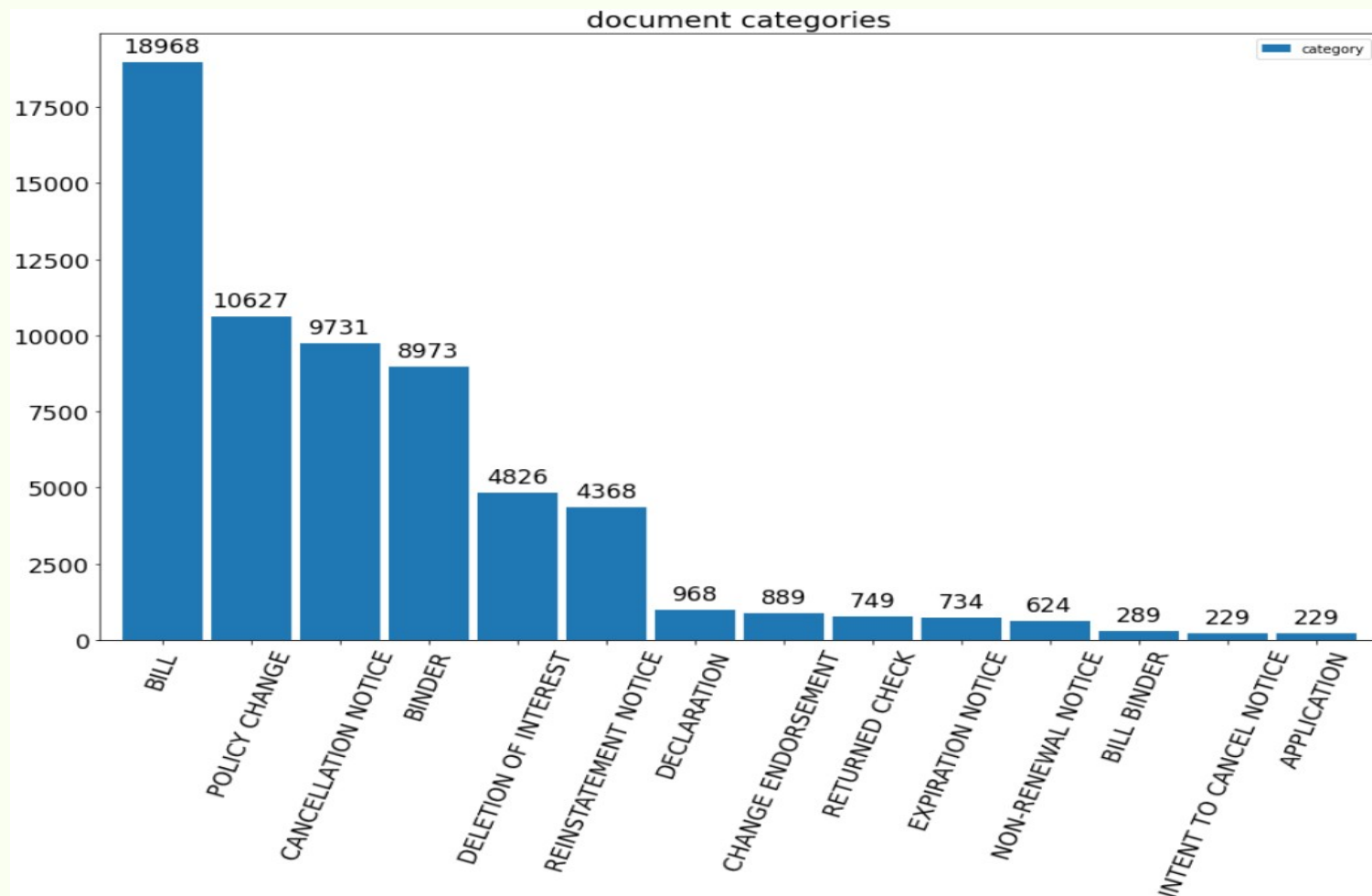


HeavyWater Machine Learning Problem

Solution by Mark Wilber

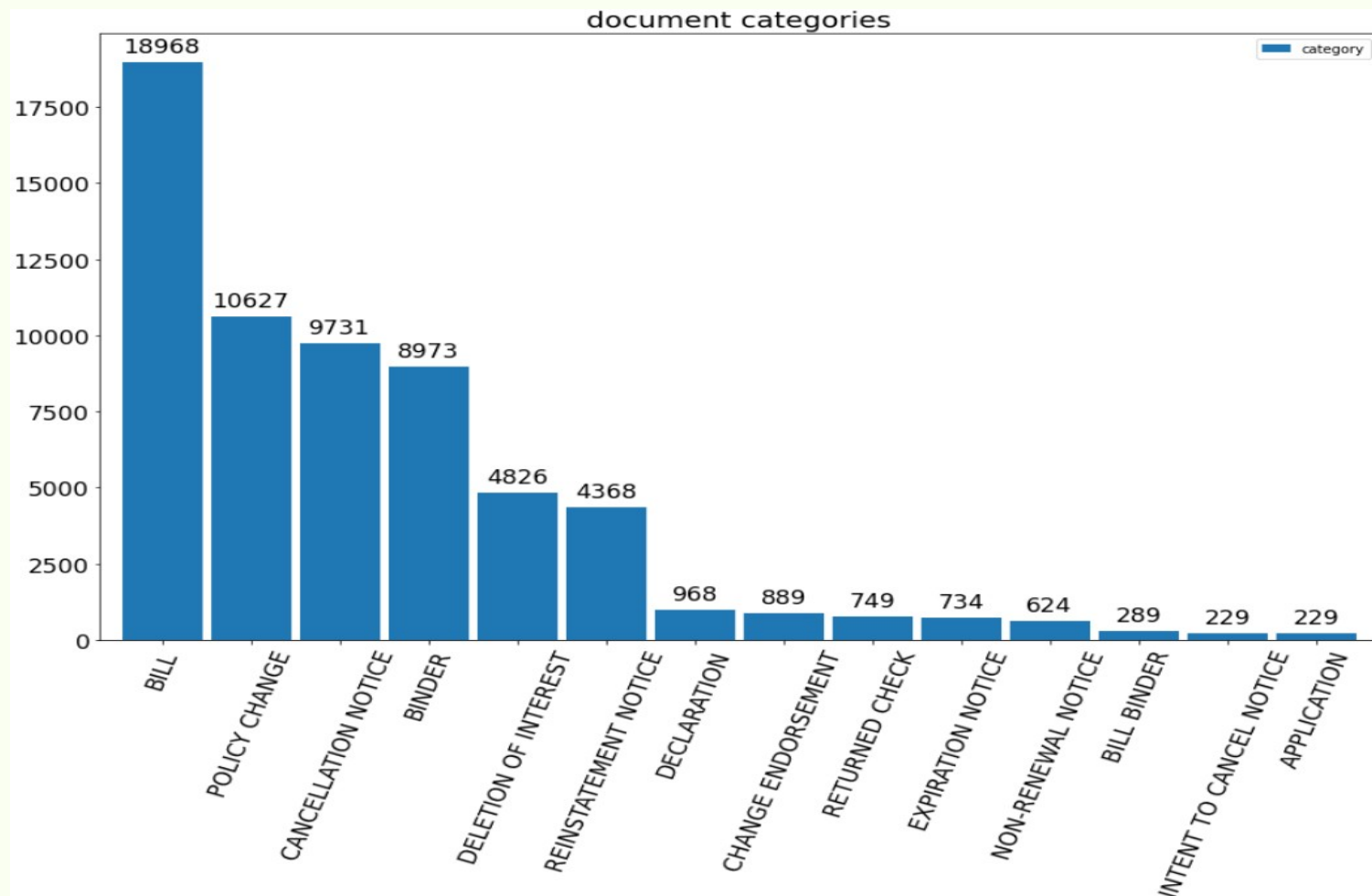
What we are dealing with

- 62 K documents, 14 categories



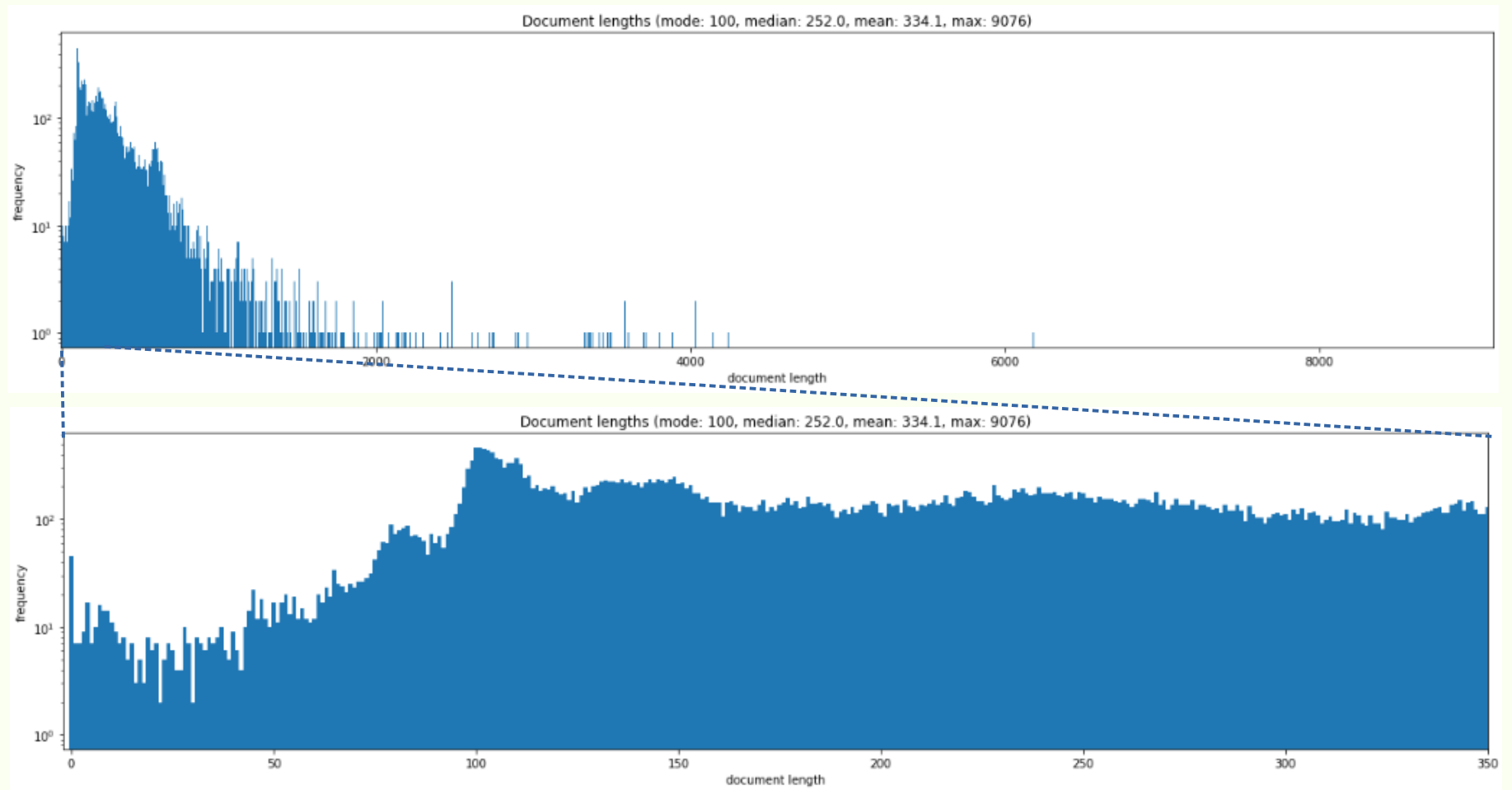
What we are dealing with

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



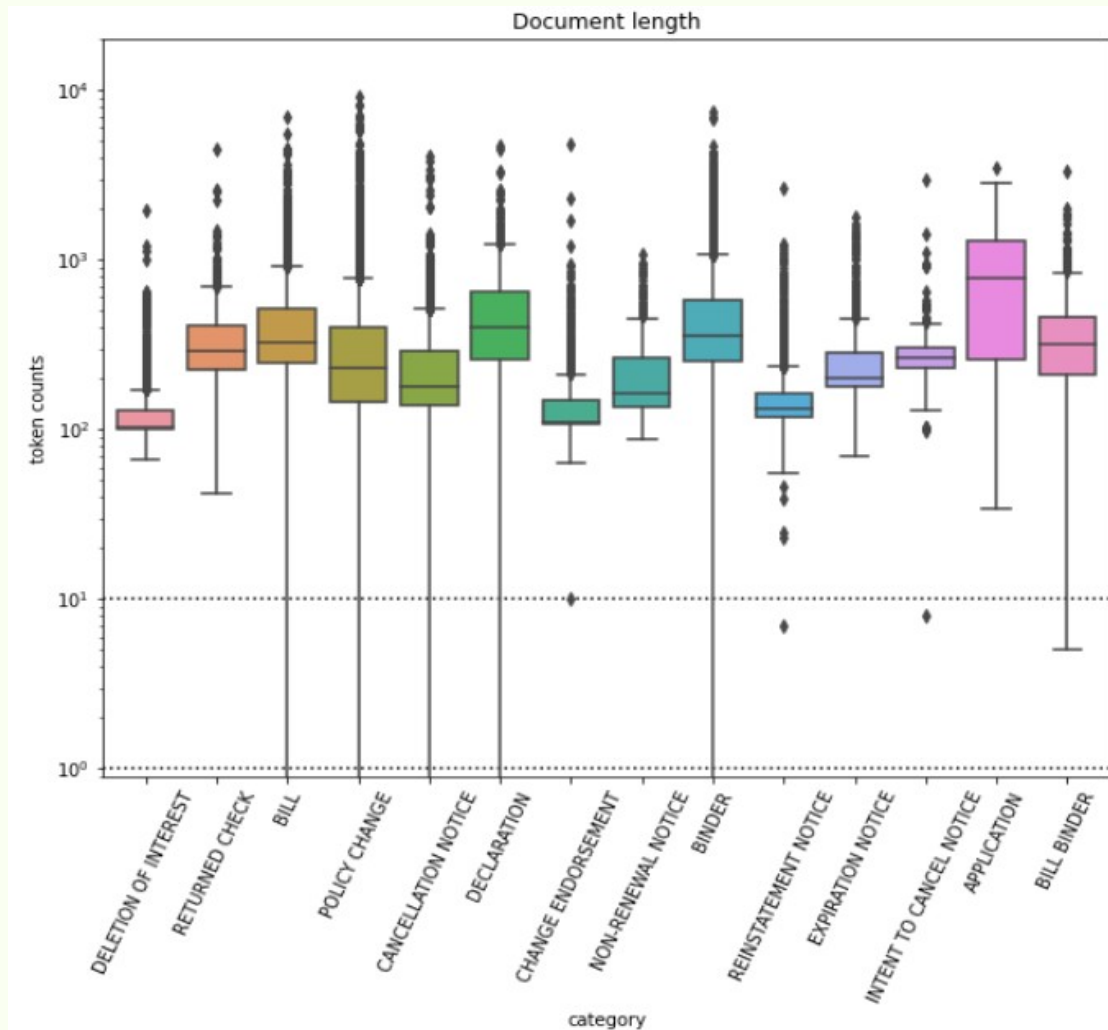
What we are dealing with

- 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	# \geq rank	frac \geq 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	# \geq rank	frac \geq 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	# \geq rank	frac \geq 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

What we are dealing with

- Consider terms occurring with lowest frequencies

