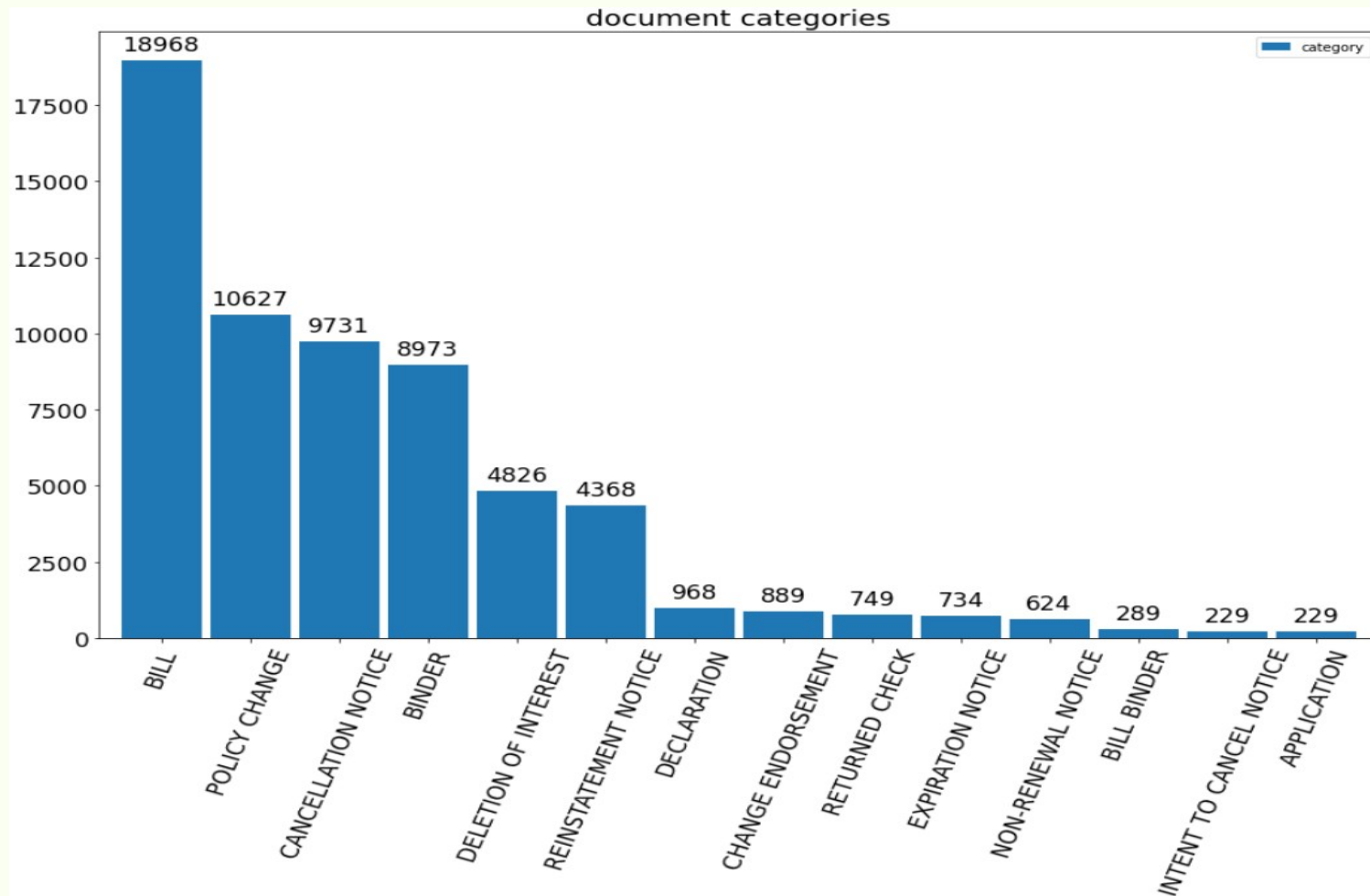


HeavyWater Machine Learning Problem

Solution by Mark Wilber

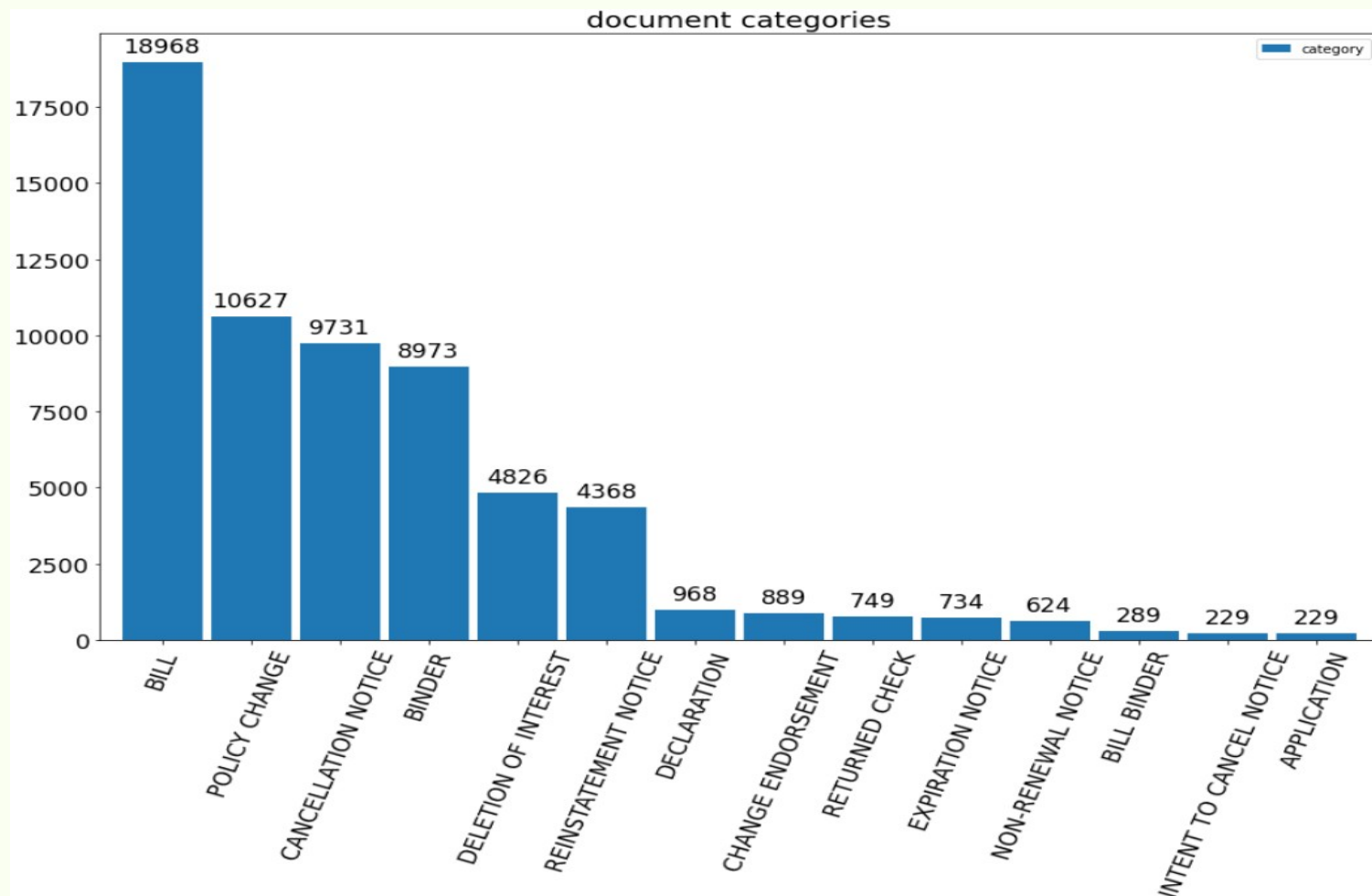
What we are dealing with

- 62,204 documents, 14 categories



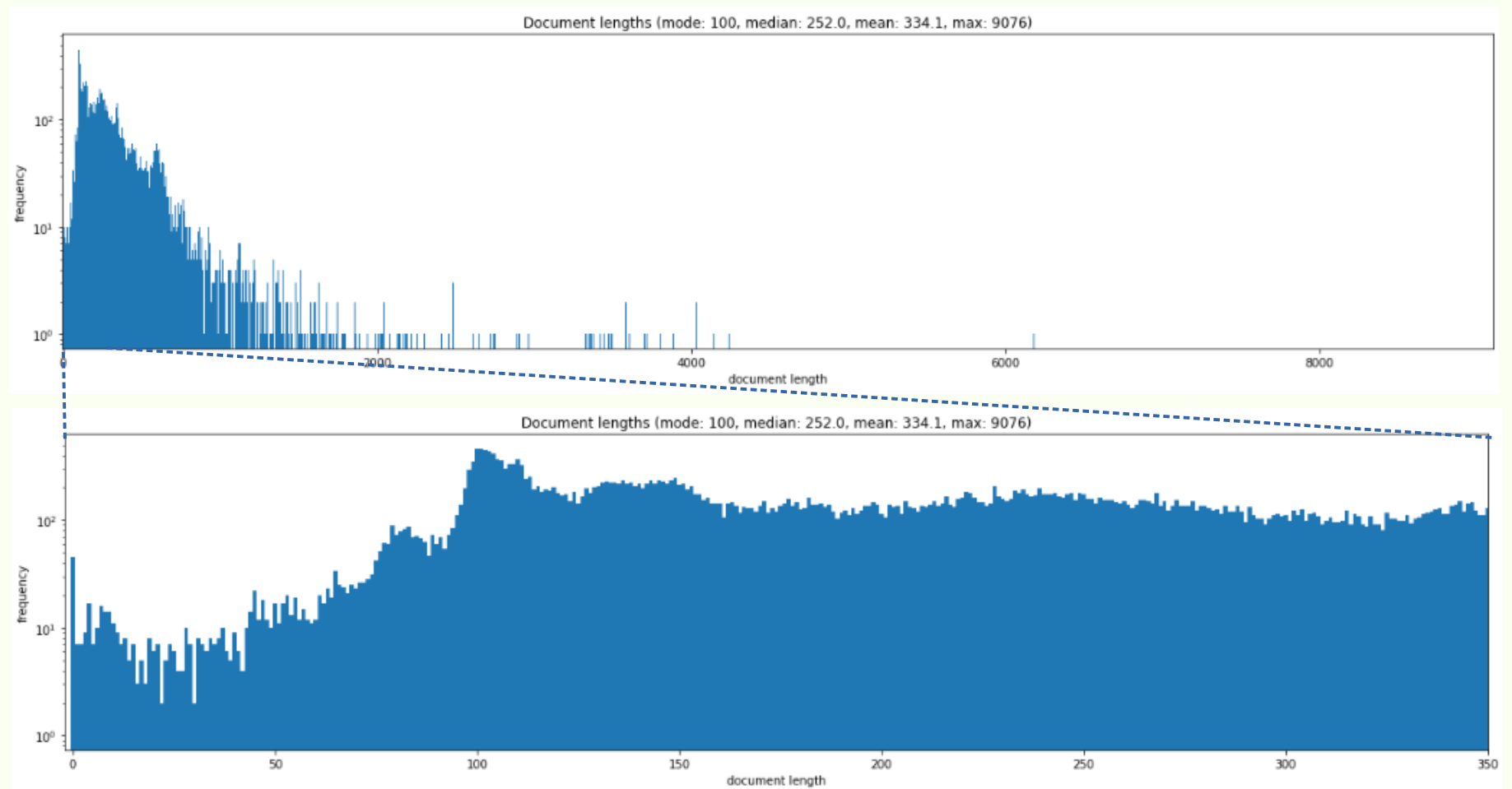
What we are dealing with

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



