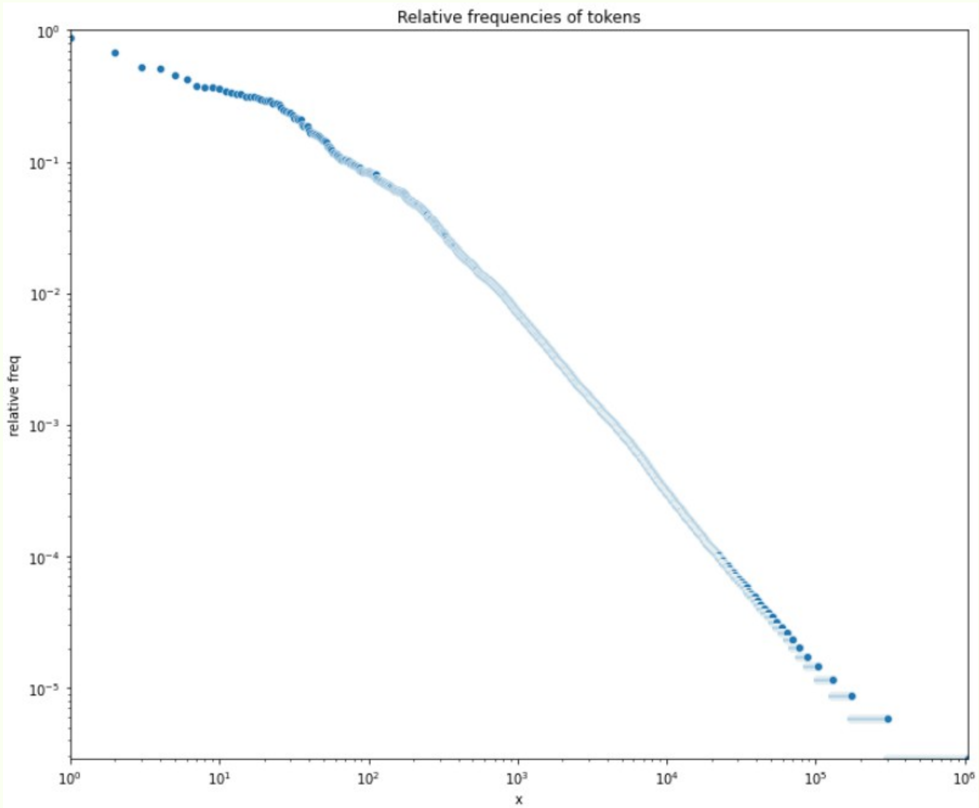
- Most frequent terms don't follow Zipf's relation



- first ~25 tokens frequency declines weakly vs Zipf (25th ranked should be 4×10^{-2})
- after 750th ranked token, looks OK

What we are dealing with

- Most frequent terms don't follow Zipf's relation



- first ~25 tokens frequency declines weakly vs Zipf (25th ranked should be 4×10^{-2})
- after 750th ranked token, looks OK

⇨ *this corpus seems to be unusual ...*

Handling Data

Problem with stop words

- can’t use curated lists for stop words, as we only have word hashes

Handling Data

Problem with stop words

- can’t use curated lists for stop words, as we only have word hashes
- test with `sklearn.feature_extraction.text.TfidfVectorizer` shows: `max_df=0.80` eliminates 9 tokens, but I can’t guess what they are. *Probably* stop words ...

Handling Data

Problem with stop words

- can't use curated lists for stop words, as we only have word hashes
- test with `sklearn.feature_extraction.text.TfidfVectorizer` shows: `max_df=0.80` eliminates 9 tokens, but I can't guess what they are. *Probably* stop words ..
- given time and *justification*, could use statistical techniques, e.g.:

  Gerlach, M., Shi, H. & Amaral, L.A.N. A universal information theoretic approach to the identification of stopwords. Nat Mach Intell 1, 606–612 (2019). https://doi.org/10.1038/s42256-019-0112-6

Handling Data

Trouble with small classes

- with smallest class sizes $\mathcal{O}(200)$, even train-test split yields ~10% uncertainty in test stats

Handling Data

Trouble with small classes

- with smallest class sizes $\mathcal{O}(200)$, even train-test split yields ~10% uncertainty in test stats
- ⇨ can’t be sure cross-validation picks best model

Handling Data

Trouble with small classes

- with smallest class sizes $\mathcal{O}(200)$, even train-test split yields ~10% uncertainty in test stats
- ⇨ can’t be sure cross-validation picks best model
- ⇨ can’t trust relative scores between techniques

Handling Data

Trouble with small classes

- with smallest class sizes $\mathcal{O}(200)$, even train-test split yields ~10% uncertainty in test stats
- ⇨ can't be sure cross-validation picks best model
- ⇨ can't trust relative scores between techniques

Many documents seem too short

- what financial information can be conveyed in 10 terms?

Handling Data

Trouble with small classes

- with smallest class sizes $\mathcal{O}(200)$, even train-test split yields ~10% uncertainty in test stats
- ⇨ can't be sure cross-validation picks best model
- ⇨ can't trust relative scores between techniques

Many documents seem too short

- what financial information can be conveyed in 10 terms?

Test-train split

- 1st removed documents of length < 10 (still retaining very short examples)

Handling Data

Trouble with small classes

- with smallest class sizes $\mathcal{O}(200)$, even train-test split yields ~10% uncertainty in test stats
- ⇨ can’t be sure cross-validation picks best model
- ⇨ can’t trust relative scores between techniques

Many documents seem too short

- what financial information can be conveyed in 10 terms?

Test-train split

- 1st removed documents of length < 10 (still retaining very short examples)
- stratified sampling

Handling Data

Trouble with small classes

- with smallest class sizes $\mathcal{O}(200)$, even train-test split yields ~10% uncertainty in test stats
- ⇨ can't be sure cross-validation picks best model
- ⇨ can't trust relative scores between techniques

Many documents seem too short

- what financial information can be conveyed in 10 terms?

Test-train split

- 1st removed documents of length < 10 (still retaining very short examples)
- stratified sampling
- 50-50 test-train split to retain plausible stats on results, at some cost to performance …

Handling Data

Trouble with small classes

- with smallest class sizes $\mathcal{O}(200)$, even train-test split yields ~10% uncertainty in test stats
- ⇨ can't be sure cross-validation picks best model
- ⇨ can't trust relative scores between techniques

Many documents seem too short

- what financial information can be conveyed in 10 terms?

Test-train split

- 1st removed documents of length < 10 (still retaining very short examples)
- stratified sampling
- 50-50 test-train split to retain plausible stats on results, at some cost to performance …
- after model selection, could train on full data set (but wouldn't know how much better the results)

Handling Data

tf-idf features

- `min_df=5:` ⇨ Eliminates most vocabulary

Handling Data

tf-idf features

- `min_df=5:` ⇨ Eliminates most vocabulary
- `ngram_range=(1, 2):` ⇨ 283 k vocabulary

Handling Data

tf-idf features

- `min_df=5:` ⇨ Eliminates most vocabulary
- `ngram_range=(1, 2):` ⇨ 283 k vocabulary
- `sublinear_tf=True`

Handling Data

tf-idf features

- `min_df=5:` ⇨ Eliminates most vocabulary
- `ngram_range=(1, 2):` ⇨ 283 k vocabulary
- `sublinear_tf=True`
- `max_df=0.8:` ⇨ option (not taken) for ‘stop word’ removal

Handling Data

tf-idf features

- `min_df=5`: ➪ Eliminates most vocabulary
- `ngram_range=(1, 2)`: ➪ 283 k vocabulary
- `sublinear_tf=True`
- ~~`max_df=0.8`~~: ➪ option ~~(not taken)~~ for ‘stop word’ removal

Modeling

See `notebook/DocumentClassificationTest.ipynb` in my repo for details

Modeling

See `notebook/DocumentClassificationTest.ipynb` in [my repo] for details

`f1_scorer`: ⇨ optimize for f_1 during grid search

Modeling

See `notebook/DocumentClassificationTest.ipynb` in my repo for details

`f1_scorer`: ⇨ optimize for f_1 during grid search

- used `average=”weighted”`, but `average=”macro”` would have yielded better results on small classes

Modeling

See `notebook/DocumentClassificationTest.ipynb` in [my repo] for details

`f1_scorer`: ⇨ optimize for f_1 during grid search

- used `average=”weighted”`, but `average=”macro”` would have yielded better results on small classes

Complement Naive Bayes

- default settings for baseline

Modeling

See `notebook/DocumentClassificationTest.ipynb` in [my repo] for details

`f1_scorer`: ⇨ optimize for f_1 during grid search

- used `average="weighted"`, but `average="macro"` would have yielded better results on small classes

Complement Naive Bayes

- default settings for baseline
- followed by grid search

Modeling

See `notebook/DocumentClassificationTest.ipynb` in [my repo] for details

`f1_scorer`: ⇨ optimize for f_1 during grid search

- used `average="weighted"`, but `average="macro"` would have yielded better results on small classes

Complement Naive Bayes

- default settings for baseline
- followed by grid search
- best with `alpha=0.0139 and norm=False` yielded substantial improvements

    ⇨ model *3 × larger*

Modeling

Random Forest

- grid search: top two models had identical f_1 scores

Modeling

Random Forest

- grid search: top two models had identical f_1 scores
  - first had maximum depth of 350, second 250

Modeling

Random Forest

- grid search: top two models had identical f_1 scores
  - first had maximum depth of 350, second 250.
  - The smaller “best” model sizes much smaller at 273 M

Modeling

Random Forest

- grid search: top two models had identical f_1 scores
  - first had maximum depth of 350, second 250.
  - The smaller “best” model sizes much smaller at 273 M
- grid search for RF slow

Modeling

Random Forest

- grid search: top two models had identical f_1 scores
  - first had maximum depth of 350, second 250.
  - The smaller "best" model sizes much smaller at 273 M
- grid search for RF slow
- this 2nd version is deployed on AWS, accessible from my UI

Modeling

Random Forest

- grid search: top two models had identical f_1 scores
  - first had maximum depth of 350, second 250.
  - The smaller “best” model sizes much smaller at 273 M
- grid search for RF slow
- this 2nd version is deployed on AWS, accessible from my UI

GradientBoostingClassifier

- scikit-learn’s own algo slowest to train

Modeling

Random Forest

- grid search: top two models had identical f_1 scores
  - first had maximum depth of 350, second 250.
  - The smaller “best” model sizes much smaller at 273 M
- grid search for RF slow
- this 2nd version is deployed on AWS, accessible from my UI

GradientBoostingClassifier

- scikit-learn’s own algo slowest to train ⇨ opted not to grid search

Modeling

Random Forest

- grid search: top two models had identical f_1 scores
  - first had maximum depth of 350, second 250.
  - The smaller “best” model sizes much smaller at 273 M
- grid search for RF slow
- this 2nd version is deployed on AWS, accessible from my UI

GradientBoostingClassifier

- scikit-learn’s own algo slowest to train ⇨ opted not to grid search

XGBoost

- much faster training than for the GradientBoostingClassifier (explained later)

Modeling

Random Forest

- grid search: top two models had identical f_1 scores
  - first had maximum depth of 350, second 250.
  - The smaller “best” model sizes much smaller at 273 M
- grid search for RF slow
- this 2nd version is deployed on AWS, accessible from my UI

GradientBoostingClassifier

- scikit-learn’s own algo slowest to train ⇨ opted not to grid search

XGBoost

- much faster training than for the GradientBoostingClassifier (explained later)
- equally excellent results

Modeling

Random Forest

- grid search: top two models had identical f_1 scores
  - first had maximum depth of 350, second 250.
  - The smaller “best” model sizes much smaller at 273 M
- grid search for RF slow
- this 2nd version is deployed on AWS, accessible from my UI

GradientBoostingClassifier

- scikit-learn’s own algo slowest to train ⇨ opted not to grid search

XGBoost

- much faster training than for the GradientBoostingClassifier (explained later)
- equally excellent results
- optimized model ⇨ “best” overall