tf	rank	# \geq rank	frac \geq 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	# \geq rank	frac \geq 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	# \geq rank	frac \geq 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

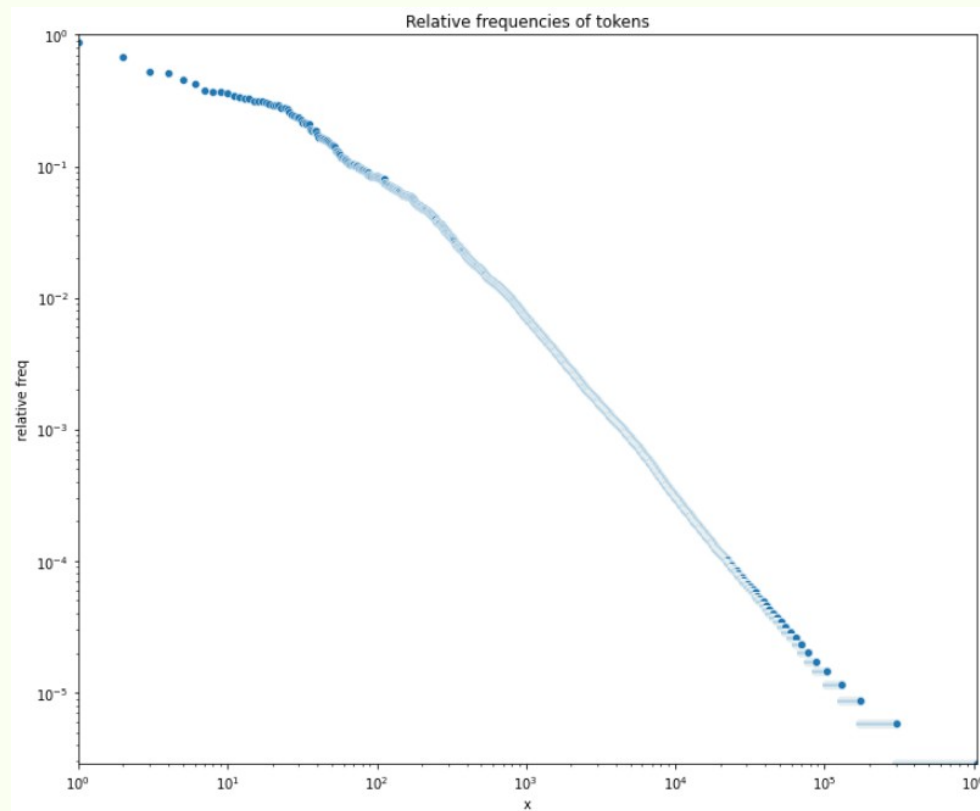
- explanation: most of the tokens are “uninformative” (garbage)
 - 71% of tokens only appear once, 92.6% occur 6 × or fewer
 - A small fraction are names (of humans, businesses), special codes
 - if each numerical value has a token (rather than each digit), that could account for many (but not all), 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

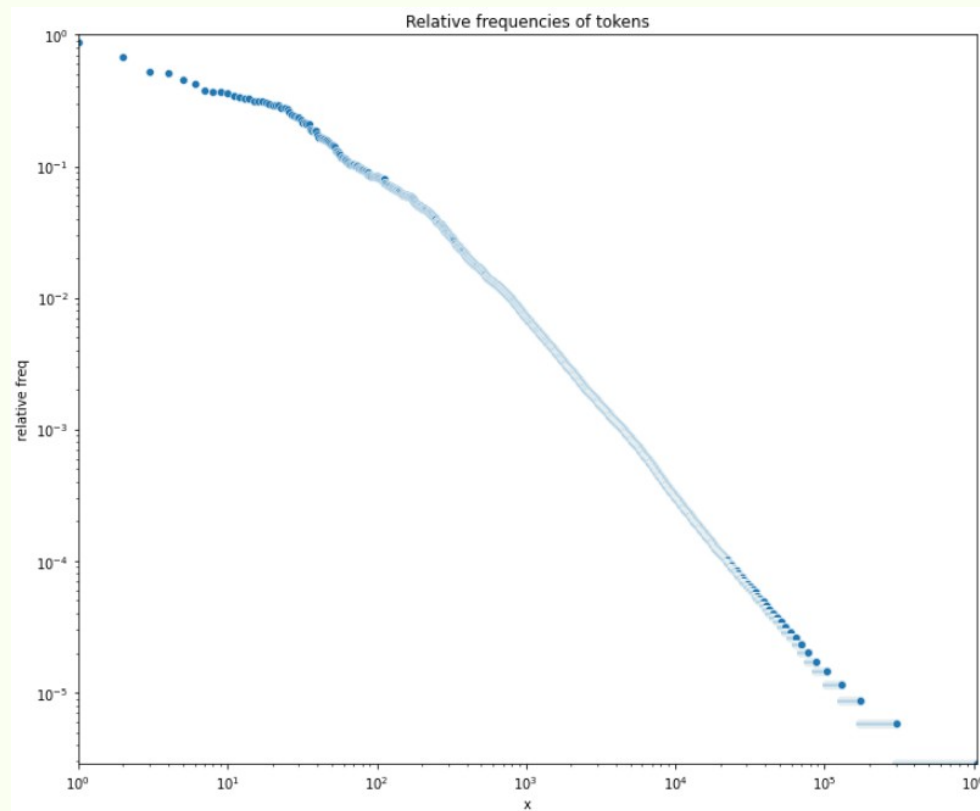
- Most frequent terms don't follow Zipf's relation