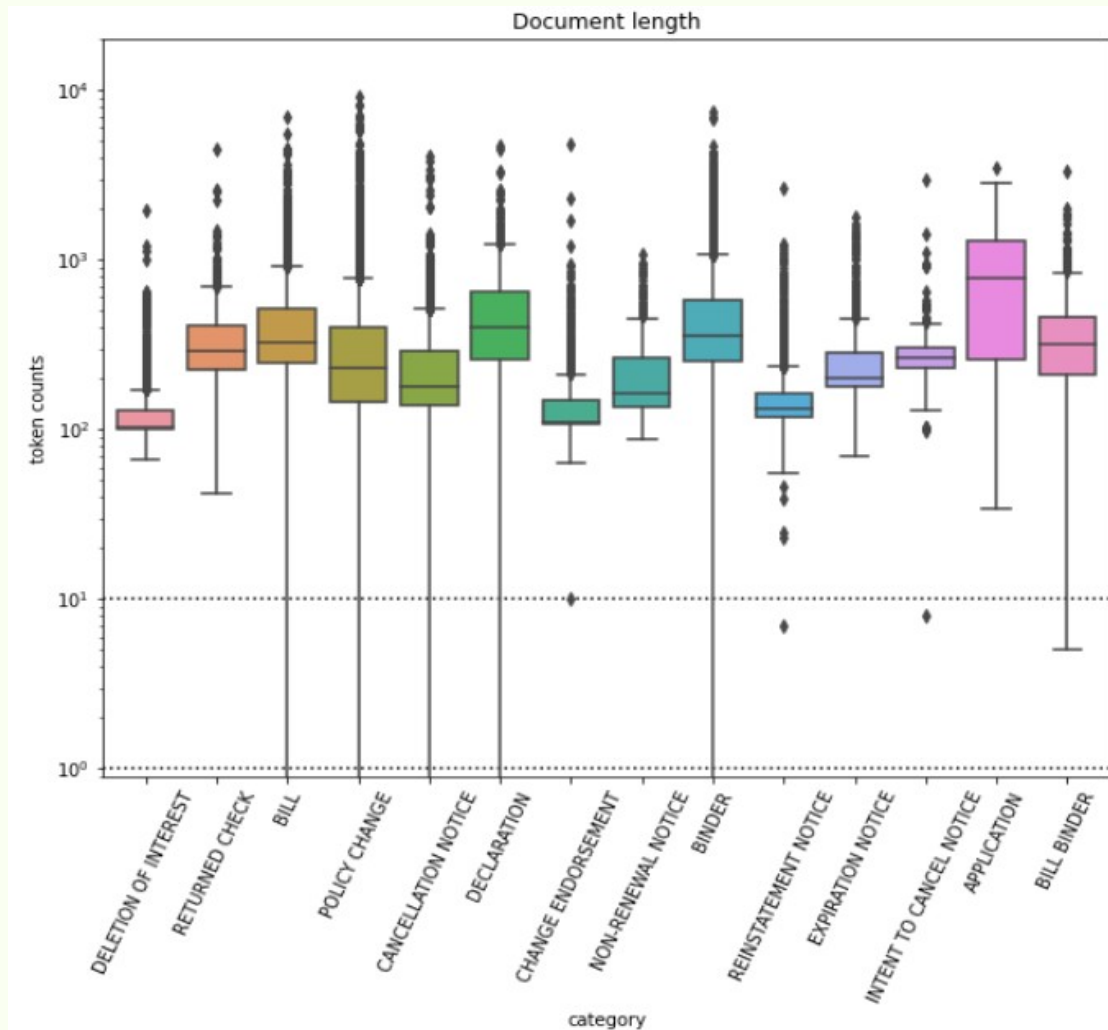
What we are dealing with

- 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

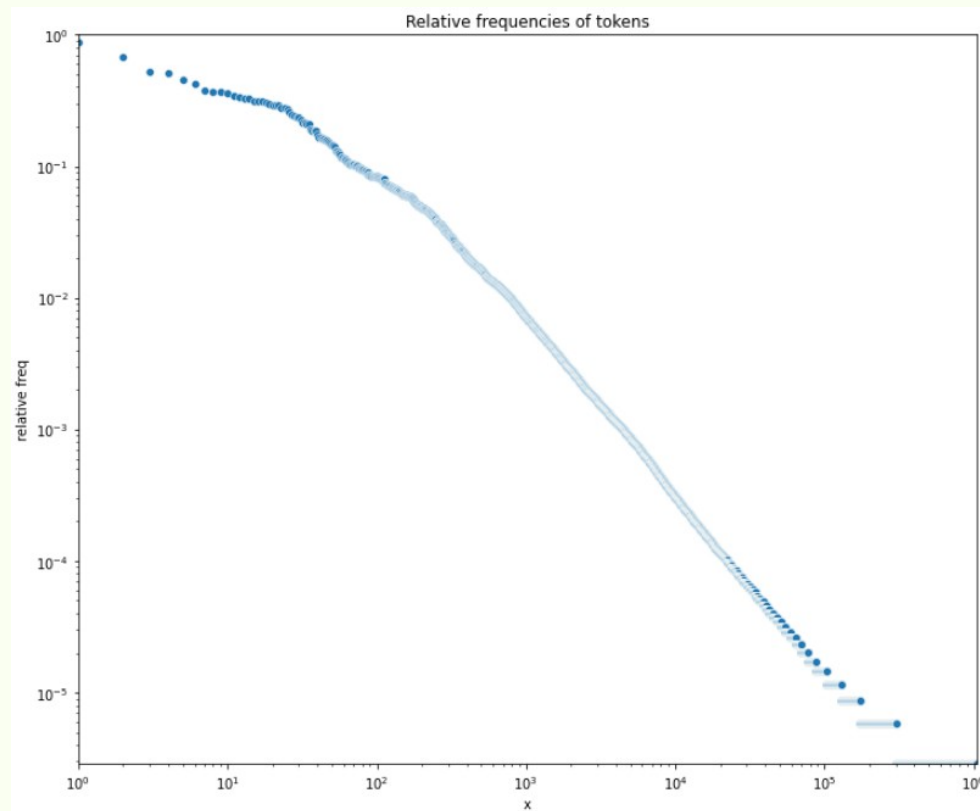
- explanation: most of the tokens are “uninformative” (garbage)
 - 71% of tokens only appear once, 92.6% occur 6 × or fewer
 - A small fraction are names (of humans, businesses), special codes