Modeling

Random Forest

- grid search: top two models had identical f_1 scores
  - first had maximum depth of 350, second 250.
  - The smaller “best” model sizes much smaller at 273 M
- grid search for RF slow
- this 2nd version is deployed on AWS, accessible from my UI

GradientBoostingClassifier

- scikit-learn’s own algo slowest to train ⇨ opted not to grid search

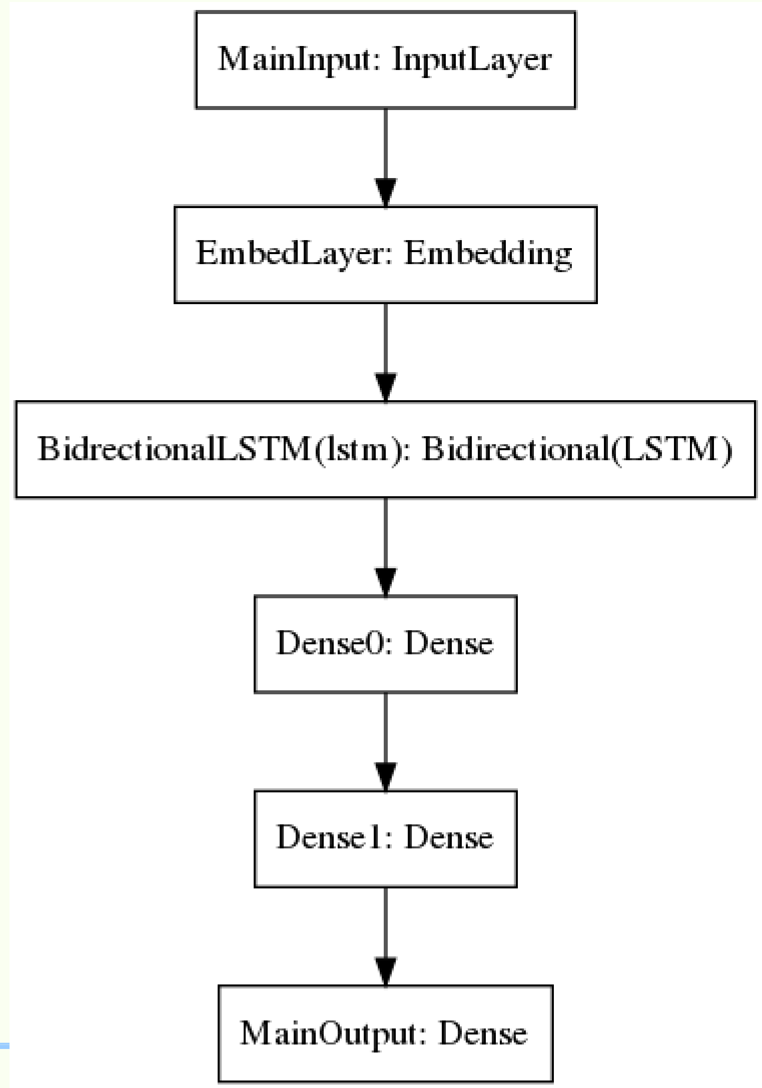
XGBoost

- much faster training than for the GradientBoostingClassifier (explained later)
- equally excellent results
- optimized model ⇨ “best” overall (but we can’t claim this strictly due to uncertainty)

Modeling

LSTM

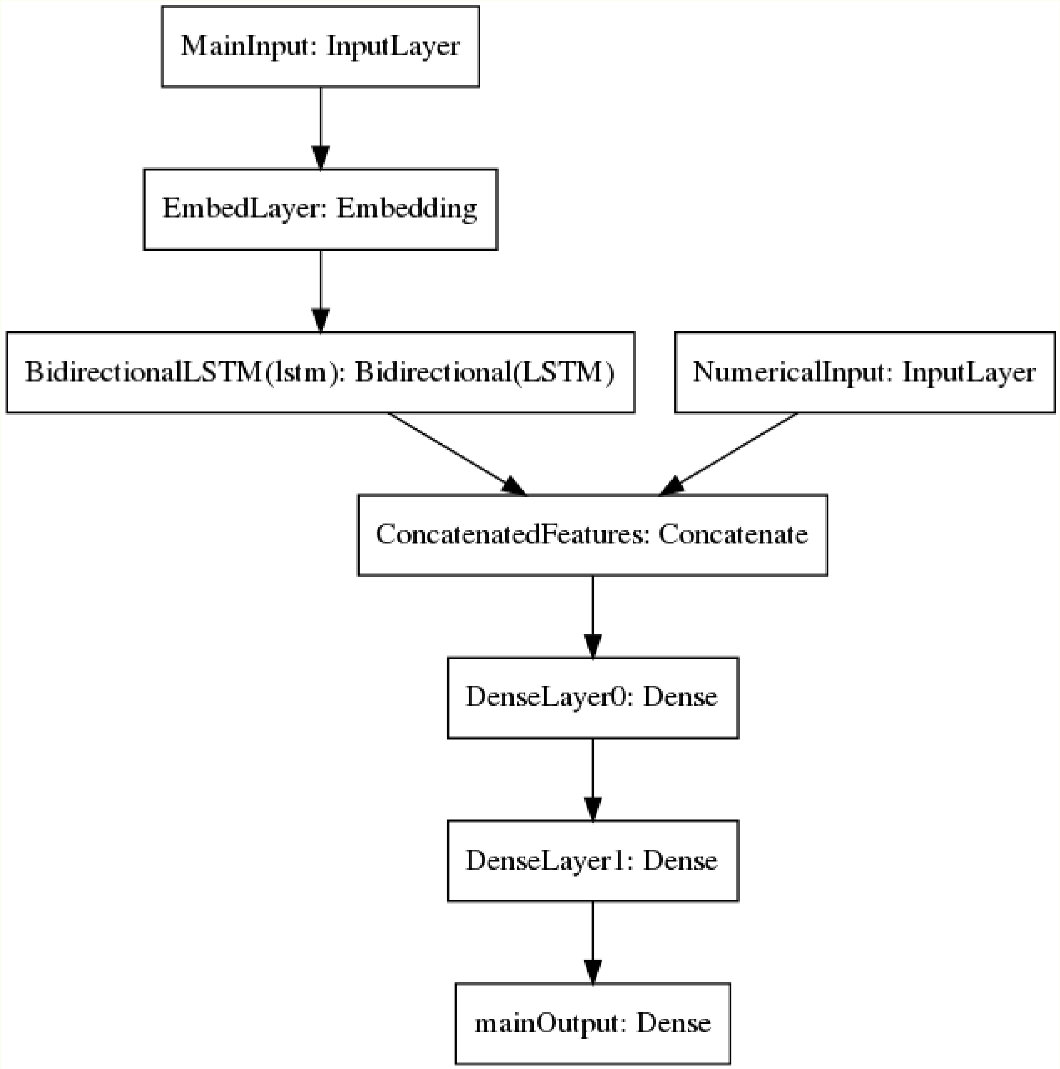
- Sequence lengths {64, 128, 256, 512)
- `maxVocabCt` {100000, 200000}
- regularizaton: `dropout=0.15`, `recurrent_dropout=0.40`
- Shorter sequences better



Modeling

LSTM + `docLength`

- auxiliary feature for document length
- sequence lengths {256, 384)
- `maxVocabCt` {100000, 200000}
- regularizaton: `dropout=0.15`, `recurrent_dropout=0.40`
- shorter sequences better



Modeling

LSTM + `docLength`

- auxiliary feature for document length
- sequence lengths {256, 384)
- `maxVocabCt` {100000, 200000}
- regularizaton:
  `dropout=0.15`,
  `recurrent_dropout=0.40`
- shorter sequences better

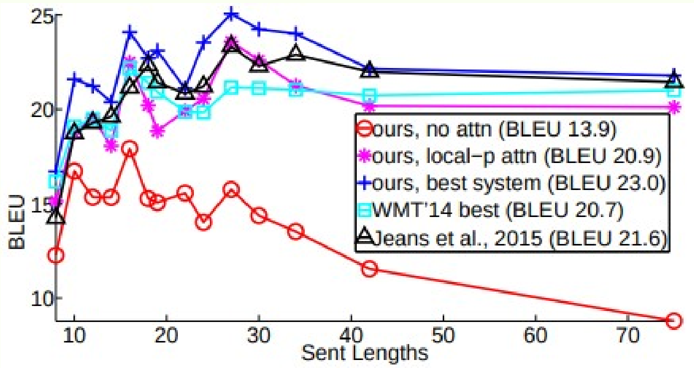
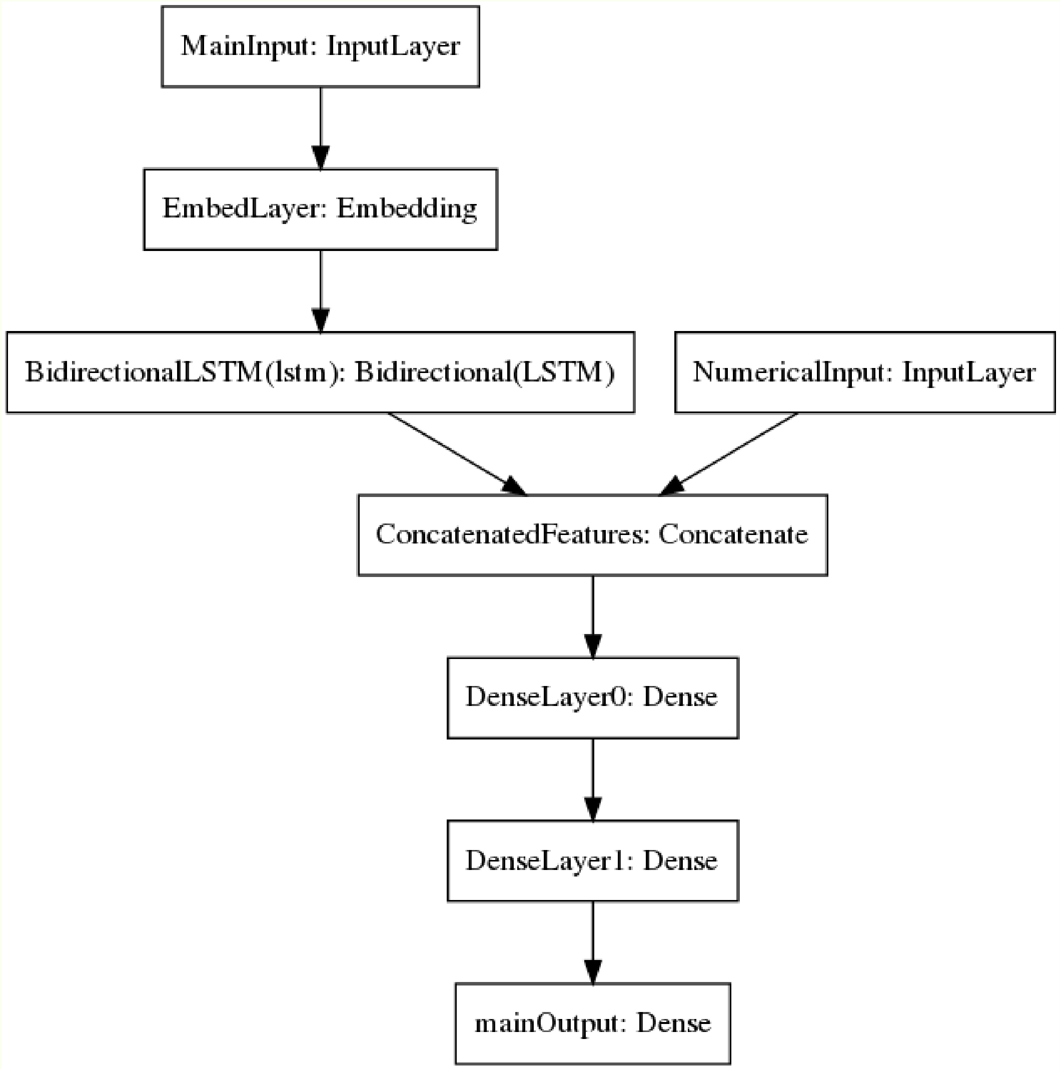


Figure 6: **Length Analysis** – translation qualities of different systems as sentences become longer.