- first ~25 tokens frequency declines weakly vs Zipf

What we are dealing with

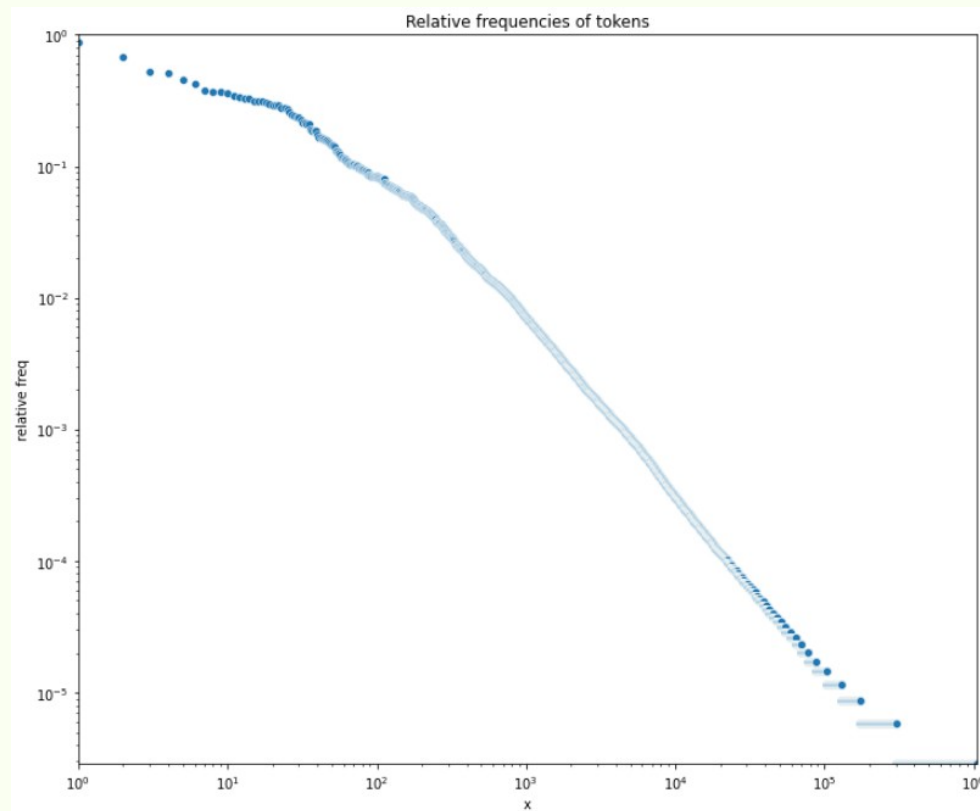
- Most frequent terms don't follow Zipf's relation



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

What we are dealing with

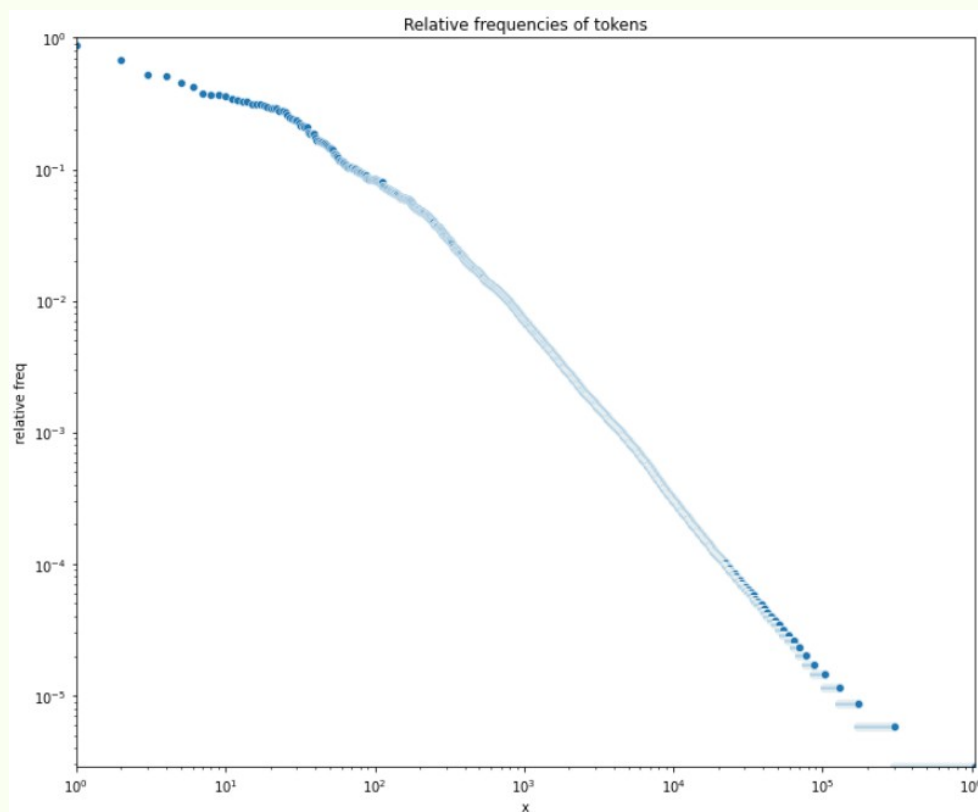
- Most frequent terms don't follow Zipf's relation



- 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 $\sim 10\%$ uncertainty in test stats

Handling Data

Trouble with small classes

- with smallest class sizes $\mathcal{O}(200)$, even train-test split yields $\sim 10\%$ uncertainty in test stats
- \Rightarrow 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 $\sim 10\%$ uncertainty in test stats
- \Rightarrow can't be sure cross-validation picks best model
- \Rightarrow 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 $\sim 10\%$ uncertainty in test stats
- \Rightarrow can't be sure cross-validation picks best model
- \Rightarrow 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 $\sim 10\%$ uncertainty in test stats
- \Rightarrow can't be sure cross-validation picks best model
- \Rightarrow 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 $\sim 10\%$ uncertainty in test stats
- \Rightarrow can't be sure cross-validation picks best model
- \Rightarrow 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 $\sim 10\%$ uncertainty in test stats
- \Rightarrow can't be sure cross-validation picks best model
- \Rightarrow 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 $\sim 10\%$ uncertainty in test stats
- \Rightarrow can't be sure cross-validation picks best model
- \Rightarrow 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`: \Rightarrow Eliminates most vocabulary

Handling Data

tf-idf features

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

Handling Data

tf-idf features

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

Handling Data

tf-idf features

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

Handling Data

tf-idf features

- `min_df=5`: \Rightarrow Eliminates most vocabulary
- `ngram_range=(1, 2)`: \Rightarrow 283 k vocabulary
- `sublinear_tf=True`
- ~~`max_df=0.8`~~: \Rightarrow 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: \Rightarrow optimize for f_1 during grid search

Modeling

See [notebook/DocumentClassificationTest.ipynb](#) in [my repo](#) for details

f1_scorer: \Rightarrow 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: \Rightarrow 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: \Rightarrow 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: \Rightarrow 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
 \Rightarrow model $3 \times$ 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 \Rightarrow 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 \Rightarrow 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 \Rightarrow 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 \Rightarrow opted not to grid search

XGBoost

- much faster training than for the GradientBoostingClassifier (explained later)
- equally excellent results
- optimized model \Rightarrow “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 \Rightarrow opted not to grid search

XGBoost

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

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 \Rightarrow straight average of scores for each class
weighted average \Rightarrow 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 \Rightarrow straight average of scores for each class
weighted average \Rightarrow 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 \Rightarrow straight average of scores for each class
weighted average \Rightarrow 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

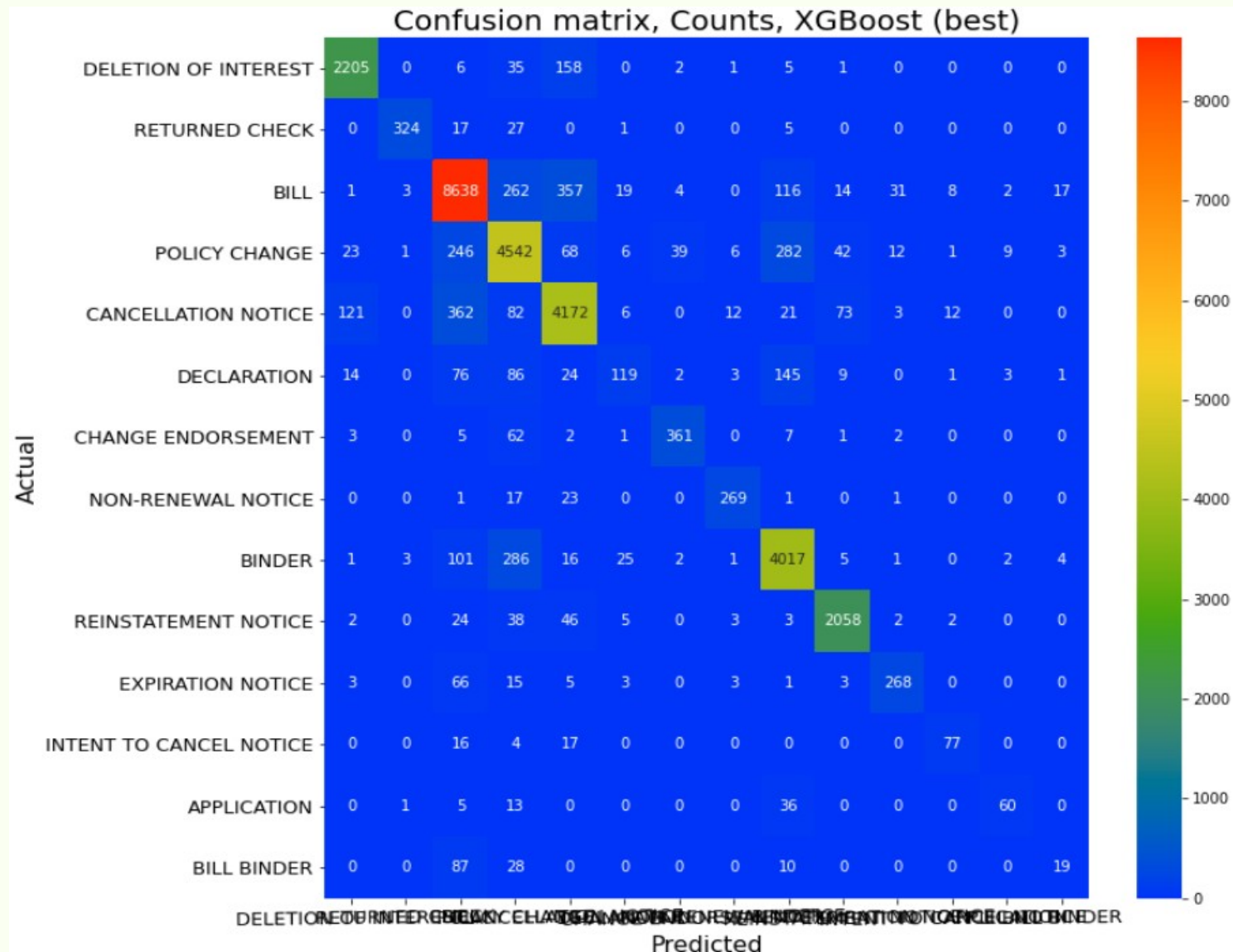
- reminder: macro averaged \Rightarrow straight average of scores for each class
weighted average \Rightarrow average of all class scores weighted by support
- if need good scores for smaller classes, focus on macro averages
- if overall results most important, focus on weighted averages
- Random Forest and XGBoost “identical” good results