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

⇒ smudges create nonsense terms

What we are dealing with

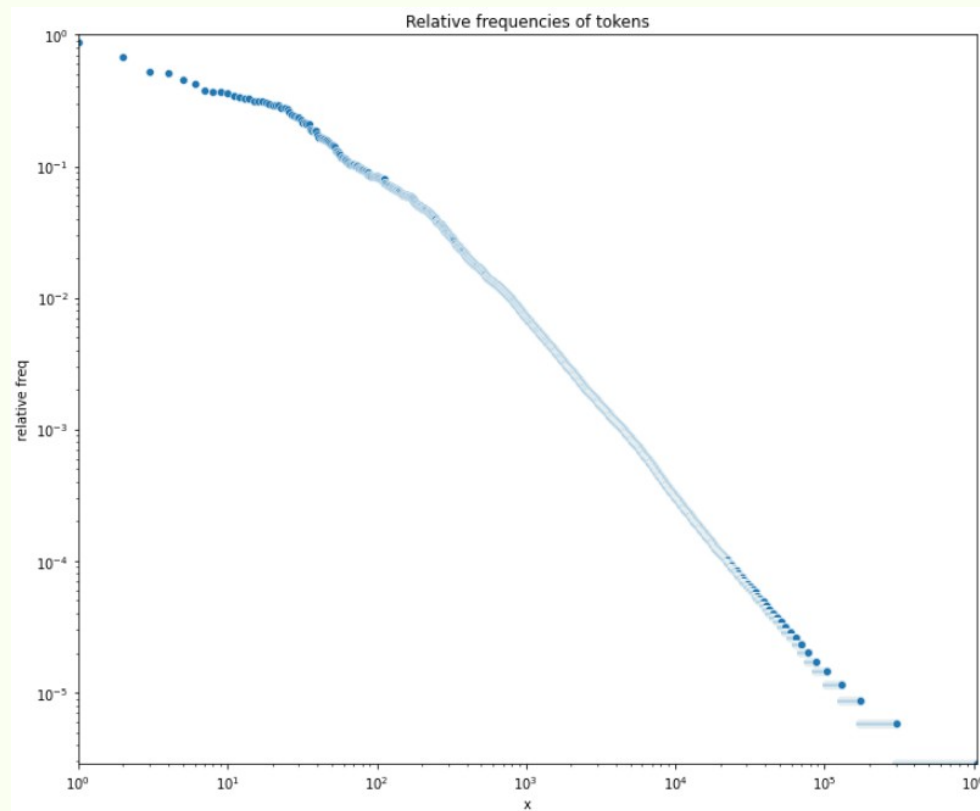
- Most frequent terms don't follow Zipf's relation



- first ~25 tokens frequency declines weakly vs Zipf

What we are dealing with

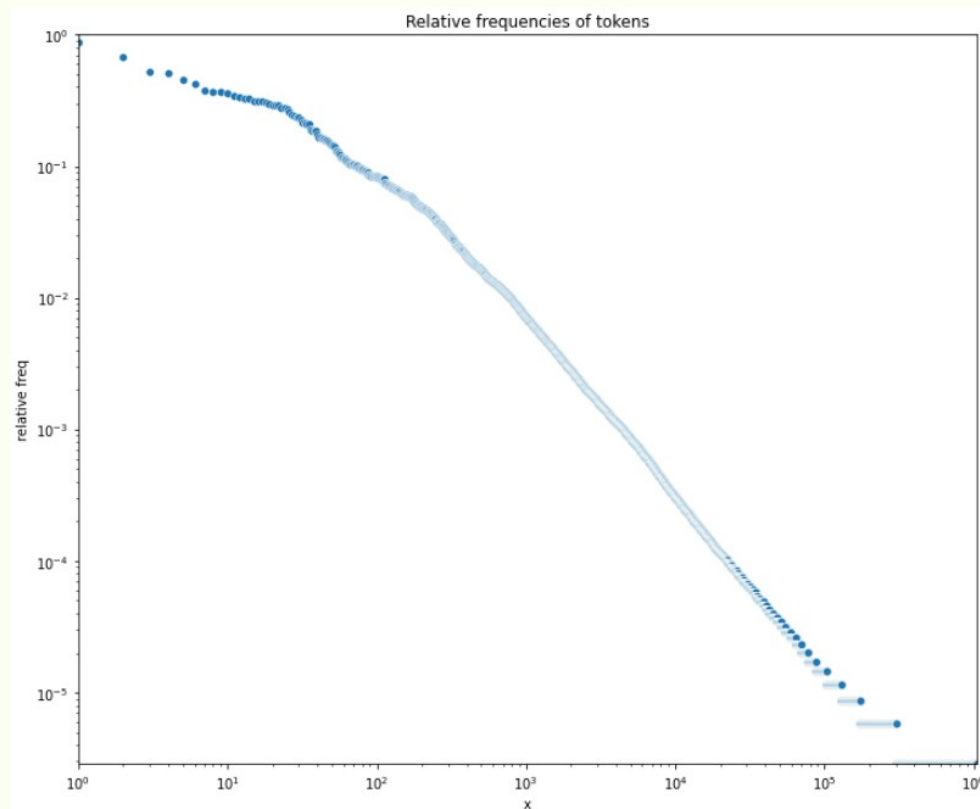
- Most frequent terms don't follow Zipf's relation



- first ~25 tokens frequency declines weakly vs Zipf
- 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
- 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)
- 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)
- 50-50 test-train split to retain plausible stats on results, at some cost to performance ...
- 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)
- 50-50 test-train split to retain plausible stats on results, at some cost to performance ...
- stratified sampling
- 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 yield 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 yield 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 yield 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 yield 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: two models with identical f_1 scores

Modeling

Random Forest

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

Modeling

Random Forest

- grid search: two models with 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: two models with 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: two models with 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: two models with 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: two models with 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 out of grid search

Modeling

Random Forest

- grid search: two models with 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 out of grid search

XGBoost

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

Modeling

Random Forest

- grid search: two models with 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 out of grid search

XGBoost

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

Modeling

Random Forest

- grid search: two models with 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 out of grid search

XGBoost

- much faster training than for the GradientBoostingClassifier (explained later)
- equally excellent results
- optimized model \Rightarrow best overall

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
CNN			?			?	?
Bidirectional LSTM			?			?	?

- 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
CNN			?			?	?
Bidirectional LSTM			?			?	?

- 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
CNN			?			?	?
Bidirectional LSTM			?			?	?

- 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
CNN			?			?	?
Bidirectional LSTM			?			?	?

- 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 (may indicate limits of info in feature set)