Luong, M-T, et al [arXiv 1508.04025v5, 2015]



Model Results

Model	Macro Averaged			Weighted Average			Model size (MB)
	precision	recall	f_1	precision	recall	f_1	
Naive Bayes default (baseline)	0.75	0.50	0.53	0.79	0.78	0.76	63
Naive Bayes best	0.80	0.58	0.62	0.81	0.81	0.80	272
Random Forest default	0.80	0.65	0.70	0.84	0.85	0.84	451
Random Forest best	0.80	0.75	0.77	0.87	0.87	0.87	273
Gradient Boosting default	0.81	0.64	0.69	0.81	0.81	0.80	1.2
XGBoost default	0.79	0.65	0.70	0.82	0.82	0.82	5.4
XGBoost best	0.82	0.73	0.76	0.87	0.87	0.87	21
Bidirectional LSTM	0.54	0.58	0.54	0.76	0.70	0.72	53
Bidirectional LSTM w/ `docLength`	0.50	0.61	0.52	0.78	0.69	0.73	202

- reminder: macro averaged ⇨ straight average of scores for each class
  weighted average ⇨ average of all class scores weighted by support

Model Results

Model	Macro Averaged			Weighted Average			Model size (MB)
	precision	recall	f_1	precision	recall	f_1	
Naive Bayes default (baseline)	`0.75`	`0.50`	`0.53`	0.79	0.78	0.76	63
Naive Bayes best	0.80	0.58	0.62	0.81	0.81	0.80	272
Random Forest default	0.80	0.65	0.70	0.84	0.85	0.84	451
Random Forest best	0.80	0.75	0.77	0.87	0.87	0.87	273
Gradient Boosting default	0.81	0.64	0.69	0.81	0.81	0.80	1.2
XGBoost default	0.79	0.65	0.70	0.82	0.82	0.82	5.4
XGBoost best	0.82	0.73	0.76	0.87	0.87	0.87	21
Bidirectional LSTM	0.54	0.58	0.54	0.76	0.70	0.72	53
Bidirectional LSTM w/ `docLength`	0.50	0.61	0.52	0.78	0.69	0.73	202

- reminder: macro averaged ➪ straight average of scores for each class
  weighted average ➪ average of all class scores weighted by support
- if need good scores for smaller classes, focus on macro averages

Model Results

Model	Macro Averaged			Weighted Average			Model size (MB)
	precision	recall	f_1	precision	recall	f_1	
Naive Bayes default (baseline)	`0.75`	`0.50`	`0.53`	0.79	0.78	0.76	63
Naive Bayes best	0.80	0.58	0.62	0.81	0.81	0.80	272
Random Forest default	0.80	0.65	0.70	0.84	0.85	0.84	451
Random Forest best	0.80	0.75	0.77	0.87	0.87	0.87	273
Gradient Boosting default	0.81	0.64	0.69	0.81	0.81	0.80	1.2
XGBoost default	0.79	0.65	0.70	0.82	0.82	0.82	5.4
XGBoost best	0.82	0.73	0.76	0.87	0.87	0.87	21
Bidirectional LSTM	0.54	0.58	0.54	0.76	0.70	0.72	53
Bidirectional LSTM w/ `docLength`	0.50	0.61	0.52	0.78	0.69	0.73	202

- reminder: macro averaged ➪ straight average of scores for each class
  weighted average ➪ average of all class scores weighted by support
- if need good scores for smaller classes, focus on macro averages
- if overall results most important, focus on weighted averages

Model Results

Model	Macro Averaged			Weighted Average			Model size (MB)
	precision	recall	f_1	precision	recall	f_1	
Naive Bayes default (baseline)	0.75	0.50	0.53	0.79	0.78	0.76	63
Naive Bayes best	0.80	0.58	0.62	0.81	0.81	0.80	272
Random Forest default	0.80	0.65	0.70	0.84	0.85	0.84	451
Random Forest best	0.80	0.75	0.77	0.87	0.87	0.87	273
Gradient Boosting default	0.81	0.64	0.69	0.81	0.81	0.80	1.2
XGBoost default	0.79	0.65	0.70	0.82	0.82	0.82	5.4
XGBoost best	0.82	0.73	0.76	0.87	0.87	0.87	21
Bidirectional LSTM	0.54	0.58	0.54	0.76	0.70	0.72	53
Bidirectional LSTM w/ `docLength`	0.50	0.61	0.52	0.78	0.69	0.73	202

- reminder: macro averaged ⇨ straight average of scores for each class
  weighted average ⇨ average of all class scores weighted by support
- if need good scores for smaller classes, focus on macro averages
- if overall results most important, focus on weighted averages

- Random Forest and XGBoost “identical” good results

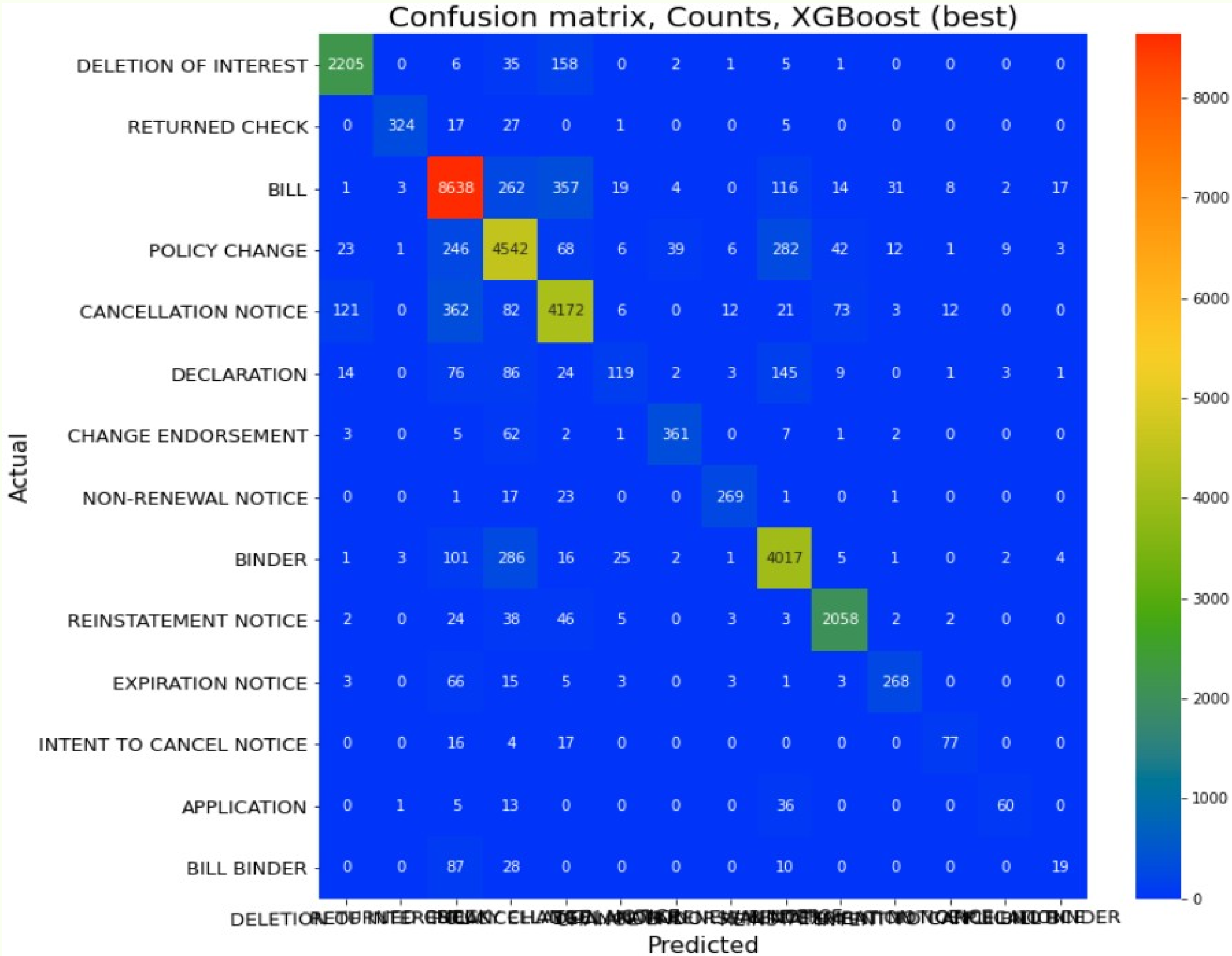
Model Results

Model	Macro Averaged			Weighted Average			Model size (MB)
	precision	recall	f_1	precision	recall	f_1	
Naive Bayes default (baseline)	0.75	0.50	0.53	0.79	0.78	0.76	63
Naive Bayes best	0.80	0.58	0.62	0.81	0.81	0.80	272
Random Forest default	0.80	0.65	0.70	0.84	0.85	0.84	451
Random Forest best	0.80	0.75	$\boxed{0.77}$	0.87	0.87	$\boxed{0.87}$	273
Gradient Boosting default	0.81	0.64	0.69	0.81	0.81	0.80	1.2
XGBoost default	0.79	0.65	0.70	0.82	0.82	0.82	5.4
XGBoost best	0.82	0.73	$\boxed{0.76}$	0.87	0.87	$\boxed{0.87}$	21
Bidirectional LSTM	0.54	0.58	0.54	0.76	0.70	0.72	53
Bidirectional LSTM	0.50	0.61	0.52	0.78	0.69	0.73	202

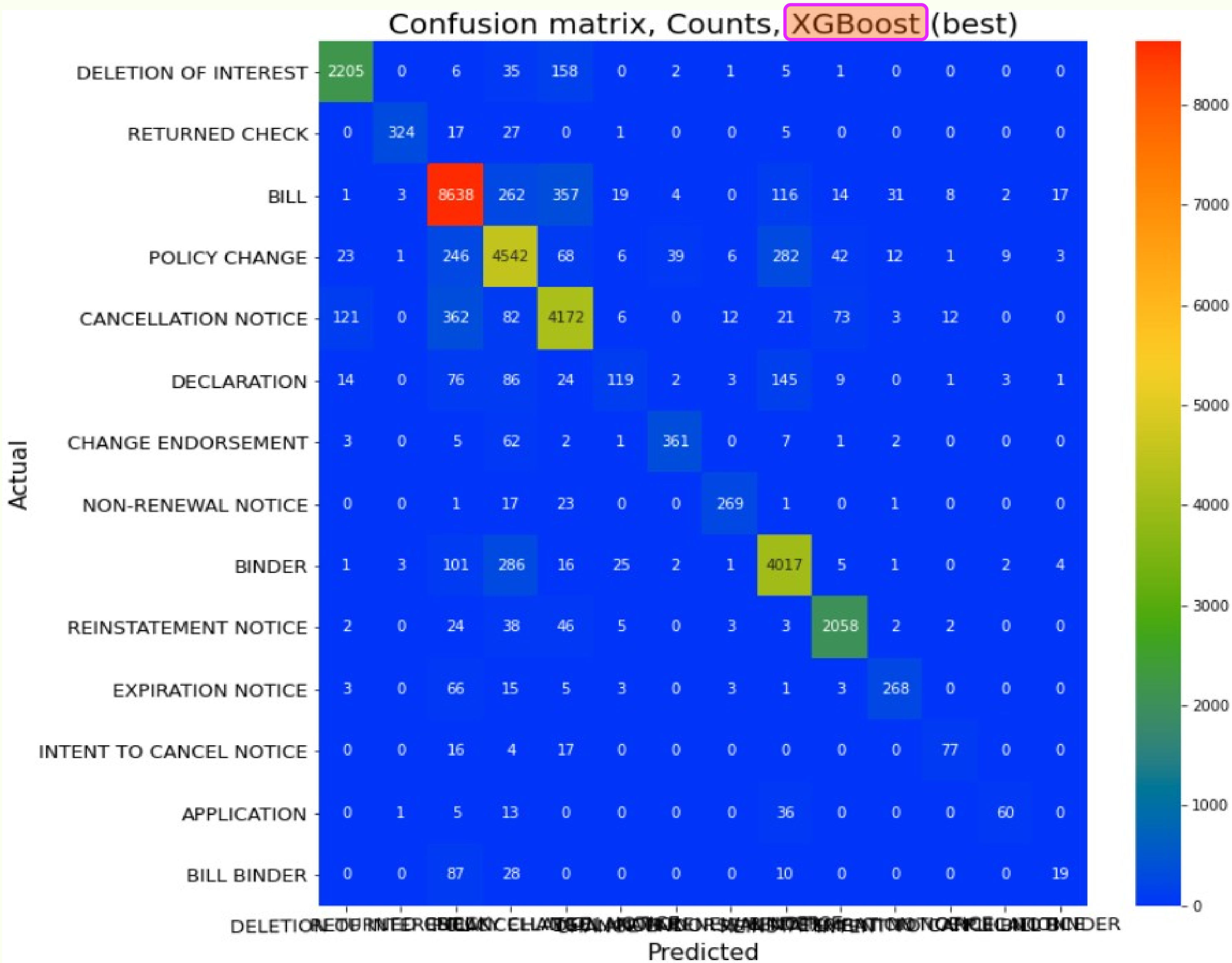
- reminder: macro averaged ⇨ straight average of scores for each class
  weighted average ⇨ average of all class scores weighted by support
- if need good scores for smaller classes, focus on macro averages
- if overall results most important, focus on weighted averages