Model Results

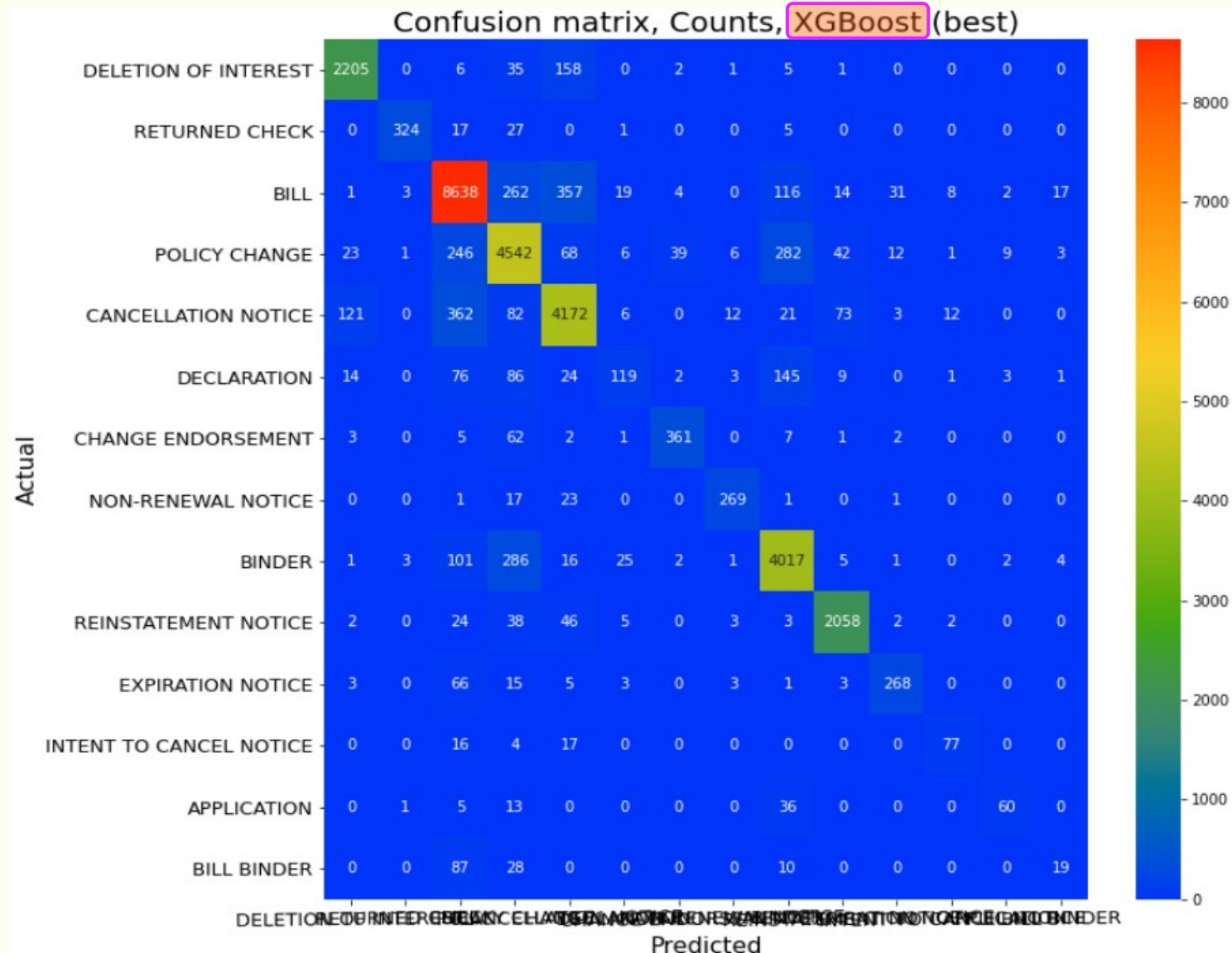
Model	Macro Averaged			Weighted Average			Model size (MB)
	precision	recall	f_1	precision	recall	f_1	
Naive Bayes default (baseline)	0.75	0.50	0.53	0.79	0.78	0.76	63
Naive Bayes best	0.80	0.58	0.62	0.81	0.81	0.80	272
Random Forest default	0.80	0.65	0.70	0.84	0.85	0.84	451
Random Forest best	0.80	0.75	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	0.50	0.61	0.52	0.78	0.69	0.73	202

- reminder: macro averaged \Rightarrow straight average of scores for each class
weighted average \Rightarrow 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 0 (10%) \Rightarrow impact macro averages most