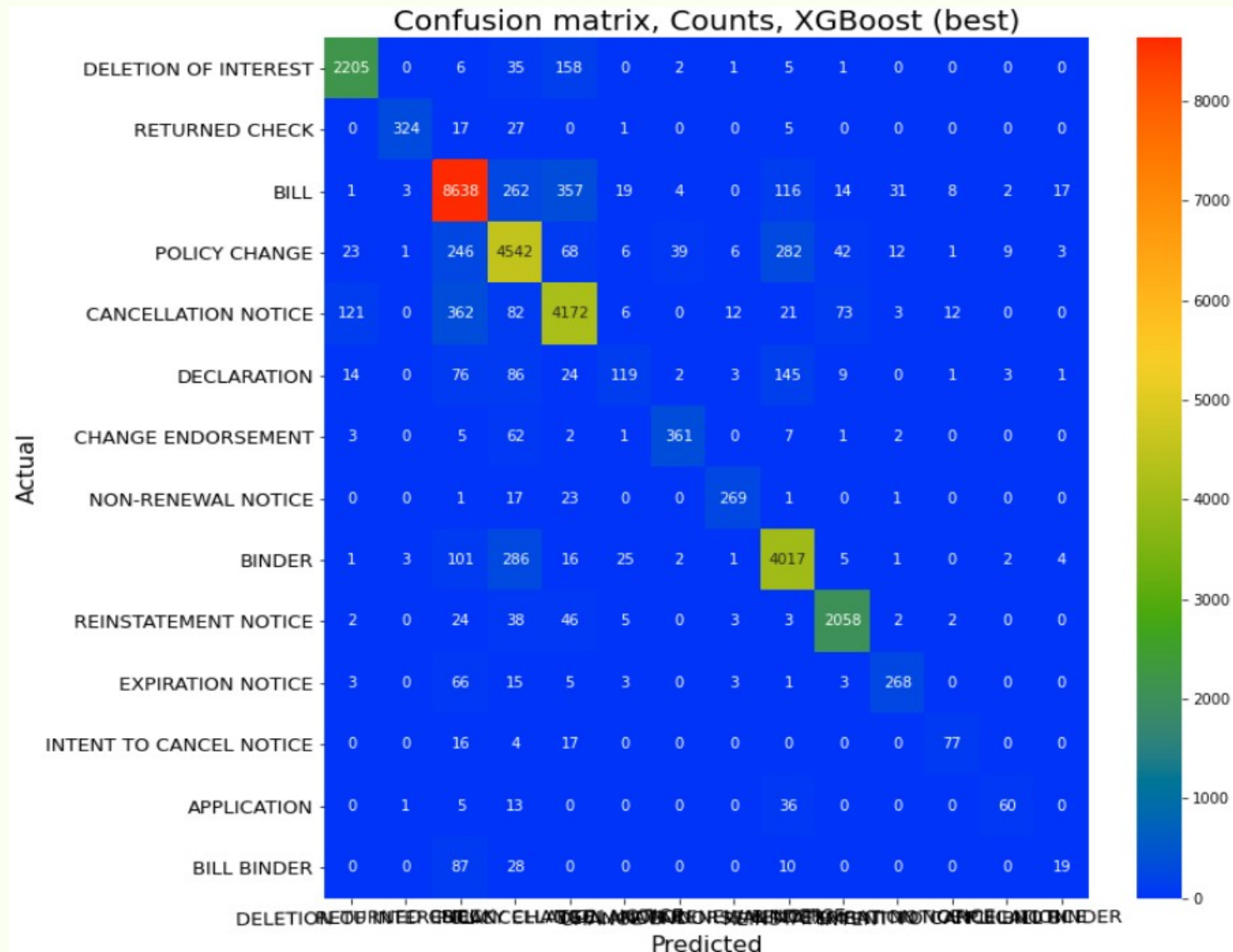
Model Results

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
CNN			?			?	?
Bidirectional LSTM			?			?	?

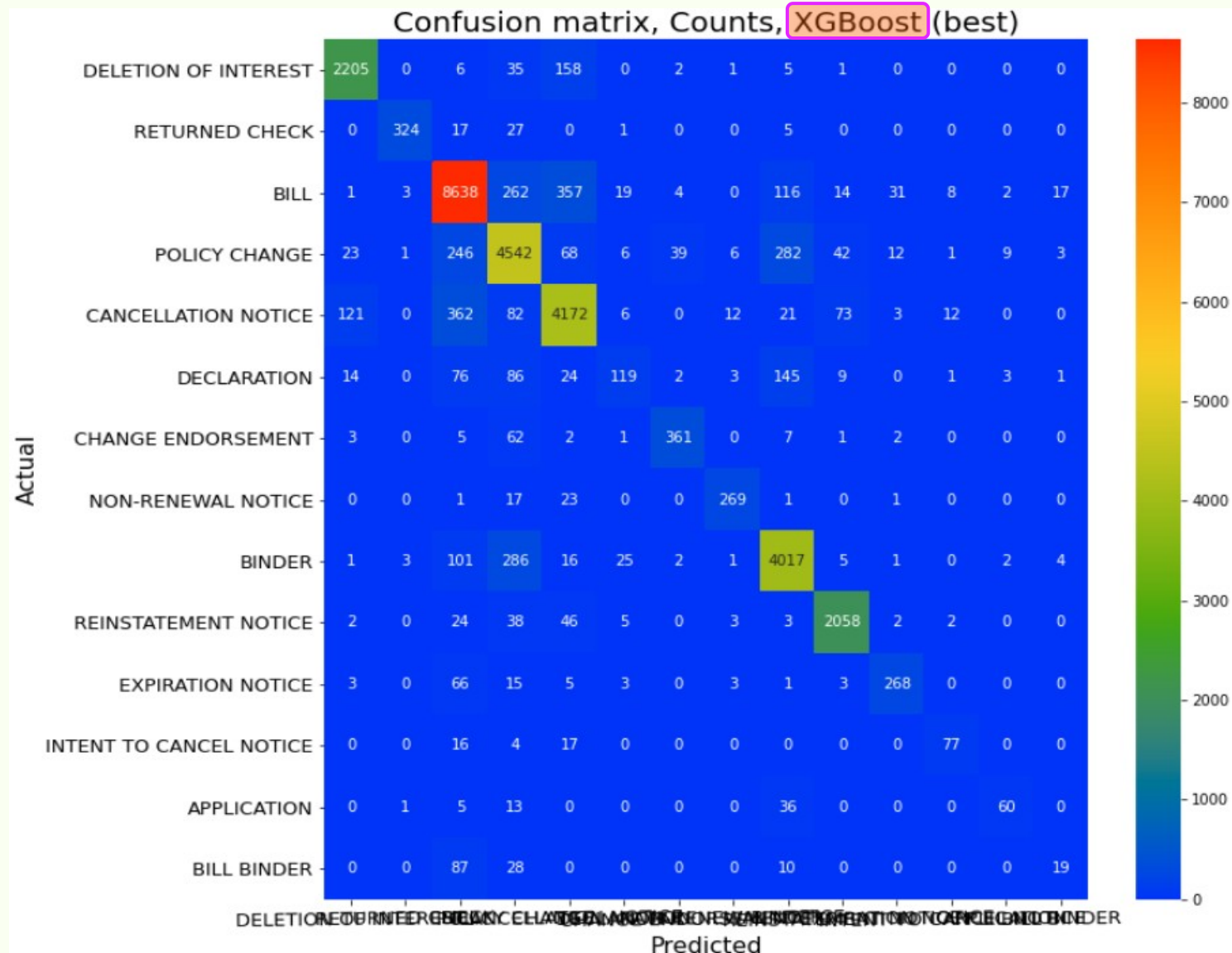
- reminder: macro averaged \Rightarrow straight average of scores for each class
weighted average \Rightarrow average of all class scores weighted by support
- if need good scores for smaller classes, focus on macro averages
- if overall results most important, focus on weighted averages