- Random Forest and XGBoost “identical” good results
- Caution: errors in small classes $\mathcal{O}$(10%) ⇨ impact macro averages most

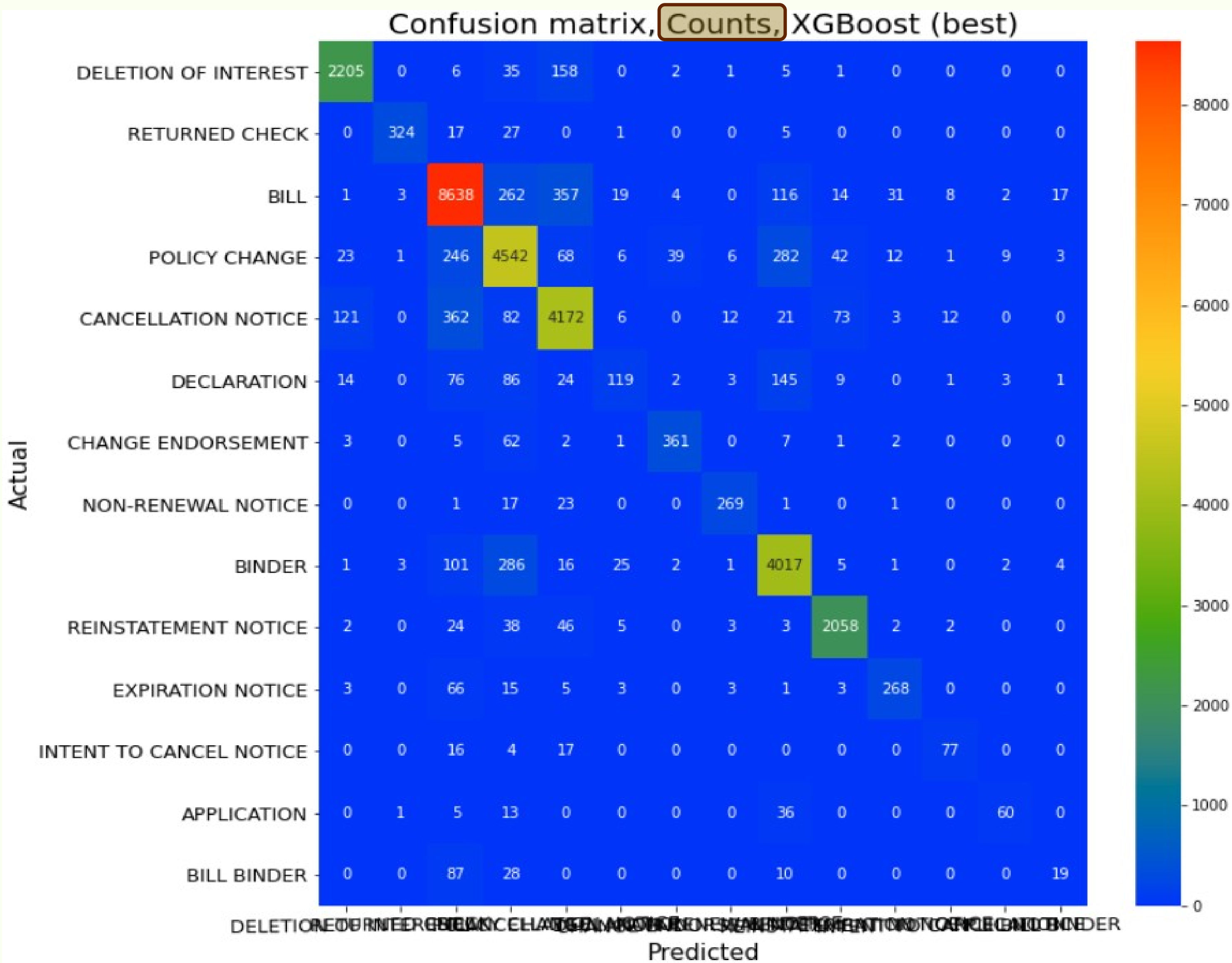
Model Results



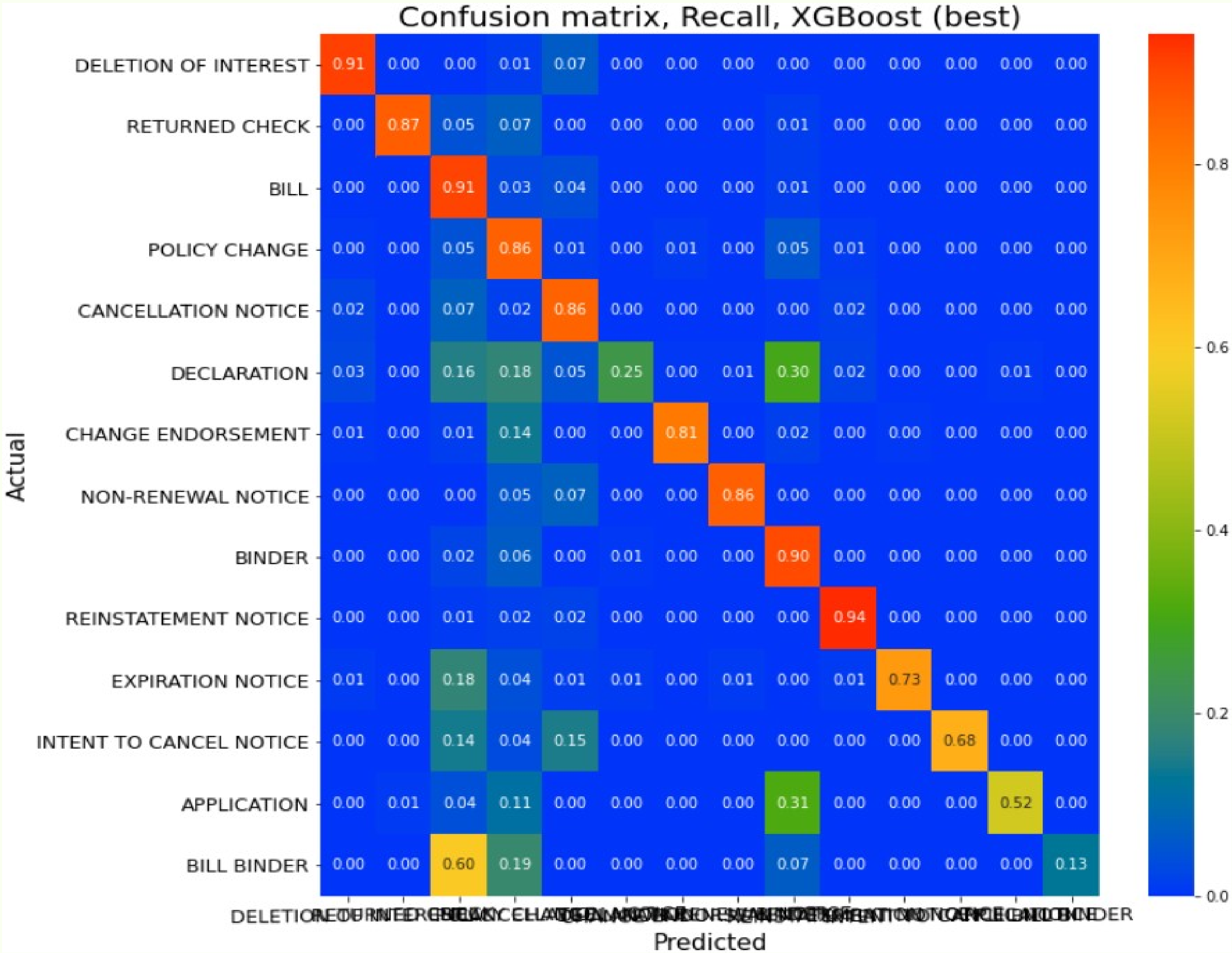
Model Results



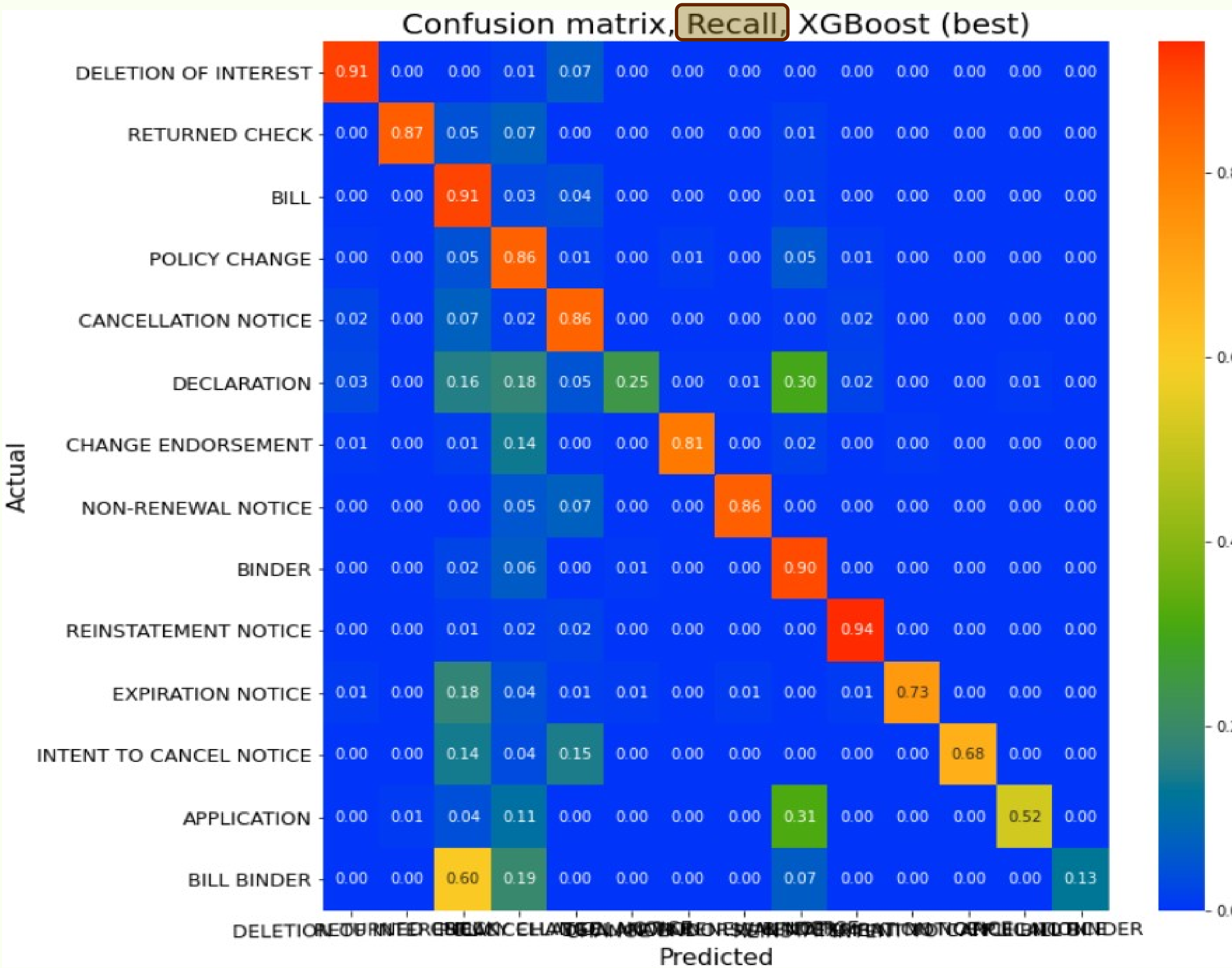
Model Results



Model Results

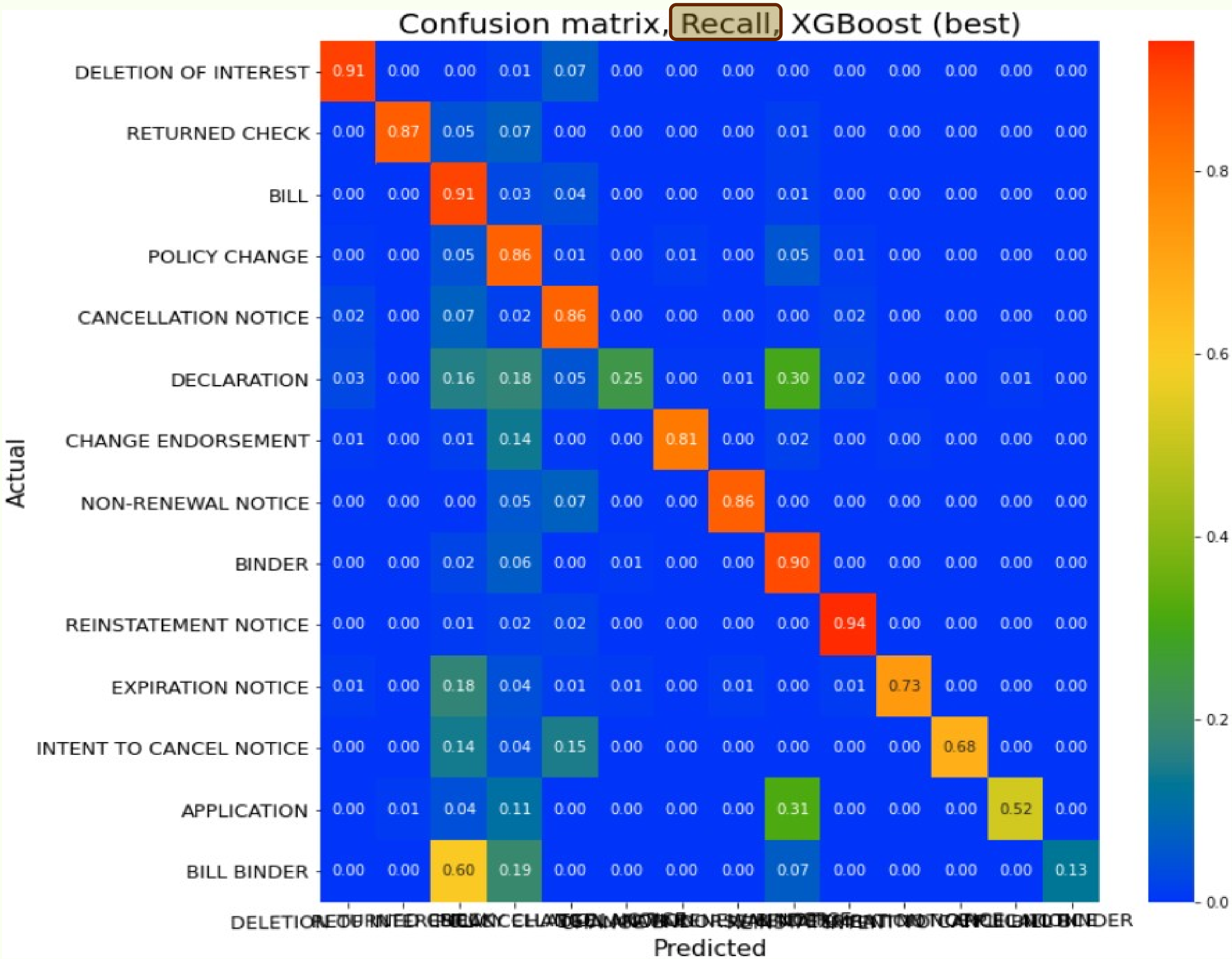


Model Results



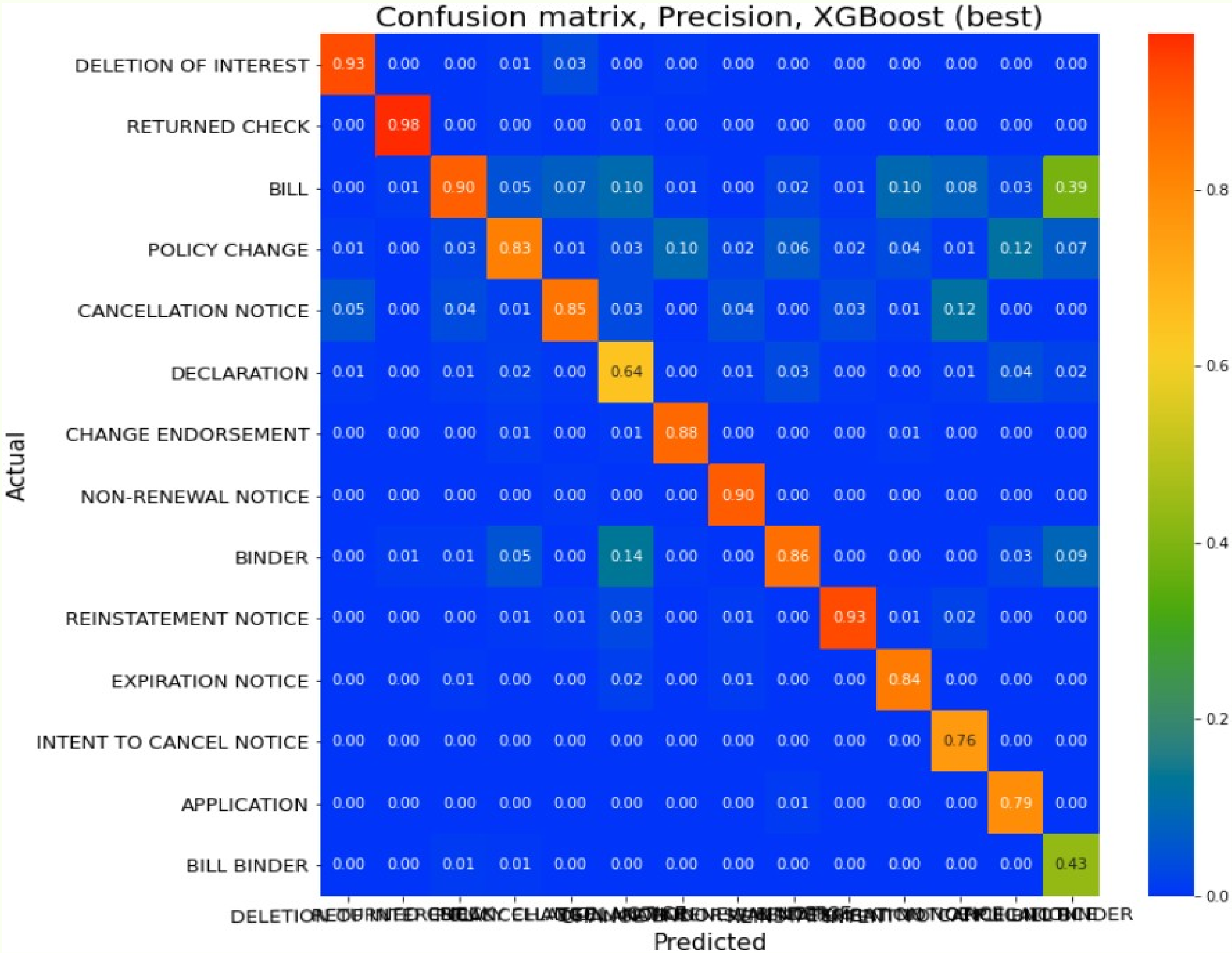
$$R = \frac{TP}{TP + FN}$$

Model Results

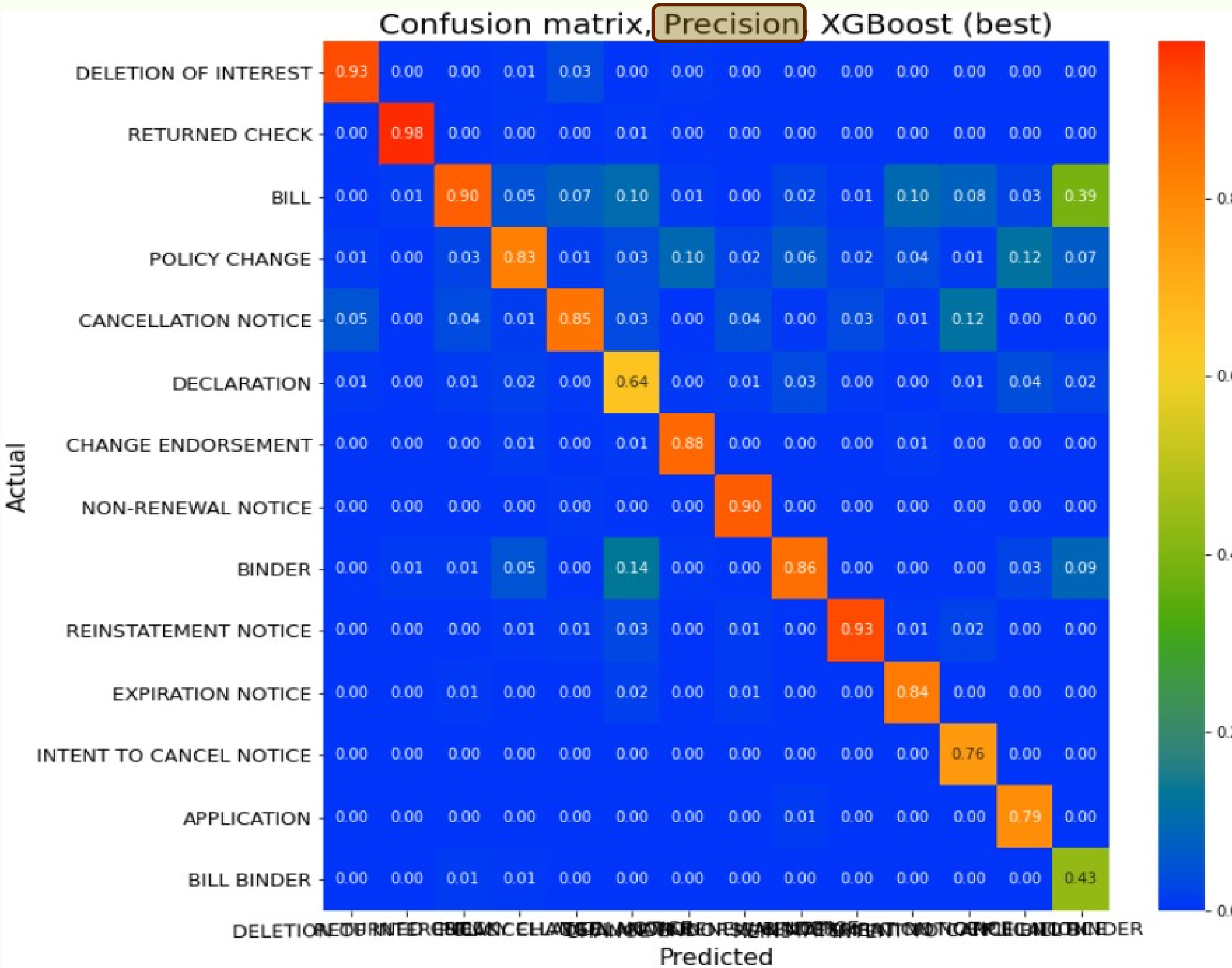


$$R = \frac{TP}{TP + FN}$$

Model Results

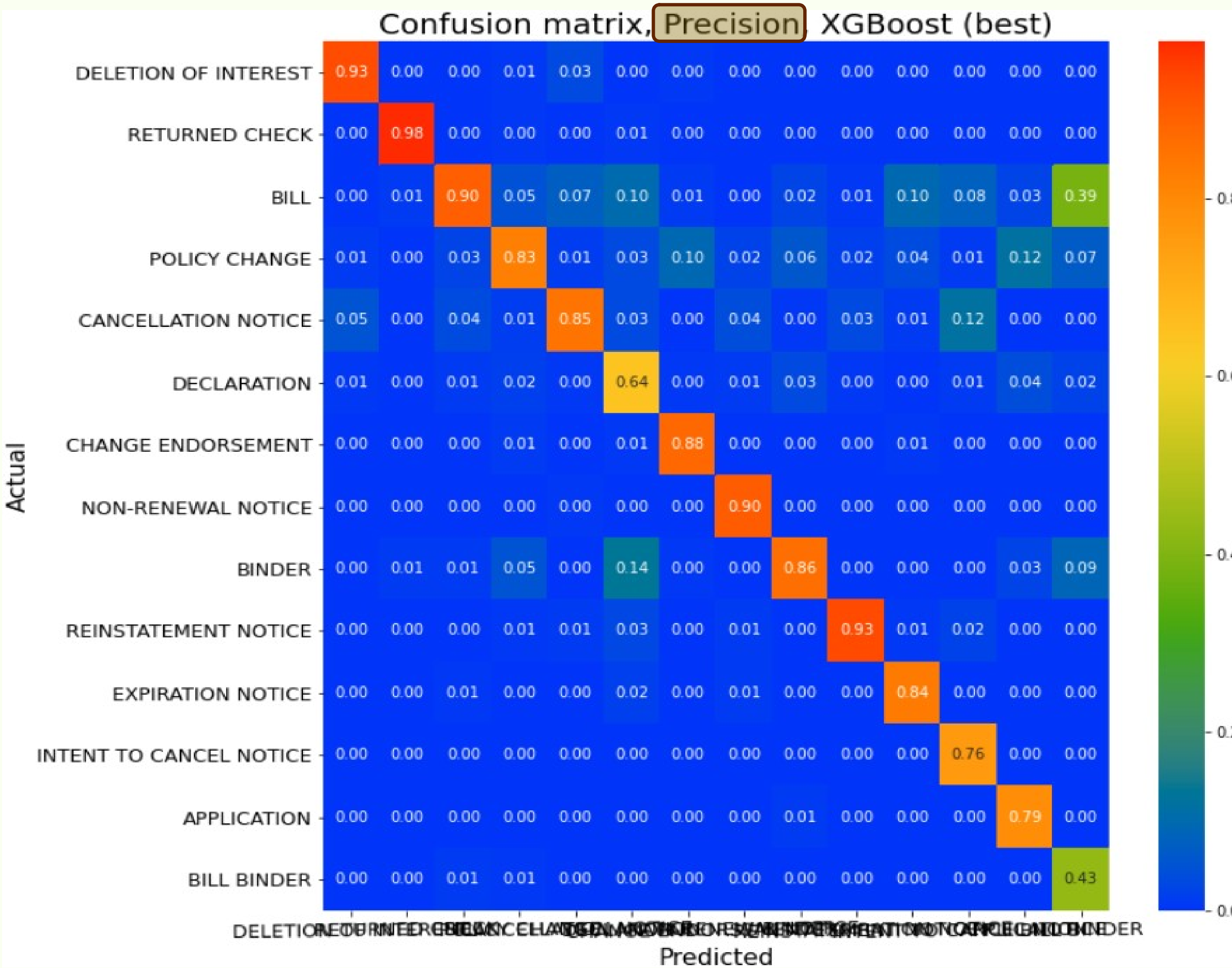


Model Results



$$P = \frac{TP}{TP + FP}$$

Model Results



$$P = \frac{TP}{TP + FP}$$

Deployed Solution

General notes

- Find code in github repo

Deployed Solution

General notes

- Find code in [github repo]
- Significant learning curve, both for Docker and deployment of end points

Deployed Solution

General notes

- Find code in [github repo]
- Significant learning curve, both for Docker and deployment of end points
- Learned the hard way why XGBoost training was much faster than GradientBoostingClassifer

Deployed Solution

General notes

- Find code in github repo
- Significant learning curve, both for Docker and deployment of end points
- Learned the hard way why XGBoost training was much faster than GradientBoostingClassifer
  - XGBoost discovered GPUs on local machine when training

Deployed Solution