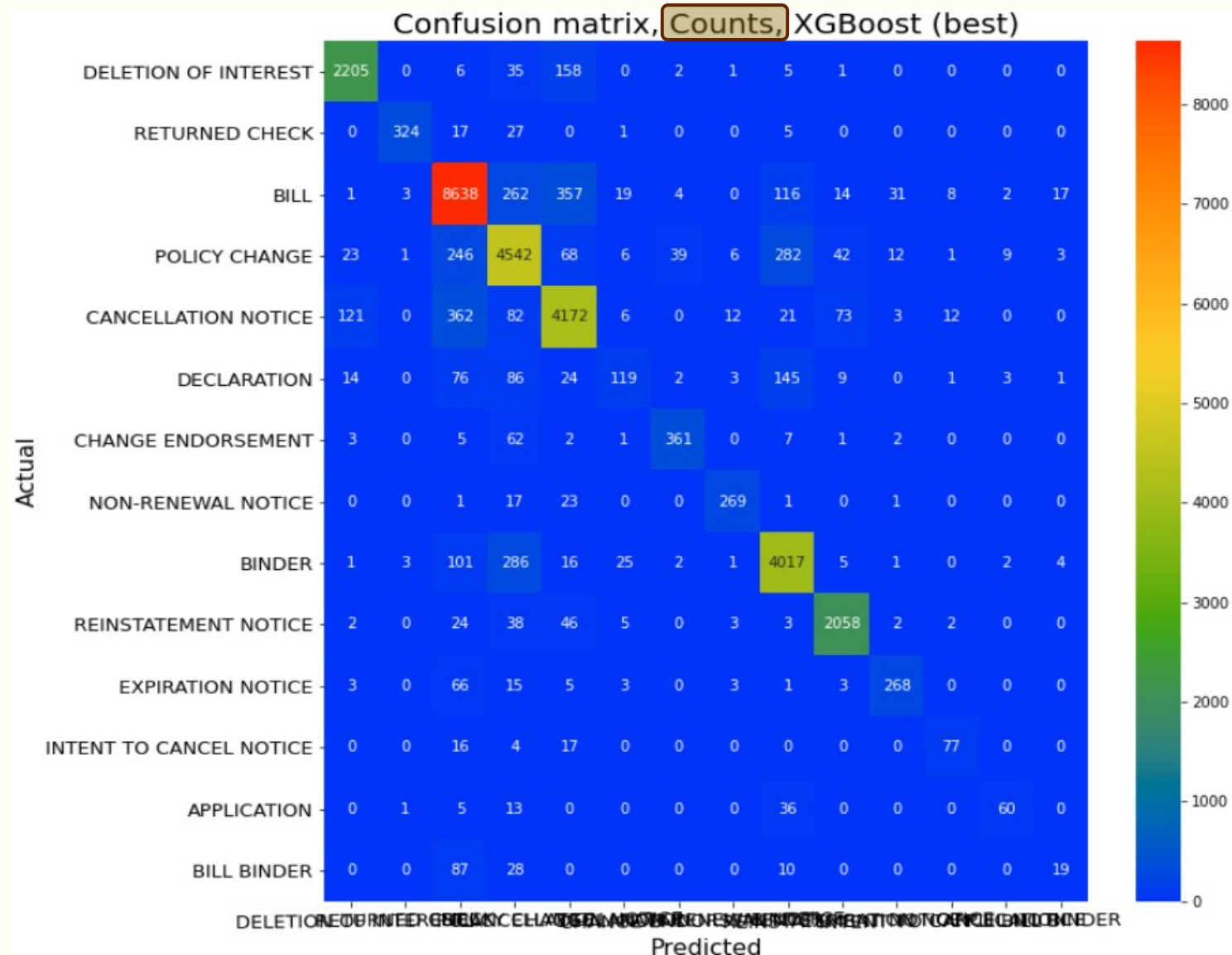
Model Results



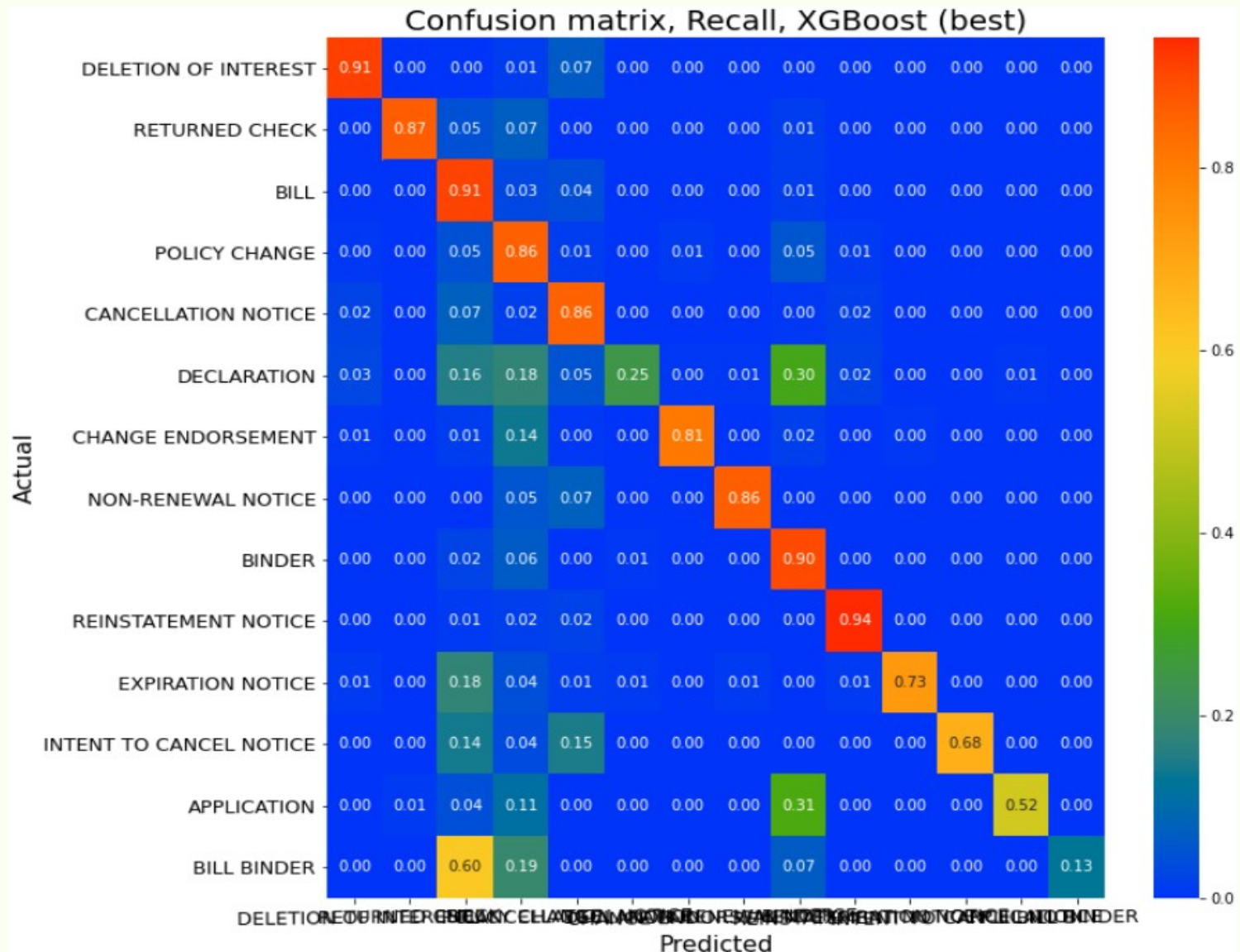
Model Results



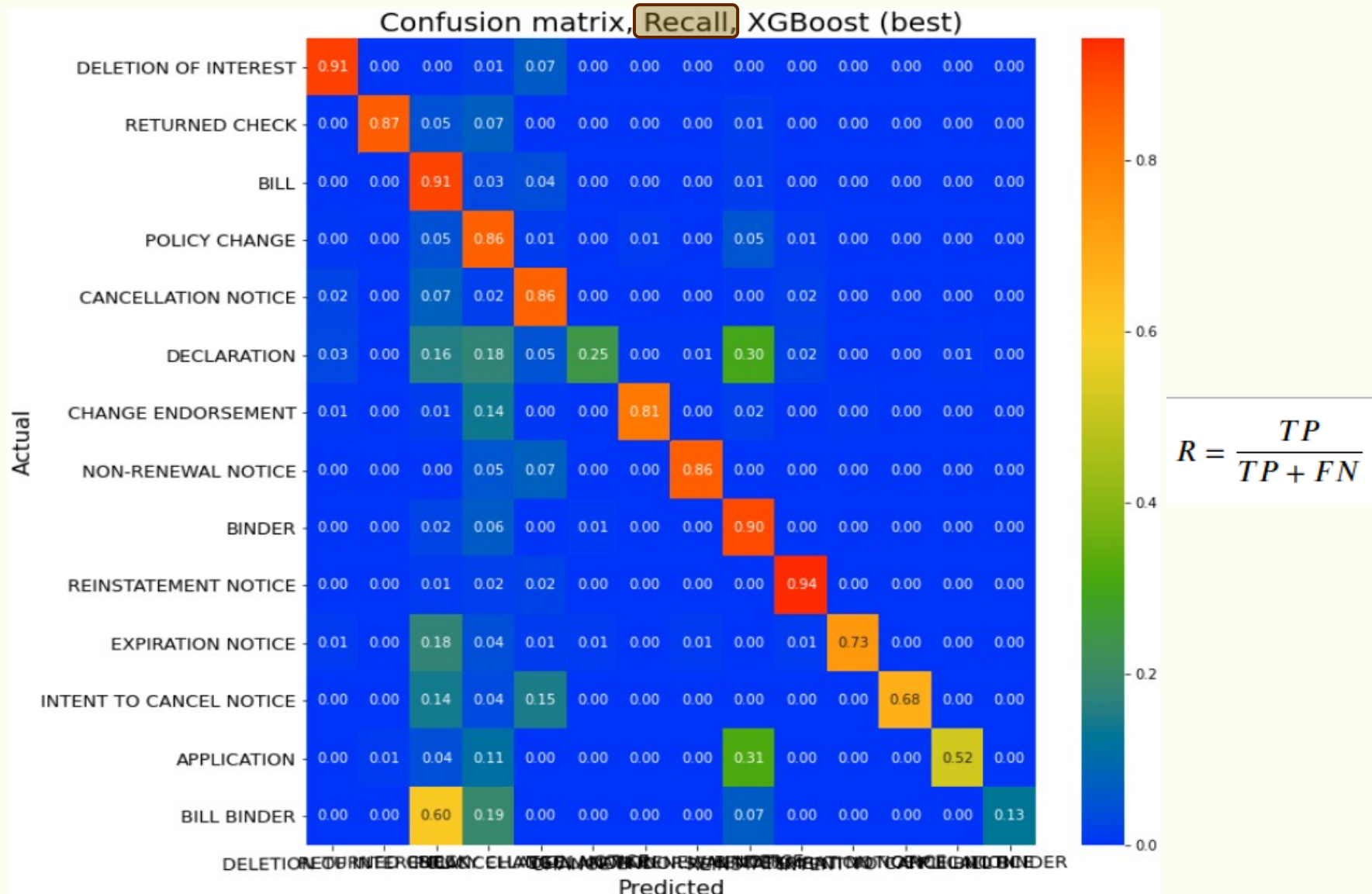
Model Results



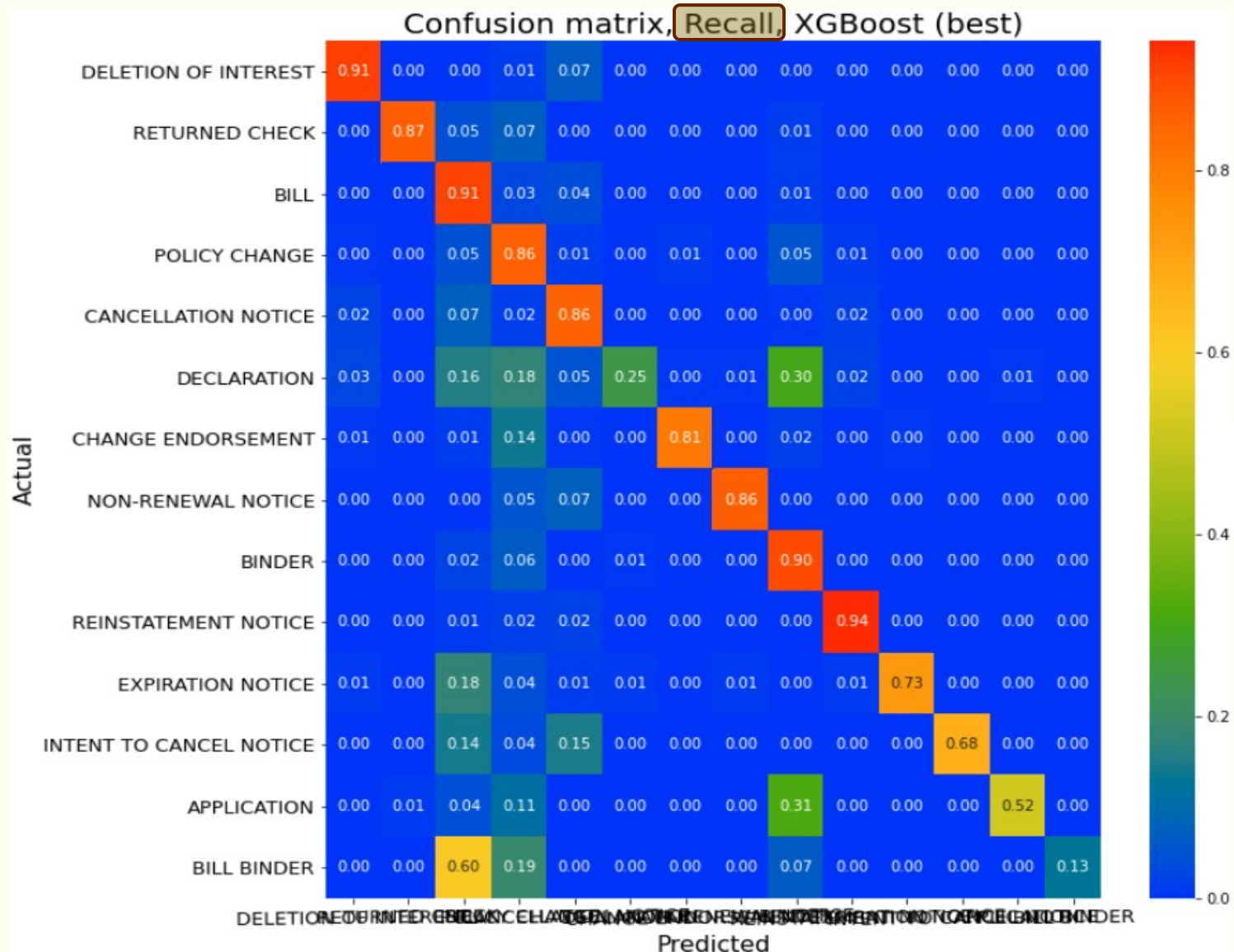
Model Results



Model Results

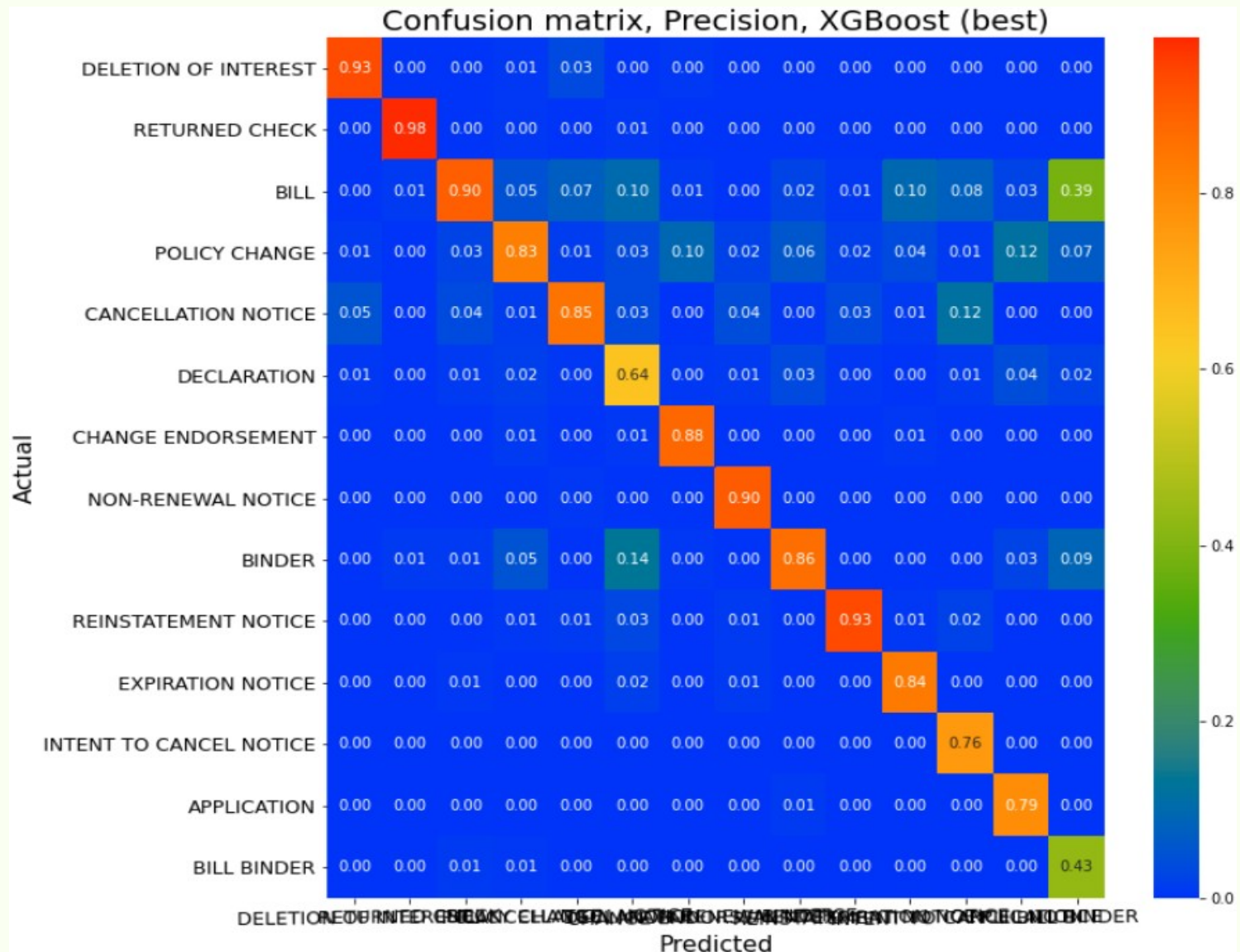


Model Results

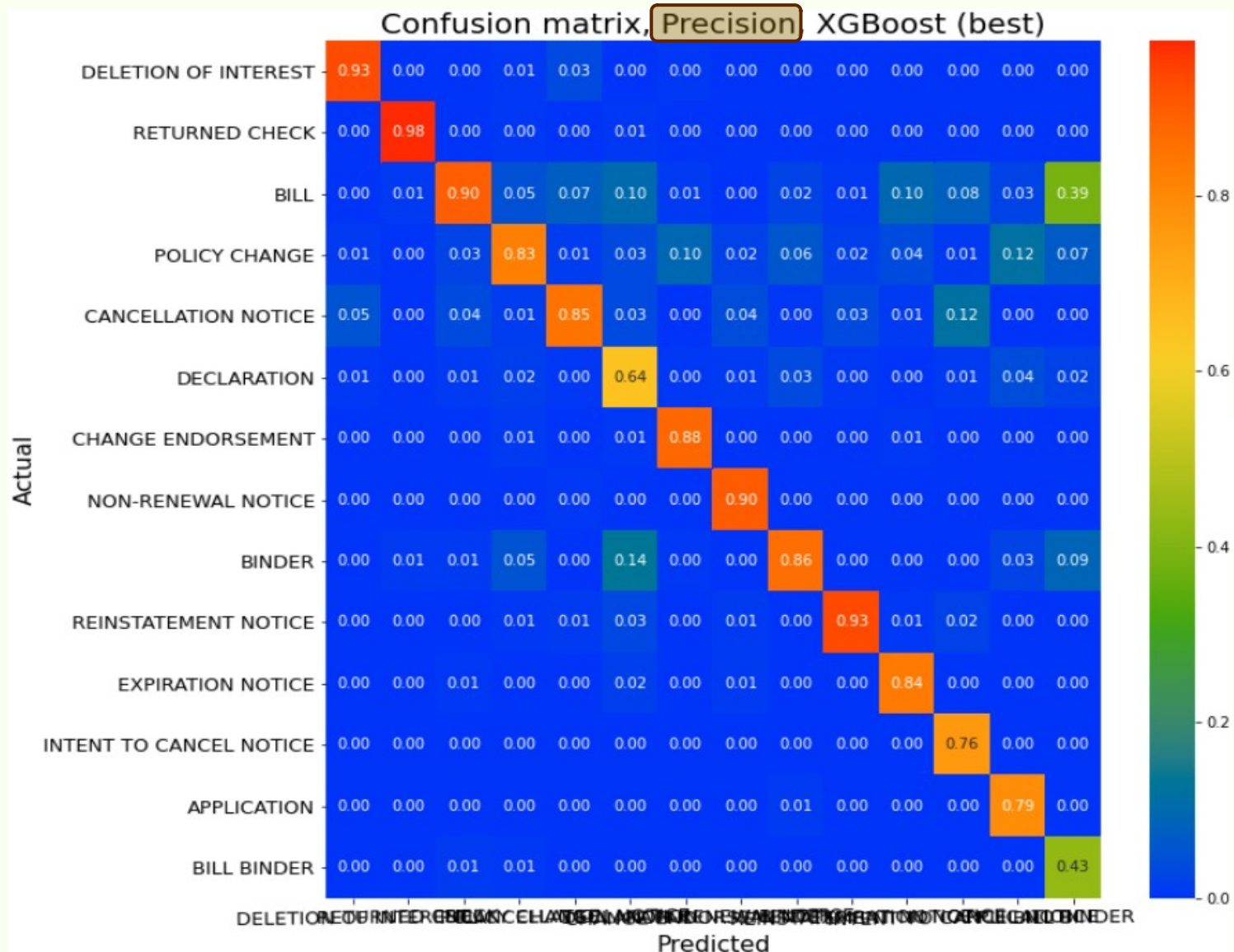


$$R = \frac{TP}{TP + FN}$$

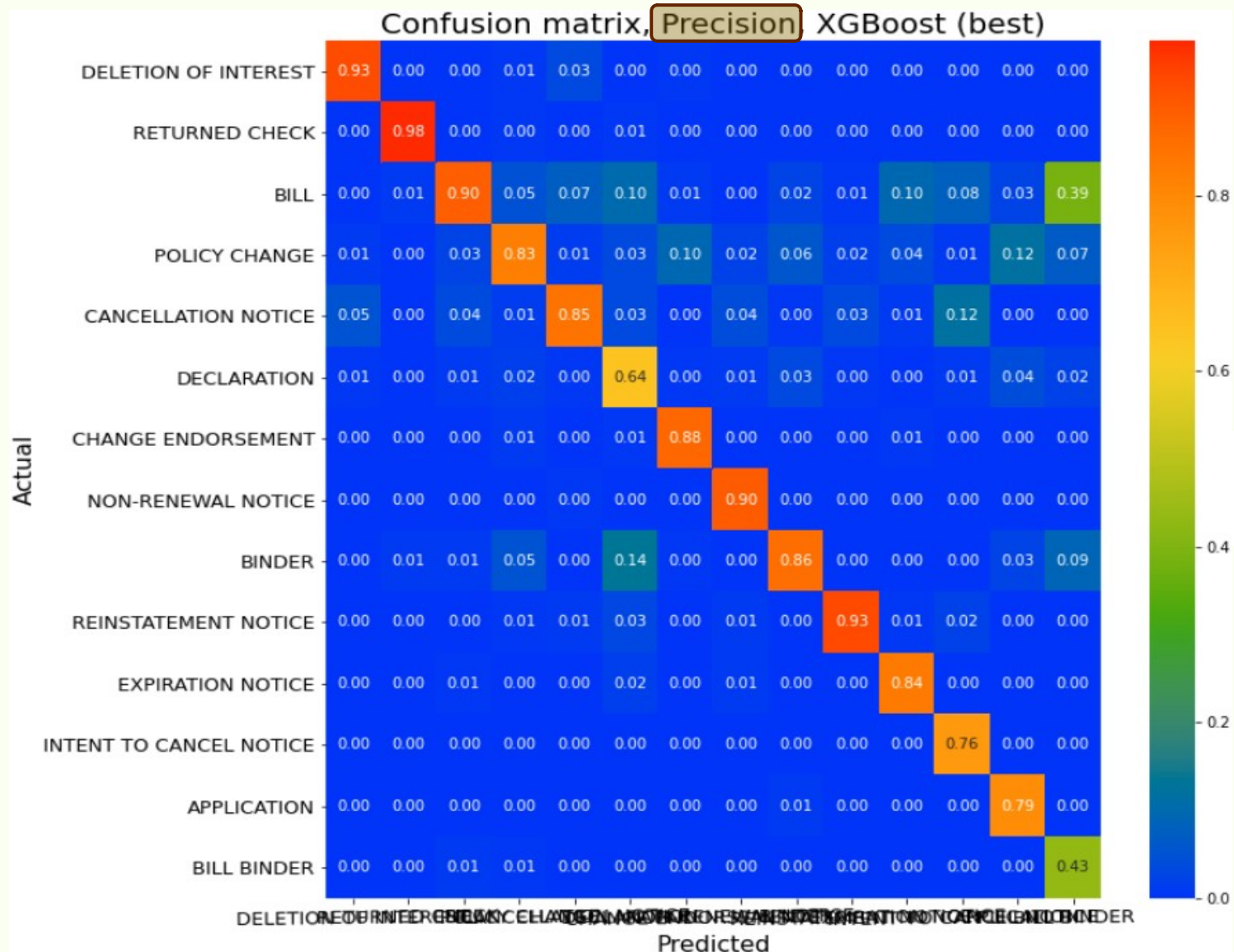
Model Results



Model Results



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 GradientBoostingClassifier

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 GradientBoostingClassifier
 - 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 GradientBoostingClassifier
 - 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 GradientBoostingClassifier
 - 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 GradientBoostingClassifier
 - 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 GradientBoostingClassifier
 - 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 GradientBoostingClassifier
 - 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 GradientBoostingClassifier
 - 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

The screenshot shows the AWS IAM console interface for a role. The 'Permissions' tab is active, displaying a list of policies. Two policies are applied: 'AWSLambdaBasicExecutionRole-e0580e60-7813-4c5b-b5de-9754d93cc1dc' (Managed policy) and 'SageMakerInvokeEndpoint' (Inline policy). The 'SageMakerInvokeEndpoint' policy is expanded, showing its JSON definition. The JSON allows the 'sagemaker:InvokeEndpoint' action on all resources.