Caution: errors in small classes 0 (10%) \Rightarrow impact macro averages most

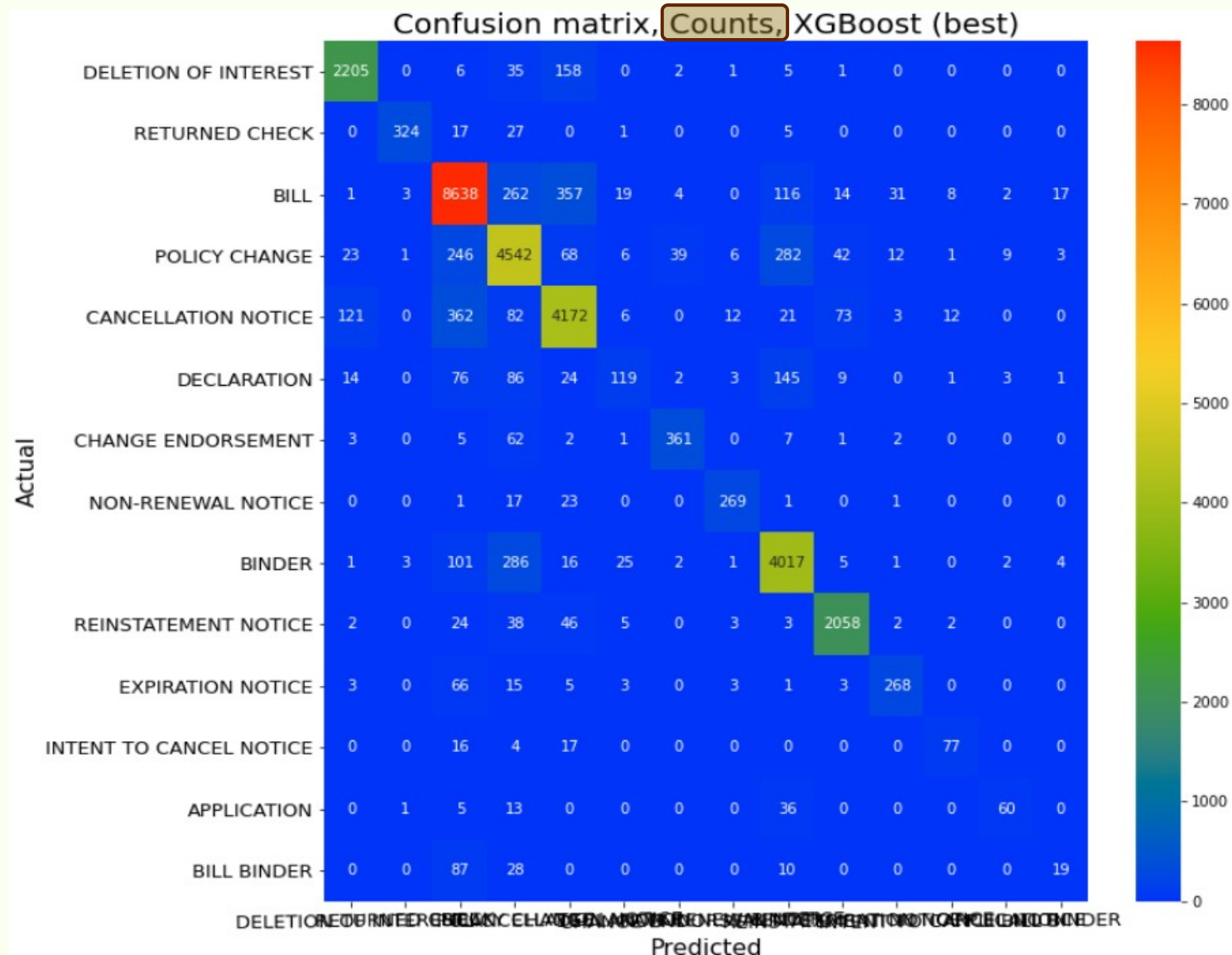
Model Results



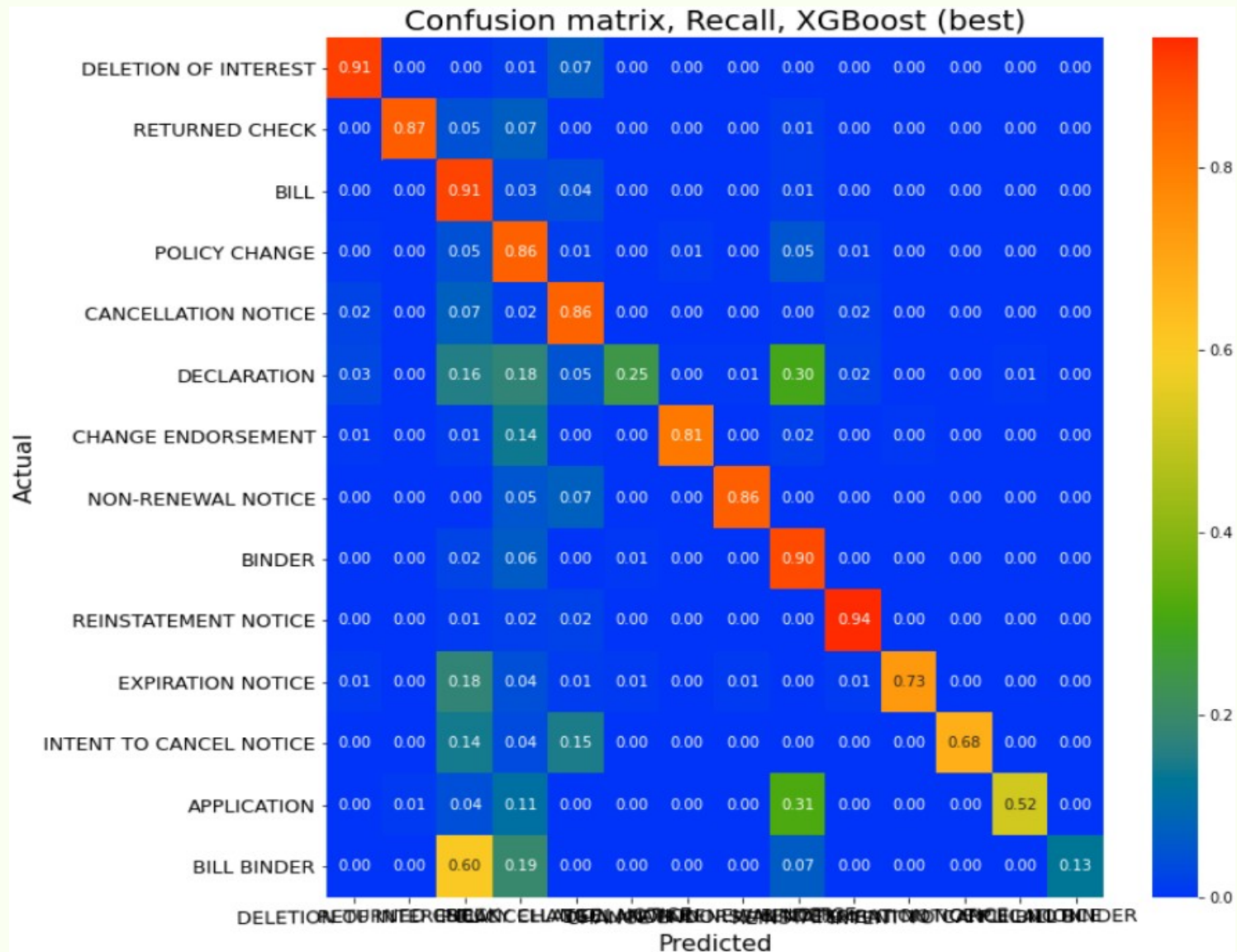
Model Results



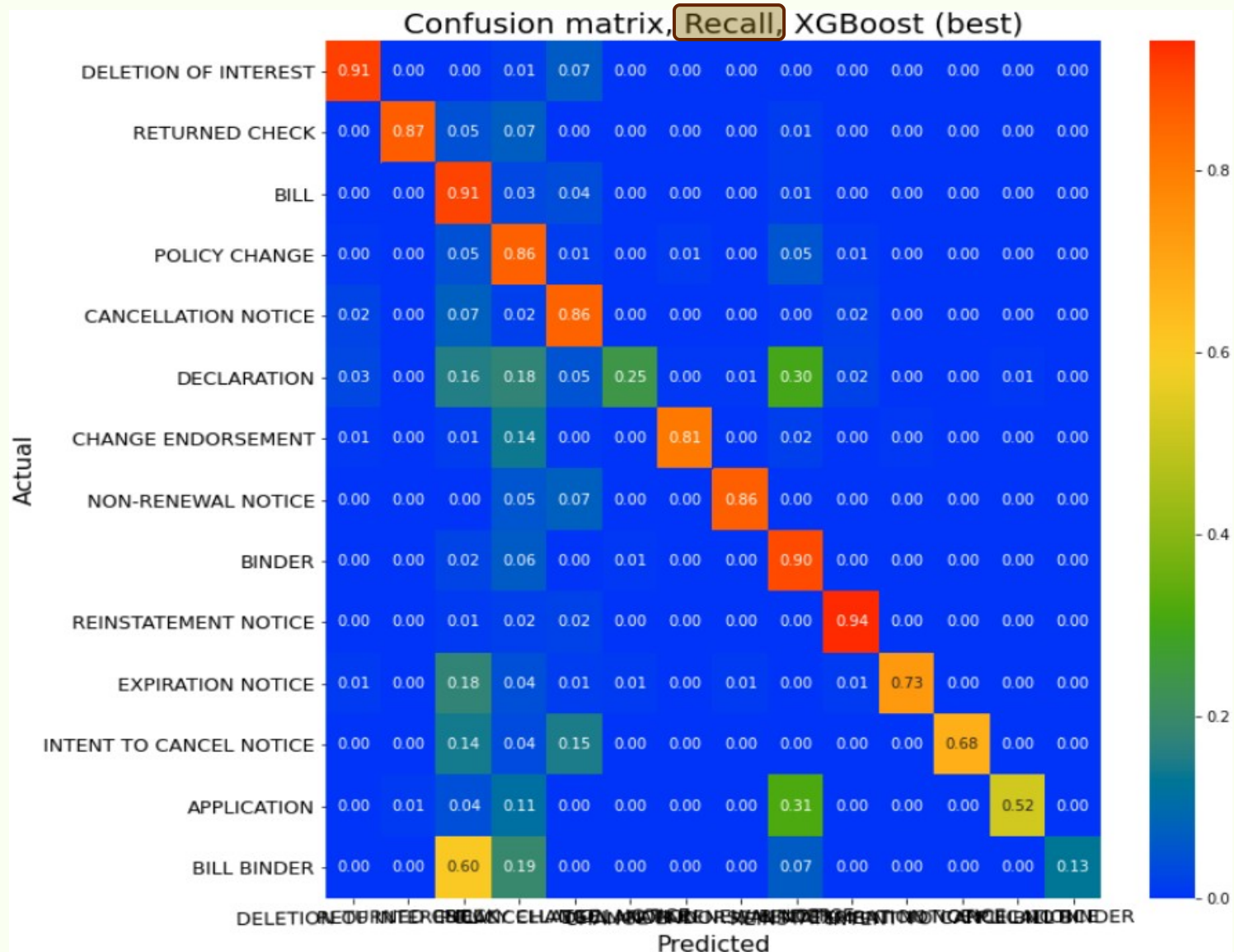
Model Results