General notes

- Find code in github repo
- Significant learning curve, both for Docker and deployment of end points
- Learned the hard way why XGBoost training was much faster than GradientBoostingClassifer
  - XGBoost discovered GPUs on local machine when training
  - ⇨ trained model insisted on GPUs for inference

Deployed Solution

General notes

- Find code in github repo
- Significant learning curve, both for Docker and deployment of end points
- Learned the hard way why XGBoost training was much faster than GradientBoostingClassifer
  - XGBoost discovered GPUs on local machine when training
  - ⇨ trained model insisted on GPUs for inference
  - ⇨ redeployed with Naive Bayes and Random Forest

Deployed Solution

General notes

- Find code in github repo
- Significant learning curve, both for Docker and deployment of end points
- Learned the hard way why XGBoost training was much faster than GradientBoostingClassifer
  - XGBoost discovered GPUs on local machine when training
  - ⇨ trained model insisted on GPUs for inference
  - ⇨ redeployed with Naive Bayes and Random Forest

Docker container

- `buildDockerImage.sh` for local testing

Deployed Solution

General notes

- Find code in github repo
- Significant learning curve, both for Docker and deployment of end points
- Learned the hard way why XGBoost training was much faster than GradientBoostingClassifer
  - XGBoost discovered GPUs on local machine when training