Permissions policies (2 policies applied)

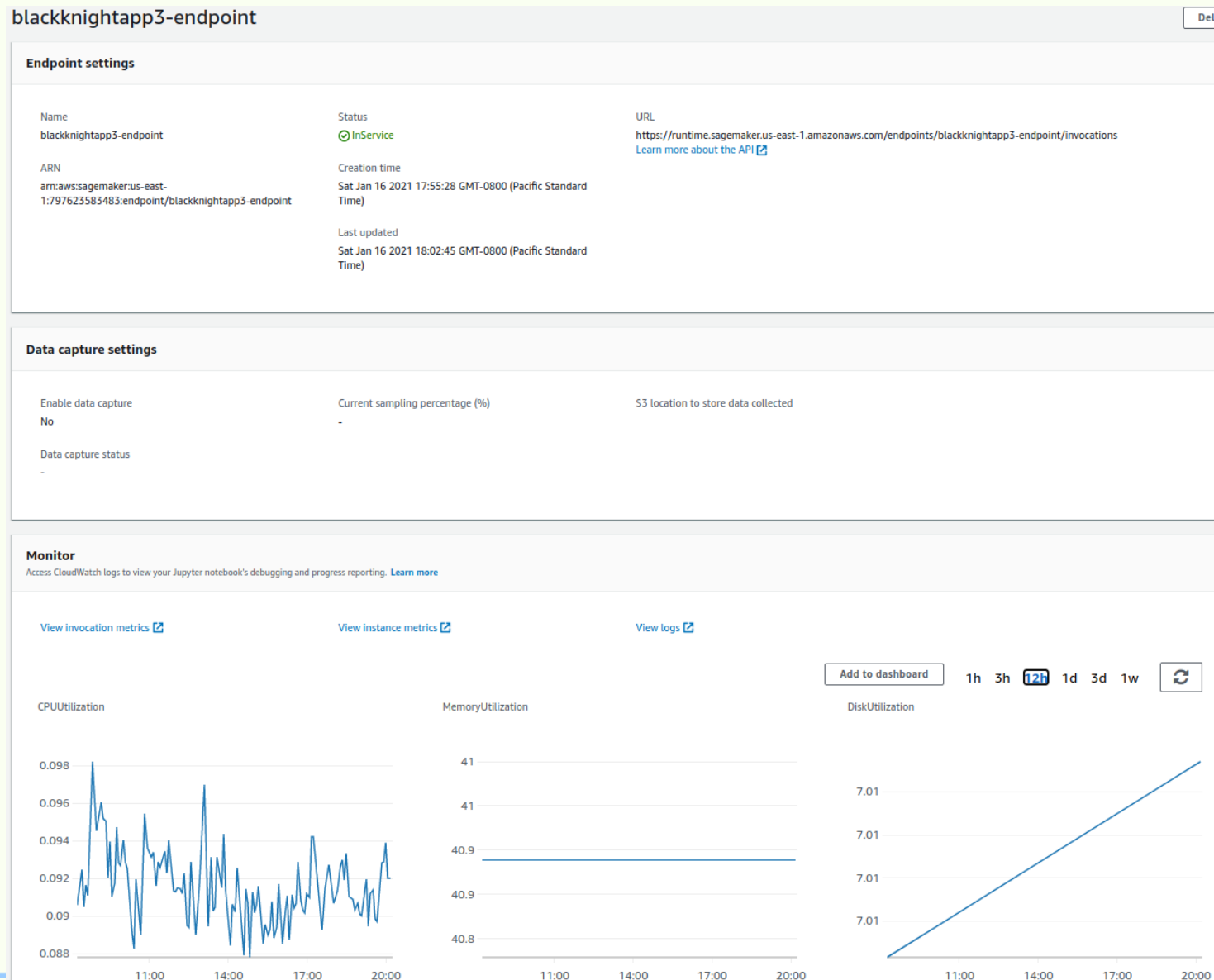
Attach policies [+ Add inline policy](#)

Policy name	Policy type
AWSLambdaBasicExecutionRole-e0580e60-7813-4c5b-b5de-9754d93cc1dc	Managed policy
SageMakerInvokeEndpoint	Inline policy

Policy summary [{} JSON](#) [Edit policy](#) [Simulate policy](#)

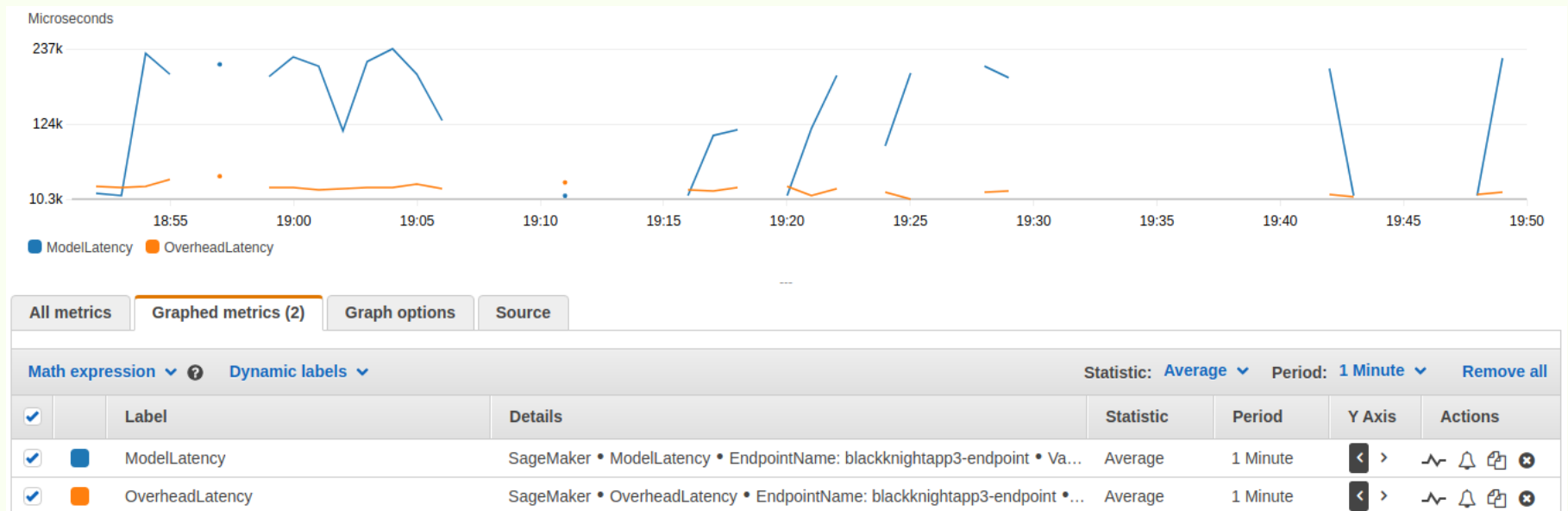
```
1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Sid": "Stmt1464440182000",
6       "Effect": "Allow",
7       "Action": [
8         "sagemaker:InvokeEndpoint"
9       ],
10      "Resource": [
11        "*"
12      ]
13    }
14  ]
15 }
```

Deployed Endpoint



Deployed Endpoint

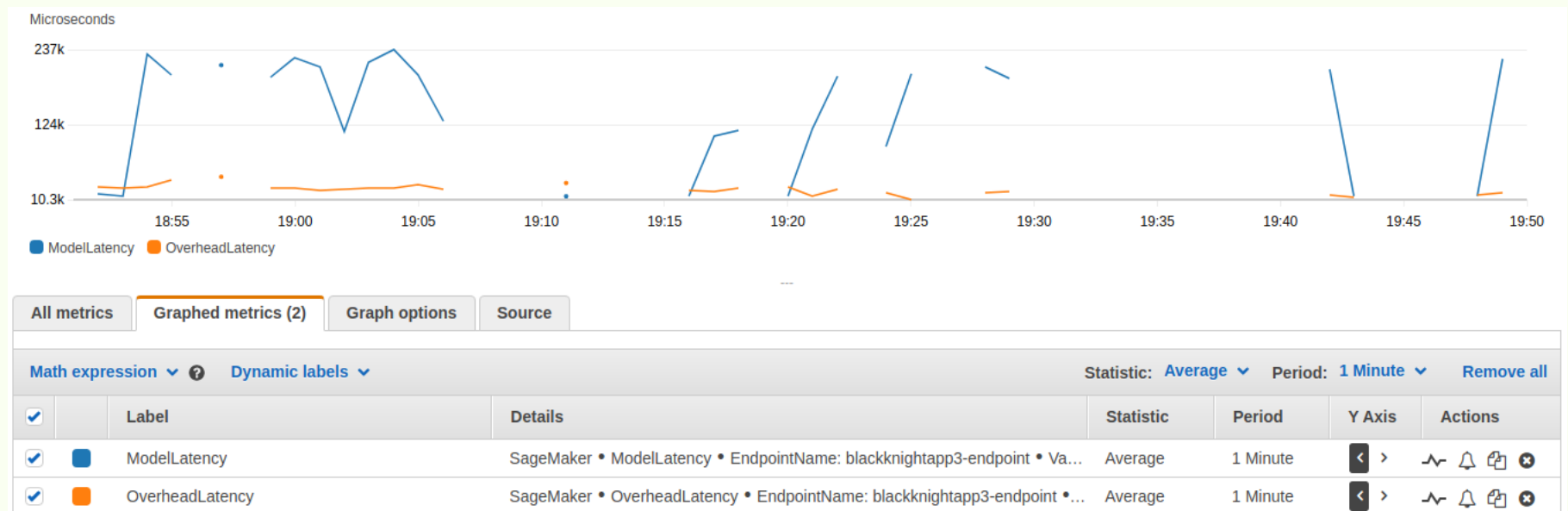
Latencies



- 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

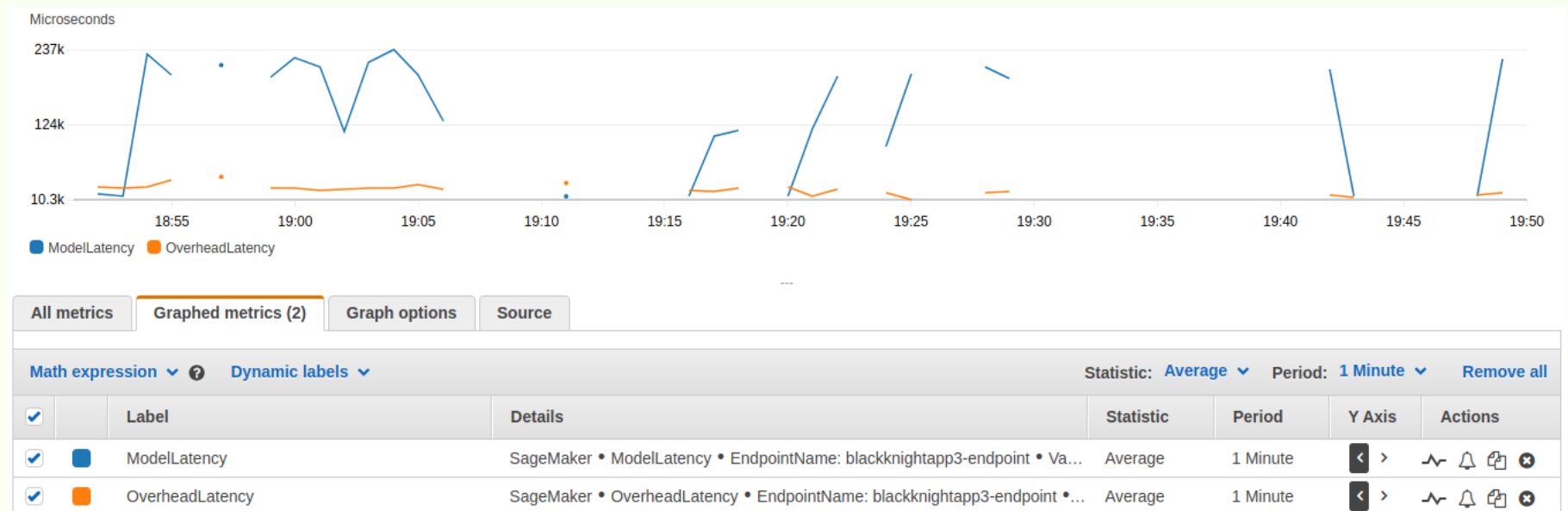
Latencies



- 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

About

This is a quick UI for testing classifiers created for the Problem. Multiple models were trained on TF-IDF features, with two available here:

- Naive Bayes with default settings, serving as a baseline
- Random Forest model, selected after multiple hours of grid search

Mash on the 'Get results!' button, and document strings in the box to the right will be JSONified and shipped to a RESTful API hosted on AWS. The model name, set by the radio button to the right, will determine which model to use for inference.