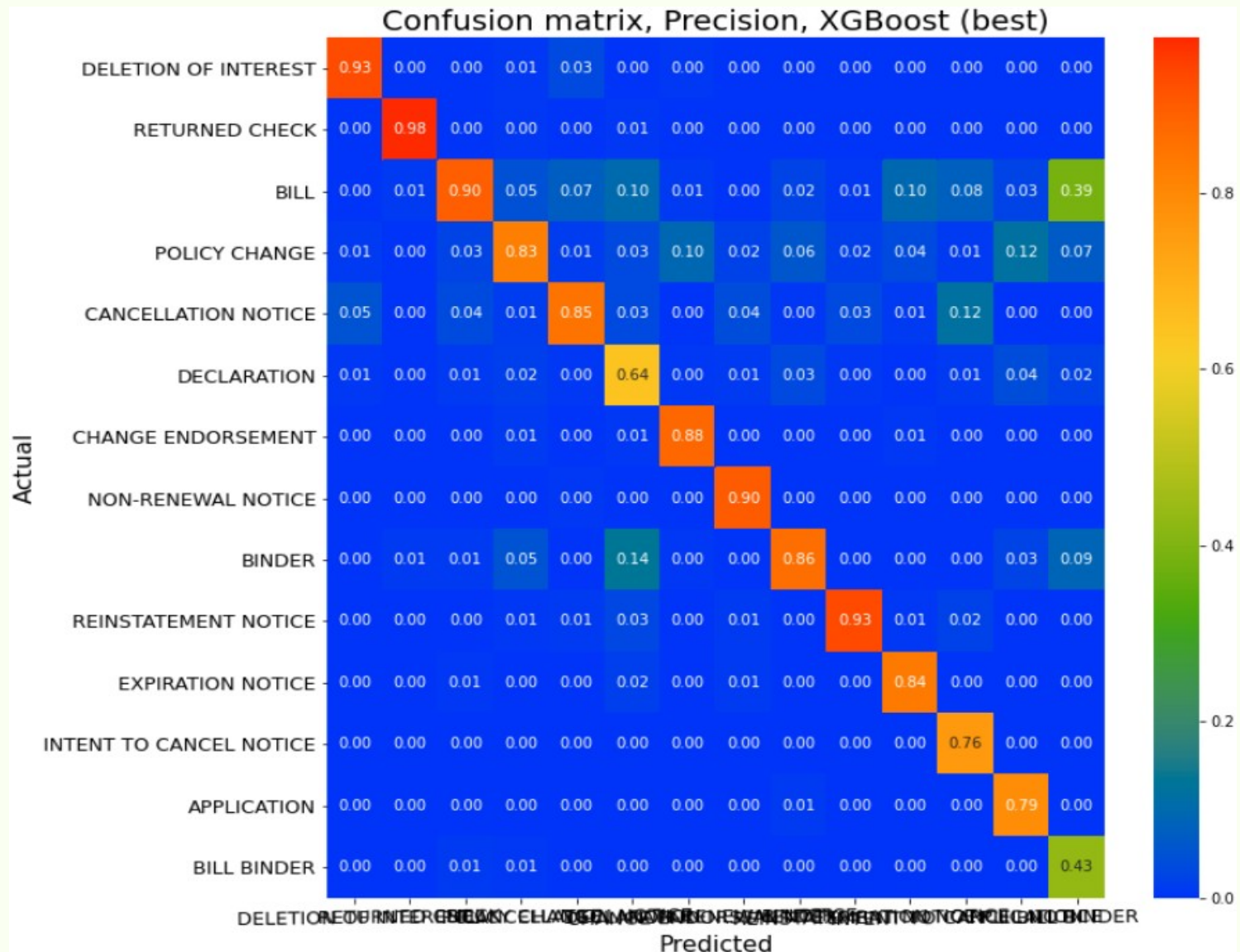
Model Results



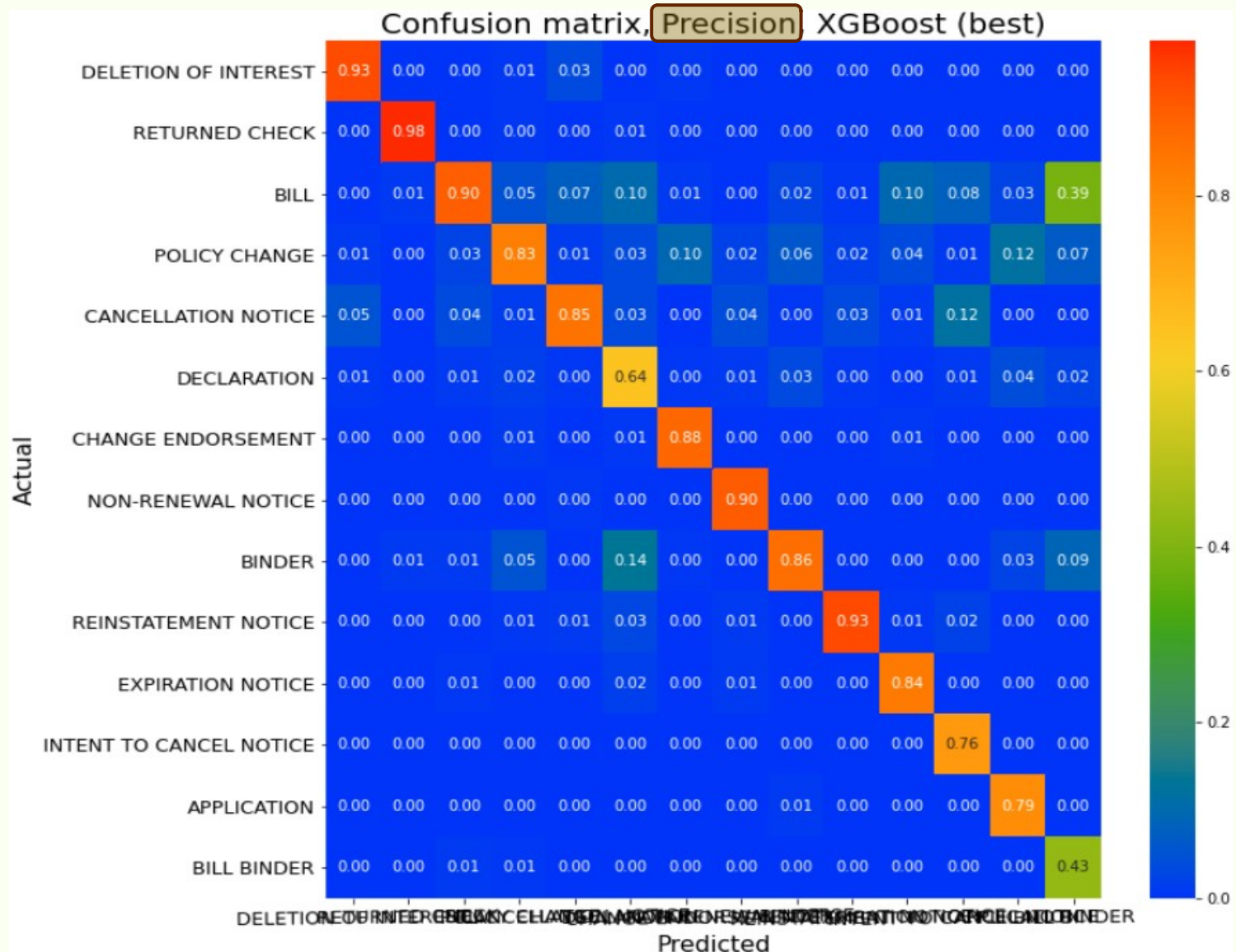
Model Results



Model Results



Model Results



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 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