  - ⇨ trained model insisted on GPUs for inference
  - ⇨ redeployed with Naive Bayes and Random Forest

Docker container

- `buildDockerImage.sh` for local testing
- `build_and_deploy.sh` for also deploying to AWS

Deployed Solution

General notes

- Find code in github repo
- Significant learning curve, both for Docker and deployment of end points
- Learned the hard way why XGBoost training was much faster than GradientBoostingClassifer
  - XGBoost discovered GPUs on local machine when training
  - ⇨ trained model insisted on GPUs for inference
  - ⇨ redeployed with Naive Bayes and Random Forest

Docker container

- `buildDockerImage.sh` for local testing
- `build_and_deploy.sh` for also deploying to AWS
- `Ubuntu:latest`, with minimal set of versioned python packages

Deployed Solution

General notes

- Find code in github repo
- Significant learning curve, both for Docker and deployment of end points
- Learned the hard way why XGBoost training was much faster than GradientBoostingClassifer
  - XGBoost discovered GPUs on local machine when training
  - ⇨ trained model insisted on GPUs for inference
  - ⇨ redeployed with Naive Bayes and Random Forest

Docker container

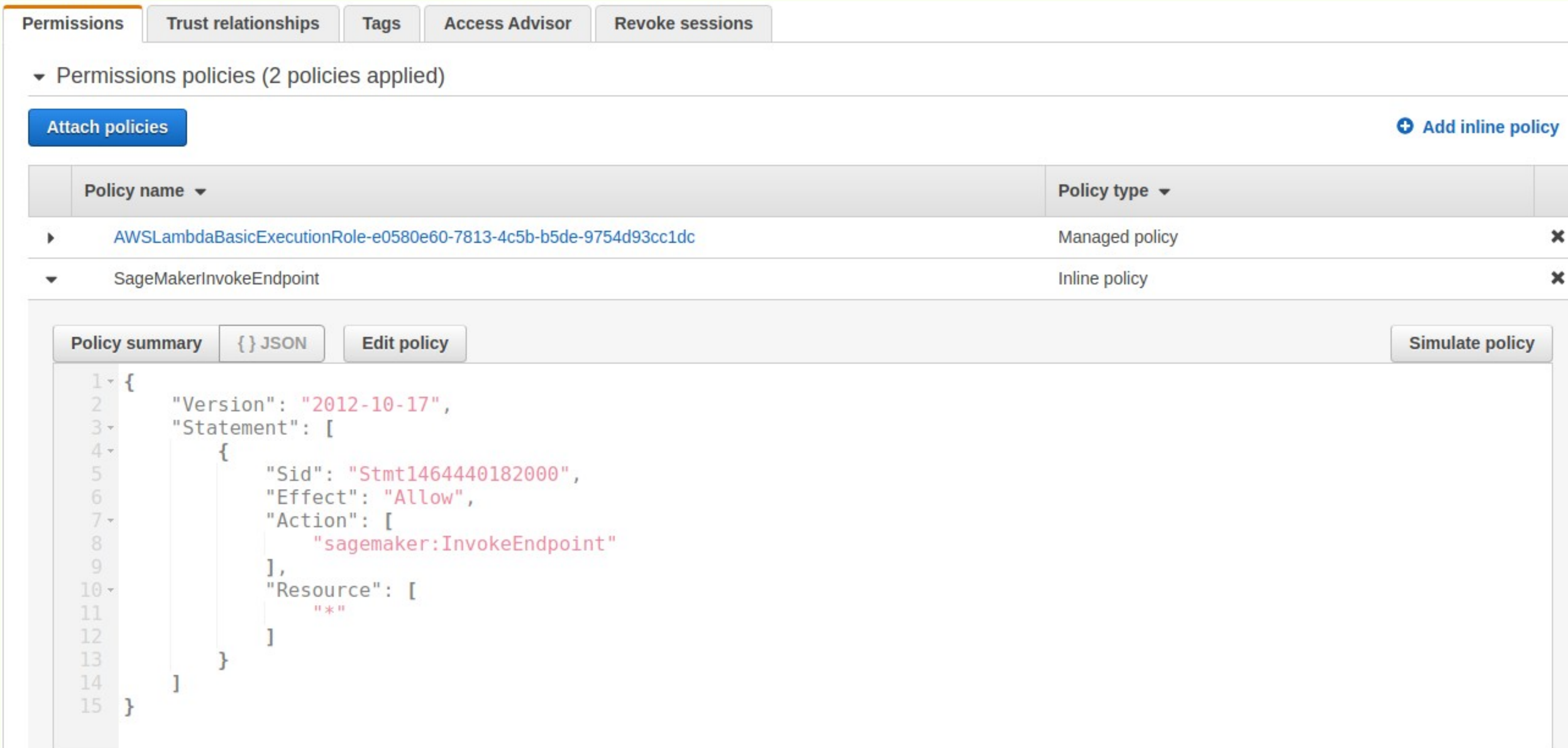
- `buildDockerImage.sh` for local testing
- `build_and_deploy.sh` for also deploying to AWS
- `Ubuntu:latest`, with minimal set of versioned python packages
- image size 912 MB