The returned JSON payload is formatted and displayed.

⚡ Alternatively, see Upload File option below for batch processing (below). For details, see [my fork](#) of HeavyWater's original github repo.

Upload File (batch processing)

⇒ After submitting file, wait for [Download results](#) link below ...

Drag and drop file here

Limit 200MB per file

Format Replace <model_name> with "NaiveBays" or "RandomForest":

```
<model_name>
07e7fe209a3b ... many tokens ... 93c9f
c1a2676df403 ... many tokens ... f1411
8b0131ee1005 ... many tokens ... 12654
.
.
.
_
```

Document Classifier for Black Knight HeavyWater

blame: Mark Wilber

Replace these example documents with your own, each separated by at least 1 new line.
(Note also: batch file upload in sidebar)

```
ad4440ac97a5 8e93a2273a93 c913f5129fe2 bfb030c0e4e2 6ce6cc5a3203 798fe9915030 42e211f8752a
7eb23b5b9603 f7ae6f8257da 9d634fae0367 2f2548bd374a 25c57acdf805 75df40507e72 ffe8decfd82e 422068f04236
3e56fed2d392 063a3ef1e75f 8db54a4cb57b 25c57acdf805 e52882a7f2b7 8db54a4cb57b 37ac79620fc6 596fbbd504aa
ffe216d9d610 6868362b998e fc96b835cfc3 ffe216d9d610 6868362b998e eca16ee06b98 25c57acdf805
641356219cbb 422068f04236 5f43e051f9a6 48d657cd9861 fc1955933b8e eca16ee06b98 957b5cf4e65e
422068f04236 fb53275d6678 f56b300cc325 48d657cd9861 6101ed18e42f 586242498a88 48d657cd9861
6b343f522f78 8db54a4cb57b e7e059c82399 6ca2dd348663 b87f34b0269a bfb030c0e4e2 d38820625542
e943e5e5b779 c8d2304e52cf fbe267908bc5 2f2548bd374a cfbf3eb99bea 6ce6cc5a3203 d19b1c129f40 5f43e051f9a6
586242498a88 c8f5ad40a683 4ffb12504ac6 8cb71bb0ee27 66813d53f12a bdba286f728a f7ae6f8257da
938812903b4e 5f43e051f9a6 8cb71bb0ee27 fbe267908bc5 fbe267908bc5 2f2548bd374a a100eb50abec
2f2548bd374a ad4440ac97a5 cf4fc632eed2 2f2548bd374a 25c57acdf805 422068f04236 d19b1c129f40 a3518ffa104e
5f43e051f9a6 33043bd1c2f4 db108078ec43 5ff8f7117bc9 8cb71bb0ee27 0e9329e43507 6b3268e10628
e7e059c82399 bfb030c0e4e2 744366456381 e259a56993f4 e3a330c58136 d671855584fd eeb86a6a04e4
a3518ffa104e d736fc77c54b fbe267908bc5 fbe267908bc5 586242498a88 f7ae6f8257da a5f8a7c9a886 0c4ce226d9fe
9b88c973ae02 21e314d3afcc 11a897cb0d78 d493c688fb66 8cb71bb0ee27 de9738ee8b24 7bf4f79c3fd9
6365c4563bd1 9374c105ef84 de9738ee8b24 25c57acdf805 37ac79620fc6 8f7a92cd0ae7 cf4fa36520cb ad4440ac97a5
eb51798a89e1 8cb71bb0ee27 a100eb50abec f7ae6f8257da f7ae6f8257da 19e9f3592995 586242498a88
bfb030c0e4e2 37ac79620fc6 8cb71bb0ee27 4ffb12504ac6 10aa76ec946b ffe216d9d610 c24d76b5b80a
ba02159e05b1 033616ad6870 d2cabcd692f6 8f7a92cd0ae7 360e8b28421c 21c66f6b38af a7c177a24cab
ffe216d9d610 db108078ec43 5dc515102c7b ce02bbbeb97f 3b952c633ee4 f7d55eadc647 ad4440ac97a5
033616ad6870 038043bd66da fbf030c0e4e2 da046a9d8e36 70a81d6ffab b60e6ed0d053 32b5989b13f0
72bd4a50cf4a 1b21cf220a68 ded7b70601fc 292891f020a4 586242498a88 6f6729c54a07 60439259777c
8cb71bb0ee27 f7ae6f8257da c82f81aceab fbe267908bc5 2f2548bd374a ffe216d9d610 ce00eff819b7 25c57acdf805
f7ae6f8257da 0704e636f7b8 ad4440ac97a5 f7ae6f8257da 586242498a88 21c66f6b38af 8f75273e5510
8cb71bb0ee27 789f72dda0b0 2c129538d383 3b952c633ee4 bfb030c0e4e2 bfb030c0e4e2 e851bc6d8f3a
ad4440ac97a5 d671855584fd 6101ed18e42f d19b1c129f40 33043bd1c2f4 4ffb12504ac6 bfb030c0e4e2
8359/12000
```

Which Model?

☐ Naive Bayes (baseline)

☒ RandomForest (optimized)

☒ Show formatted request

Response from API

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

About

This is a quick UI for testing classifiers created for the Problem. Multiple models were trained on TF-IDF features, with two available here:

- Naive Bayes with default settings, serving as a baseline
- Random Forest model, selected after multiple hours of grid search

Mash on the 'Get results!' button, and document strings in the box to the right will be JSONified and shipped to a RESTful API hosted on AWS. The model name, set by the radio button to the right, will determine which model to use for inference.

The returned JSON payload is formatted and displayed.

⚡ Alternatively, see Upload File option below for batch processing (below). For details, see [my fork](#) of HeavyWater's original github repo.

Upload File (batch processing)

⇒ After submitting file, wait for [Download results](#) link below ...

Drag and drop file here
Limit 200MB per file

Browse files

Format Replace <model_name> with "NaiveBays" or "RandomForest":

```
<model_name>
07e7fe209a3b ... many tokens ... 93c9f
c1a2676df403 ... many tokens ... f1411
8b0131ee1005 ... many tokens ... 12654
```

Document Classifier for Black Knight HeavyWater

blame: Mark Wilber

Replace these example documents with your own, each separated by at least 1 new line.
(Note also: batch file upload in sidebar)

```
ad4440ac97a5 8e93a2273a93 c913f5129fe2 bfb030c0e4e2 6ce6cc5a3203 798fe9915030 42e211f8752a
7eb23b5b9603 f7ae6f8257da 9d634fae0367 2f2548bd374a 25c57acdf805 75df40507e72 ffe8decfd82e 422068f04236
3e56fed2d392 063a3ef1e75f 8db54a4cb57b 25c57acdf805 e52882a7f2b7 8db54a4cb57b 37ac79620fc6 596fbbd504aa
ffe216d9d610 6868362b998e fc96b835cfc3 ffe216d9d610 6868362b998e eca16ee06b98 25c57acdf805
641356219cbc 422068f04236 5f43e051f9a6 48d657cd9861 fc1955933b8e eca16ee06b98 957b5cf4e65e
422068f04236 fb53275d6678 f56b300cc325 48d657cd9861 6101ed18e42f 586242498a88 48d657cd9861
6b343f522f78 8db54a4cb57b e7e059c82399 6ca2dd348663 b87f34b0269a bfb030c0e4e2 d38820625542
e943e5e5b779 c8d2304e52cf fbe267908bc5 2f2548bd374a cfbf3eb99bea 6ce6cc5a3203 d19b1c129f40 5f43e051f9a6
586242498a88 c8f5ad40a683 4ffb12504ac6 8cb71bb0ee27 66813d53f12a bdba286f728a f7ae6f8257da
938812903b4e 5f43e051f9a6 8cb71bb0ee27 fbe267908bc5 fbe267908bc5 2f2548bd374a a100eb50abec
2f2548bd374a ad4440ac97a5 cf4fc632eed2 2f2548bd374a 25c57acdf805 422068f04236 d19b1c129f40 a3518ffa104e
5f43e051f9a6 33043bd1c2f4 db108078ec43 5ff8f7117bc9 8cb71bb0ee27 0e9329e43507 6b3268e10628
e7e059c82399 bfb030c0e4e2 744366456381 e259a56993f4 e3a330c58136 d671855584fd eeb86a6a04e4
a3518ffa104e d736fc77c54b fbe267908bc5 fbe267908bc5 586242498a88 f7ae6f8257da a5f8a7c9a886 0c4ce226d9fe
9b88c973ae02 21e314d3afcc 11a897cb0d78 d493c688fb66 8cb71bb0ee27 de9738ee8b24 7bf4f79c3fd9
6365c4563bd1 9374c105ef84 de9738ee8b24 25c57acdf805 37ac79620fc6 8f7a92cd0ae7 cf4fa36520cb ad4440ac97a5
eb51798a89e1 8cb71bb0ee27 a100eb50abec f7ae6f8257da f7ae6f8257da 19e9f3592995 586242498a88
bfb030c0e4e2 37ac79620fc6 8cb71bb0ee27 4ffb12504ac6 10aa76ec946b ffe216d9d610 c24d76b58b0a
ba02159e05b1 033616ad6870 d2cabcd692f6 8f7a92cd0ae7 360e8b28421c 21c66f6b38af a7c177a24cab
ffe216d9d610 db108078ec43 5dc515102c7b ce02bbbeb97f 3b952c633ee4 f7d55eadc647 ad4440ac97a5
033616ad6870 038043bd66da bfb030c0e4e2 da046a9d8e36 70a81d6ffab b60e6ed0d053 32b5989b13f0
72bd4a50cf4a 1b21cf220a68 ded7b70601fc 292891f020a4 586242498a88 6f6729c54a07 60439259777c
8cb71bb0ee27 f7ae6f8257da c82f81aceab fbe267908bc5 2f2548bd374a ffe216d9d610 ce00eff819b7 25c57acdf805
f7ae6f8257da 0704e636f7b8 ad4440ac97a5 f7ae6f8257da 586242498a88 21c66f6b38af 8f75273e5510
8cb71bb0ee27 789f72dda0b0 2c129538d383 3b952c633ee4 bfb030c0e4e2 bfb030c0e4e2 e851bc6d8f3a
ad4440ac97a5 d671855584fd 6101ed18e42f d19b1c129f40 33043bd1c2f4 4ffb12504ac6 bfb030c0e4e2
```

8359/12006

Which Model?

☐ Naive Bayes (baseline)

☒ RandomForest (optimized)

Get results!

Response from API

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!