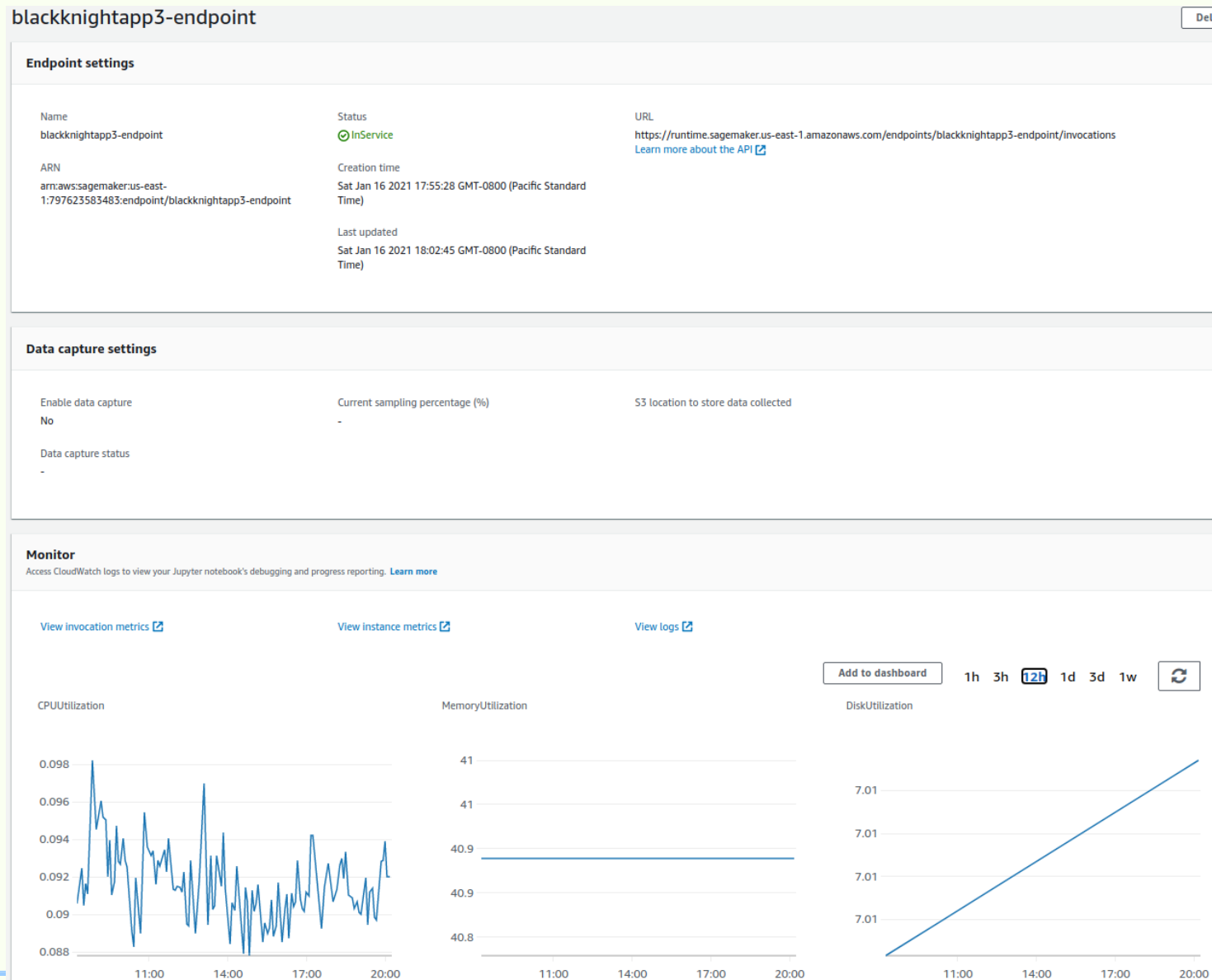
The screenshot shows the AWS IAM console interface. At the top, there are tabs for 'Permissions', 'Trust relationships', 'Tags', 'Access Advisor', and 'Revoke sessions'. The 'Permissions' tab is active. Below the tabs, it says 'Permissions policies (2 policies applied)'. There is a blue button 'Attach policies' and a link '+ Add inline policy'. A table lists the policies:

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

Below the table, there is a section for the 'SageMakerInvokeEndpoint' policy. It has tabs for 'Policy summary', '{} JSON', 'Edit policy', and a 'Simulate policy' button. The 'Policy summary' tab is active, showing the following JSON policy document:

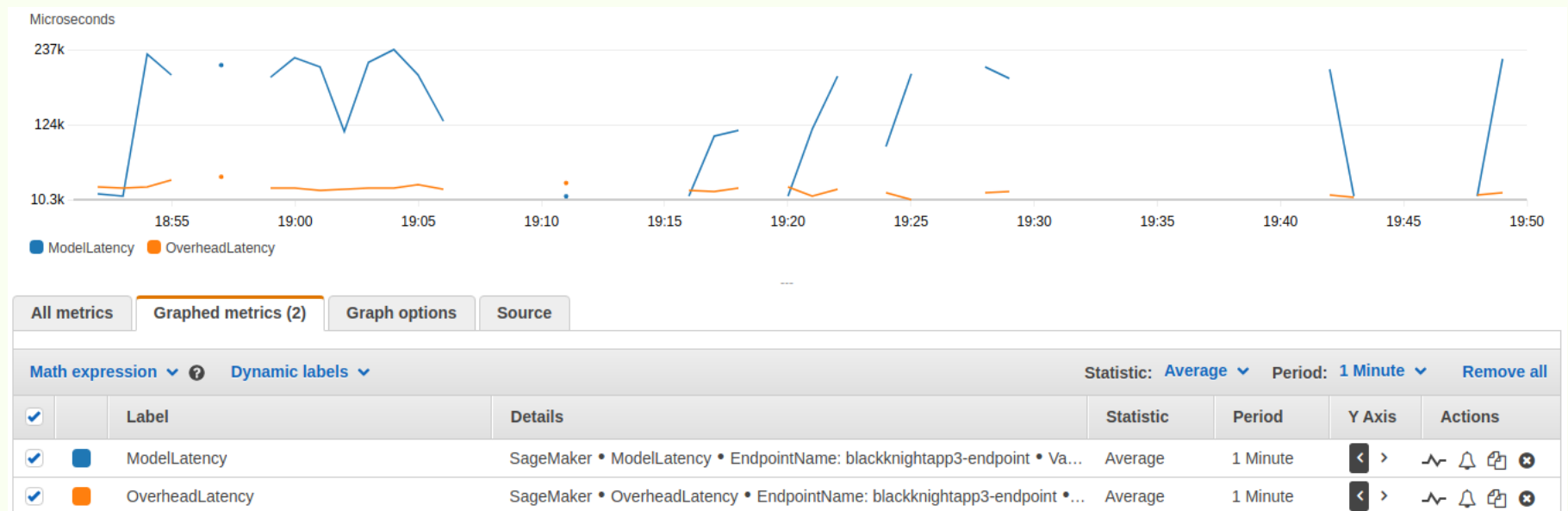
```
1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Sid": "Stmt1464440182000",
6       "Effect": "Allow",
7       "Action": [
8         "sagemaker:InvokeEndpoint"
9       ],
10      "Resource": [
11        "*"
12      ]
13    }
14  ]
15 }
```