Deployed Endpoint

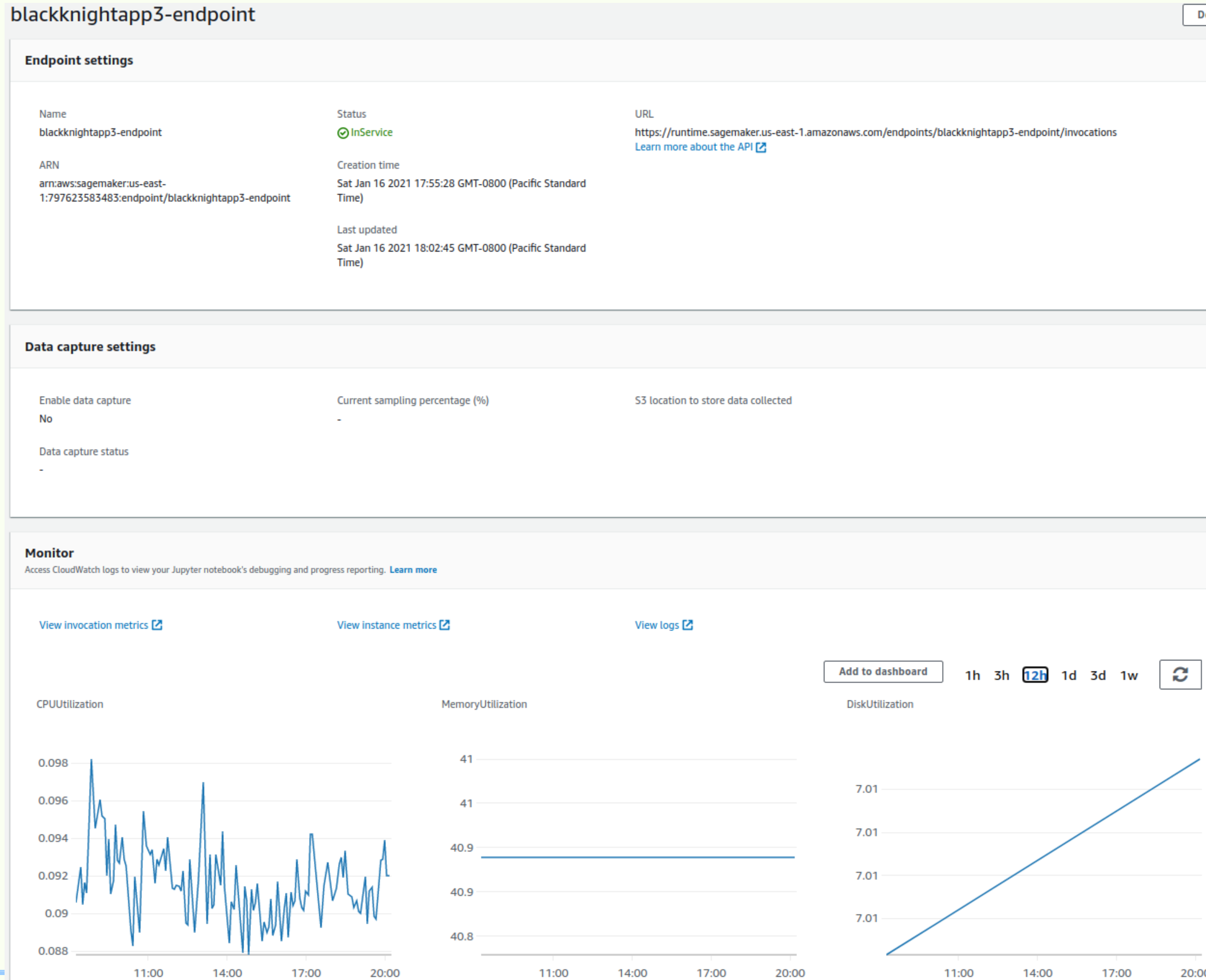
- (default) ml.m4.xlarge EC2 instance

Deployed Endpoint

- (default) ml.m4.xlarge EC2 instance
- endpoint success required extra permissions

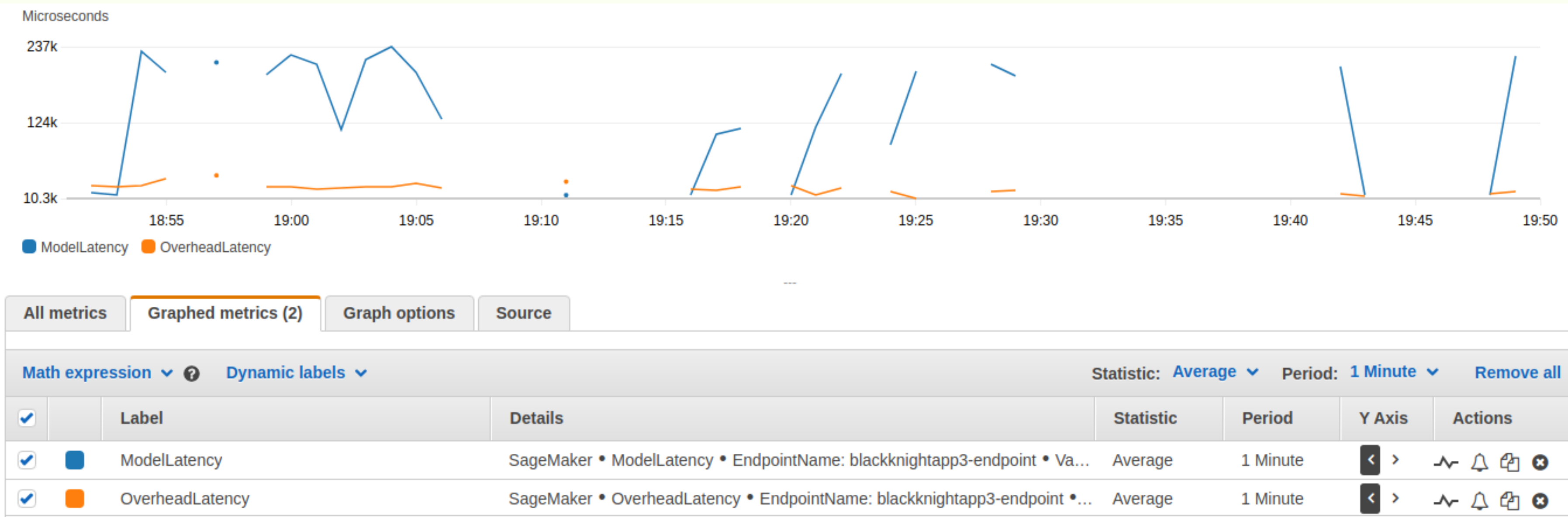


Deployed Endpoint



Deployed Endpoint

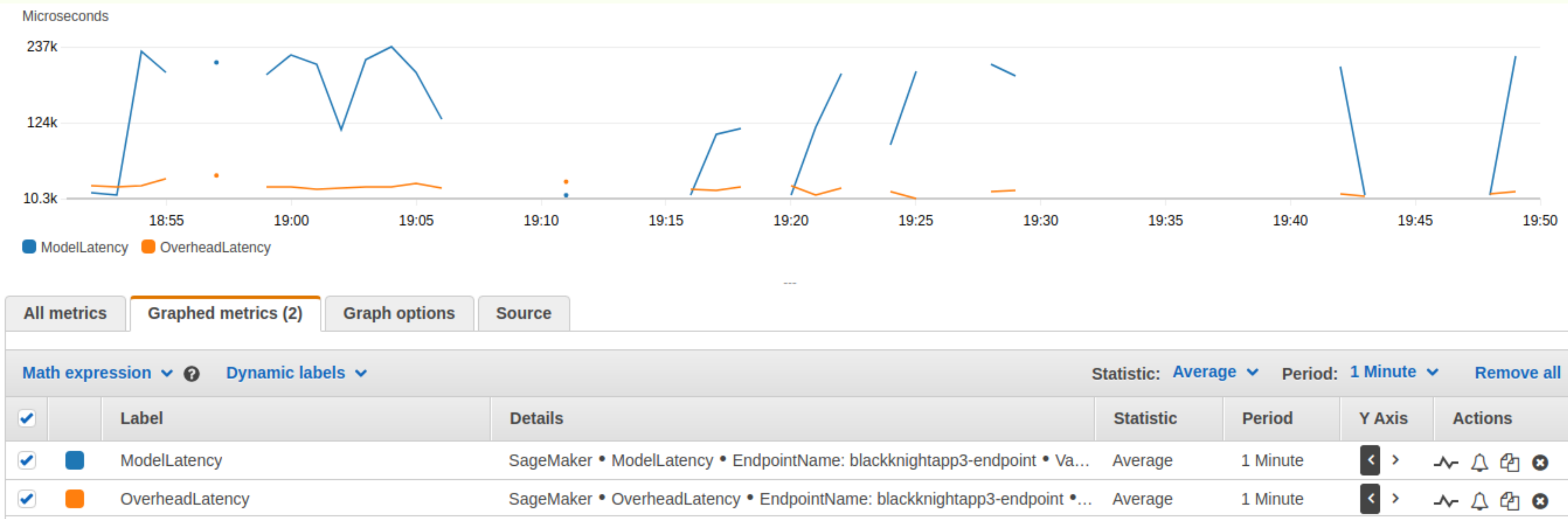
Latencies



- from isolated calls to either Naive Bayes we can see latencies of about 15 ms, while for Random Forest the latencies are about 215 ms

Deployed Endpoint

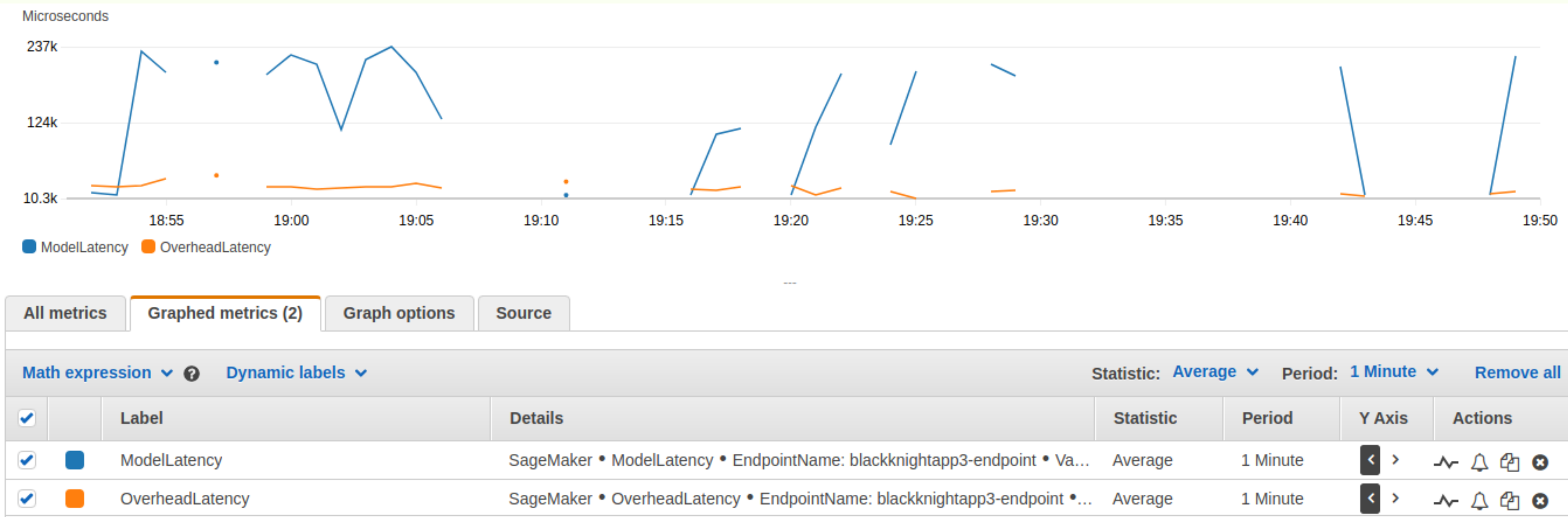
Latencies



- from isolated calls to either Naive Bayes we can see latencies of about 15 ms, while for Random Forest the latencies are about 215 ms
- the Random Forest model has 250 estimators, with maximum depths of 250 – it's a little beast