Deployed Endpoint



Deployed Endpoint

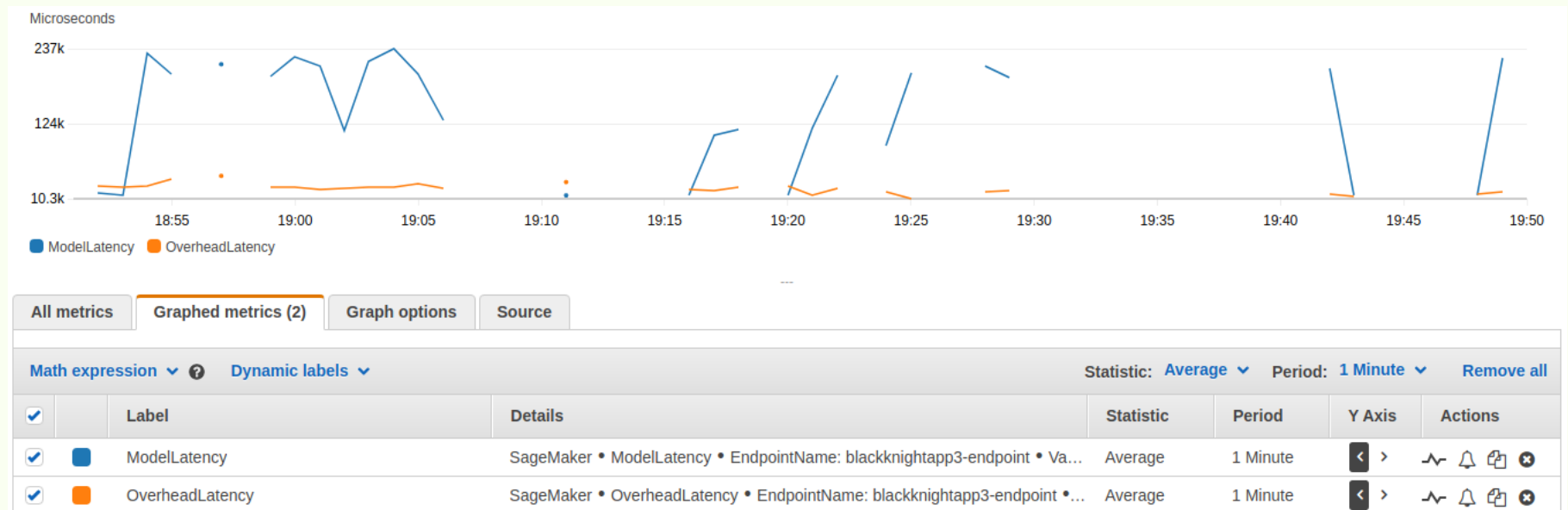
Latencies



- 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

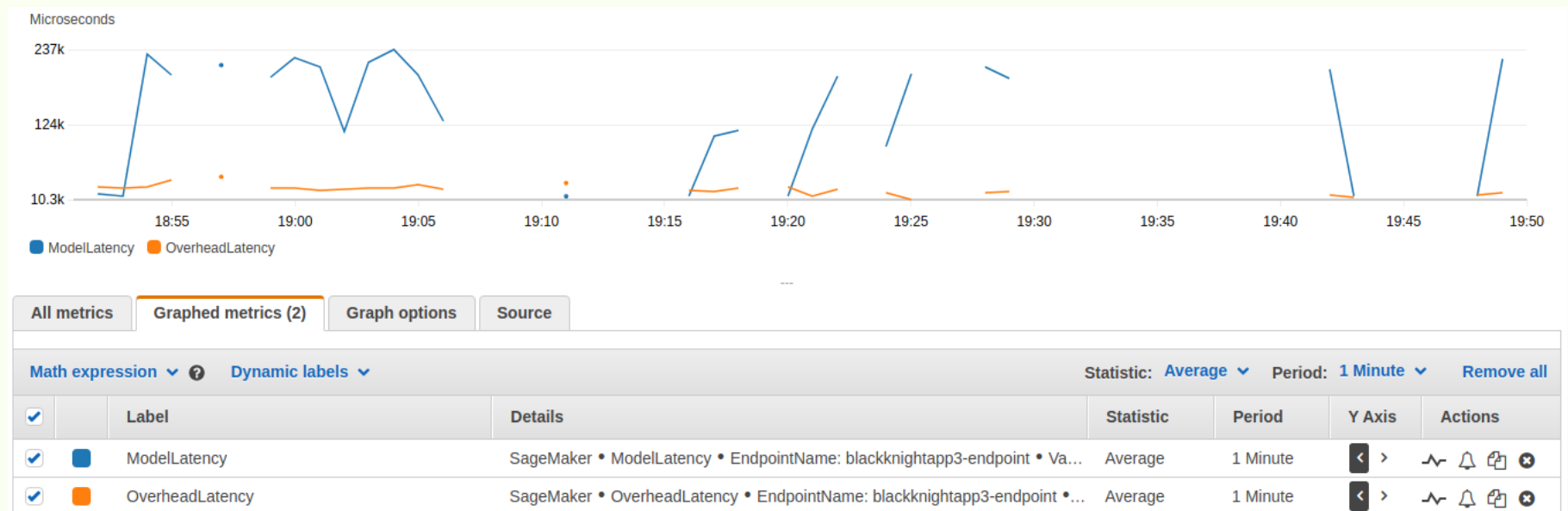
Latencies



- 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 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 shown here. The Naive Bayes model with default settings serves as a baseline, while the Random Forest model was the result of several hours of grid searching. This performed almost as well as the best XGBoost version, but the latter was trained on a GPU machine, and won't run on a standard EC2 machine. (The best Random Forest model is nearly as good.

When you mash on the 'Send' button a JSON payload is formulated and shipped to a RESTful API hosted on AWS. It used the model name to select which version to use for inference. When the returned payload is returned, this app displays the results.

For details, see [my fork](#) of HeavyWater's original github repo.

Document Classifier for Black Knight HeavyWater

blame: Mark Wilber

Replace these example documents with your own, each separated by at least 1 new line.

```
4e5019f629a9 54fb196d55ce 0cf4049f1c7c ef4ea2777c02 f8552412da3f 0a9b859f7b89 a31962fbd5f3 2bccc4e05d9d
b61f1af56200 036087ac04f9 6d25574664d2 9cdf4a63deb0 07e7fe209a3b 93c988b67c47 8a3fc46e34c1
b59e343416f7 e4ed491481ed e7f29d3843e7 87aeaddfb7f2 612cc551a793 9bc65adcc033c 6bc122aa4b06
fe3fe35491b4 6faa0d565869 f0382a00d499 f79da29e041c 3c510ffe475f 543615850429 fc462213a9d4 8f75273e5510
133d46f7ed38 2519927ae3fa 0f1d041f5921 8871e31e57e2 ce68d85c1b08 93790ade6682 9415e522bc59
4357c81e10c1 b208ae1e8232 f79da29e041c 8dedae08a79f 0cbca93be301 35991d8609e2 43af6db29054
6ca2dd348663 6b304aabdcee d38820625542 8e0a28537681 98c9498f85a3 2f58ef8b979c 932deb3b70cc
a3360a4991fa 79aa7fd11cec 133d46f7ed38 cd4c3c5e83cc 0f12fd1c2b99 31fd3123f41c 33630ee5f812 d9ef68daef4c
0cbca93be301 76b296c8d48d 0cbca93be301 c1a2676df403 f1413affa34b 3d9a3fcc2f1c 86985e33826a b73e657498f2
20127286030f 2d00e7e4d33f f4a65848d21f 6365c4563bd1 6bf9c0cb01b4 c79b01c5629d e504ee0aaf6d
8b0131ee1005 12654bbe59c7 6c998bcc2f5e b9699ce57810 a2b3b5dae4b9 17b109eb308e 0562c756a2f2
0cbca93be301 8532b14a158d b3d1c1eff9d7 10e45001c2f2 e7f10ad56136 40cb5e209b92 ba963af414e7
97b6014f9e50 b9699ce57810 6bf9c0cb01b4 3f612d66ae3b d38820625542 28bce73d3237 7c95e780d5ec
7336405c7cfa 0cbca93be301 72bd4a50cf4a 0c490affb30d f3ff20955734 7009efbdb7b1 d38820625542 e1b9e4df3a88
4ad52689d690 a024d1e04168 c337a85b8ef9 f1393c430fd1 95de2151d27c 586242498a88 8071efb3570c
90e906ce4b90 1015893e384a d037ca00fa63 22fa1184be26 9e5639a57e81 7860028b1d17 56b98f5aef76
be95012ebf2b 586242498a88 845c5d0ccbf8 61442b440484 6b343f522f78 d9ef68daef4c ec522bf7f985
ec3406979928 816aed74475e 89cce1c7ef23 43af6db29054 43af6db29054 8f75273e5510 4ad52689d690
a024d1e04168 1b6d0614f2c7 ef4ba44cdf5f 1015893e384a 1015893e384a 5748149bc6f5 ffe8decfd82e
427028e08976 9bc65adc033c 133d46f7ed38 564aaf0c408b 6defcd633808 de078996c1a5 5ff8f7117bc9
32150e5d4311 6f6fb5a7797f 48cda5a5171a 288ccf089872 56c2c356d772 3b06427e873d 9bc65adc033c
6ef2ade170d9 d9ef68daef4c e3a330c58136 2ce0277ae4e0 be95012ebf2b 2d00e7e4d33f 0cbca93be301
20d53168dbb6 86f0841bdf32 40d775b9c777 a921ec62a35d 20fecb59250 0a4e98ab9a8f d38820625542
97b6014f9e50 0562c756a2f2 8460873735c7 a1bb6b4223d9 f1413affa34b 4ad52689d690 8f7a92cd0ae7
e9be2c2489cd b208ae1e8232 a7d66c72a972 28bce73d3237 4ad52689d690 8f7a92cd0ae7 c337a85b8ef9
5ff8f7117bc9 c5c52e33fb85 3ddfcec1d334 fe3fe35491b4 97b6014f9e50 a86f2ba617ec 47439a0d8004 f55be870e57d 2009.
```

Get results!

A 15-second demo

Next steps?

- LSTM
 - [I've done this](#) with a different text classification problem ([notebook](#))

Next steps?

- LSTM
 - [I've done this](#) with a different text classification problem ([notebook](#))
- 1-D convolutional neural network

Next steps?

- LSTM
 - [I've done this](#) with a different text classification problem ([notebook](#))
- 1-D convolutional neural network
- My preferred solution (in theory!):
 - documents sentenced-tokenized

Next steps?

- LSTM
 - [I've done this](#) with a different text classification problem ([notebook](#))
- 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 a different text classification problem ([notebook](#))
- 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 a different text classification problem ([notebook](#))
- 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 a different text classification problem ([notebook](#))
- 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 a different text classification problem ([notebook](#))
- 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 a different text classification problem ([notebook](#))
- 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!