Deployed Endpoint

Latencies



- from isolated calls to either Naive Bayes we can see latencies of about 15 ms, while for Random Forest the latencies are about 215 ms
- the Random Forest model has 250 estimators, with maximum depths of 250 – it's a little beast
- (the respective model sizes are 63 M and 273 M, and the shared TF-IDF vectorizer is 159 M)

Interactive UI

- Built using streamlit, which is the way to go for speedy development

Interactive UI

- Built using streamlit, which is the way to go for speedy development
  - user dumps new line-separated, hashed documents into text box

Interactive UI

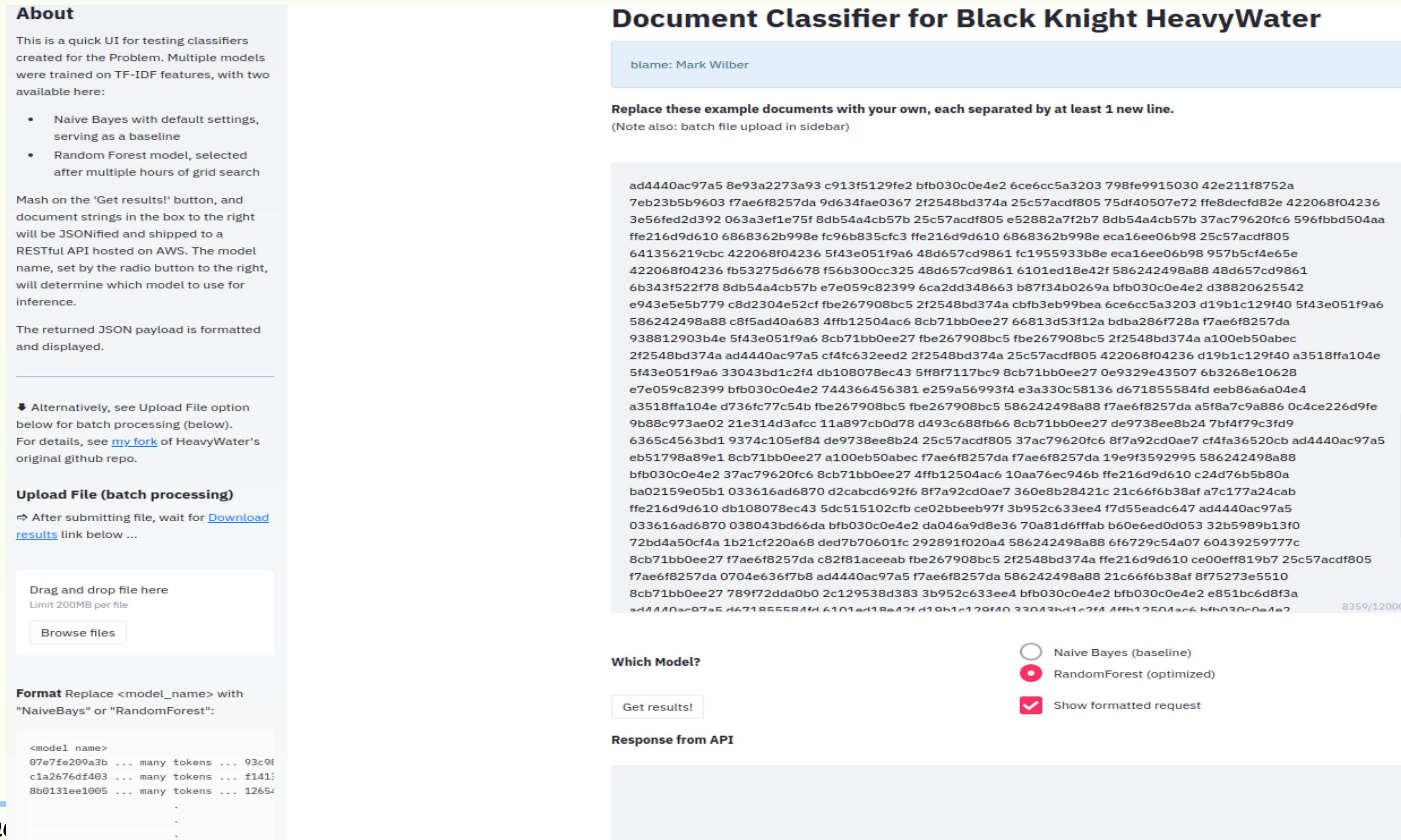
- Built using streamlit, which is the way to go for speedy development
  - user dumps new line-separated, hashed documents into text box
  - JSONified payload is sent to endpoint, which responds with JSONified results

Interactive UI

- Built using streamlit, which is the way to go for speedy development
  - user dumps new line-separated, hashed documents into text box
  - JSONified payload is sent to endpoint, which responds with JSONified results
  - If Random Forest radio button is selected, results also include confidence values

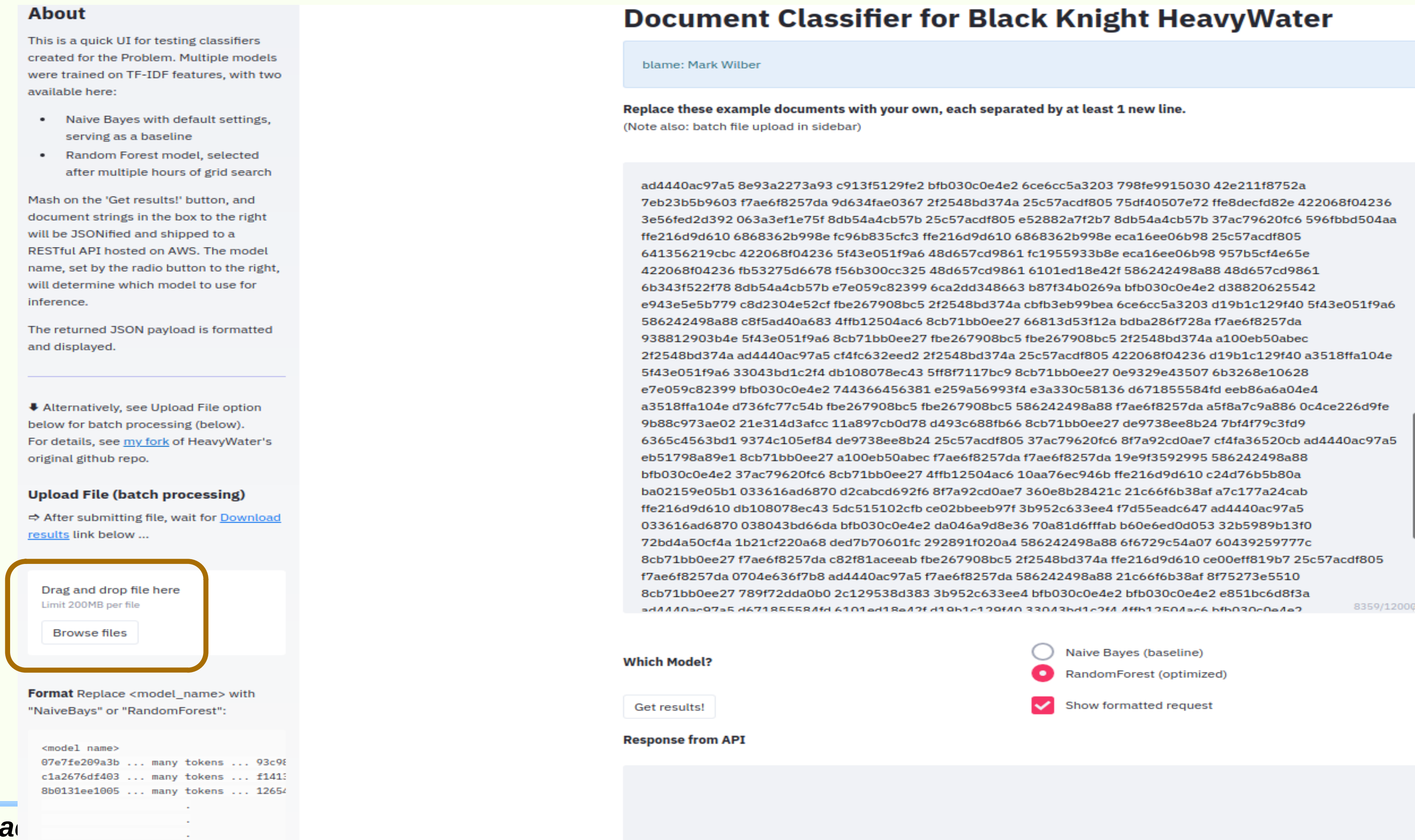
Interactive UI

- Built using [streamlit](), which is the way to go for speedy development
  - user dumps new line-separated, hashed documents into text box
  - JSONified payload is sent to endpoint, which responds with JSONified results
  - If Random Forest radio button is selected, results also include confidence values



Interactive UI

- Built using streamlit, which is the way to go for speedy development
  - user dumps new line-separated, hashed documents into text box
  - JSONified payload is sent to endpoint, which responds with JSONified results
  - If Random Forest radio button is selected, results also include confidence values



A 15-second demo

Next steps?

- LSTM
  - I’ve done this with underwhelming results *(documents are too long)*

Next steps?

- LSTM
    - I’ve done this with underwhelming results *(documents are too long)*
- 1-D convolutional neural network

Next steps?

- LSTM
  - I’ve done this with underwhelming results *(documents are too long)*
- 1-D convolutional neural network
- My preferred solution (in theory!):
  - documents sentenced-tokenized

Next steps?

- LSTM
  - I've done this with underwhelming results *(documents are too long)*
- 1-D convolutional neural network
- My preferred solution (in theory!):
  - documents sentenced-tokenized
  - encode sentences with one of
    - Universal Sentence Encoder

Next steps?

- LSTM
  - [I've done this] with underwhelming results *(documents are too long)*
- 1-D convolutional neural network

- My preferred solution (in theory!):
  - documents sentenced-tokenized
  - encode sentences with one of
    - Universal Sentence Encoder
    - BERT

Next steps?

- LSTM
  - I’ve done this with underwhelming results *(documents are too long)*
- 1-D convolutional neural network
- My preferred solution (in theory!):
  - documents sentenced-tokenized
  - encode sentences with one of
    - Universal Sentence Encoder
    - BERT
    - doc2vec

Next steps?

- LSTM
    - I've done this with underwhelming results *(documents are too long)*
- 1-D convolutional neural network
- My preferred solution (in theory!):
    - documents sentenced-tokenized
    - encode sentences with one of
        - Universal Sentence Encoder
        - BERT
        - doc2vec
        - only doc2vec can be trained from scratch on a modest corpus

Next steps?

- LSTM
  - I’ve done this with underwhelming results *(documents are too long)*
- 1-D convolutional neural network
- My preferred solution (in theory!):
  - documents sentenced-tokenized
  - encode sentences with one of
    - Universal Sentence Encoder
    - BERT
    - doc2vec
    - only doc2vec can be trained from scratch on a modest corpus
  - Sentence embeddings would then be inputs to classifier
    - averaged

Next steps?

- LSTM
  - I've done this with underwhelming results *(documents are too long)*
- 1-D convolutional neural network
- My preferred solution (in theory!):
  - documents sentenced-tokenized
  - encode sentences with one of
    - Universal Sentence Encoder
    - BERT
    - doc2vec
    - only doc2vec can be trained from scratch on a modest corpus
  - Sentence embeddings would then be inputs to classifier
    - averaged
    - sequence-based model (LSTM using sentence embeddings)

That’s all!