

Counterfactual evaluation of machine learning models

Michael Manapat
@mlmanapat
Stripe

Stripe

- APIs and tools for commerce and payments

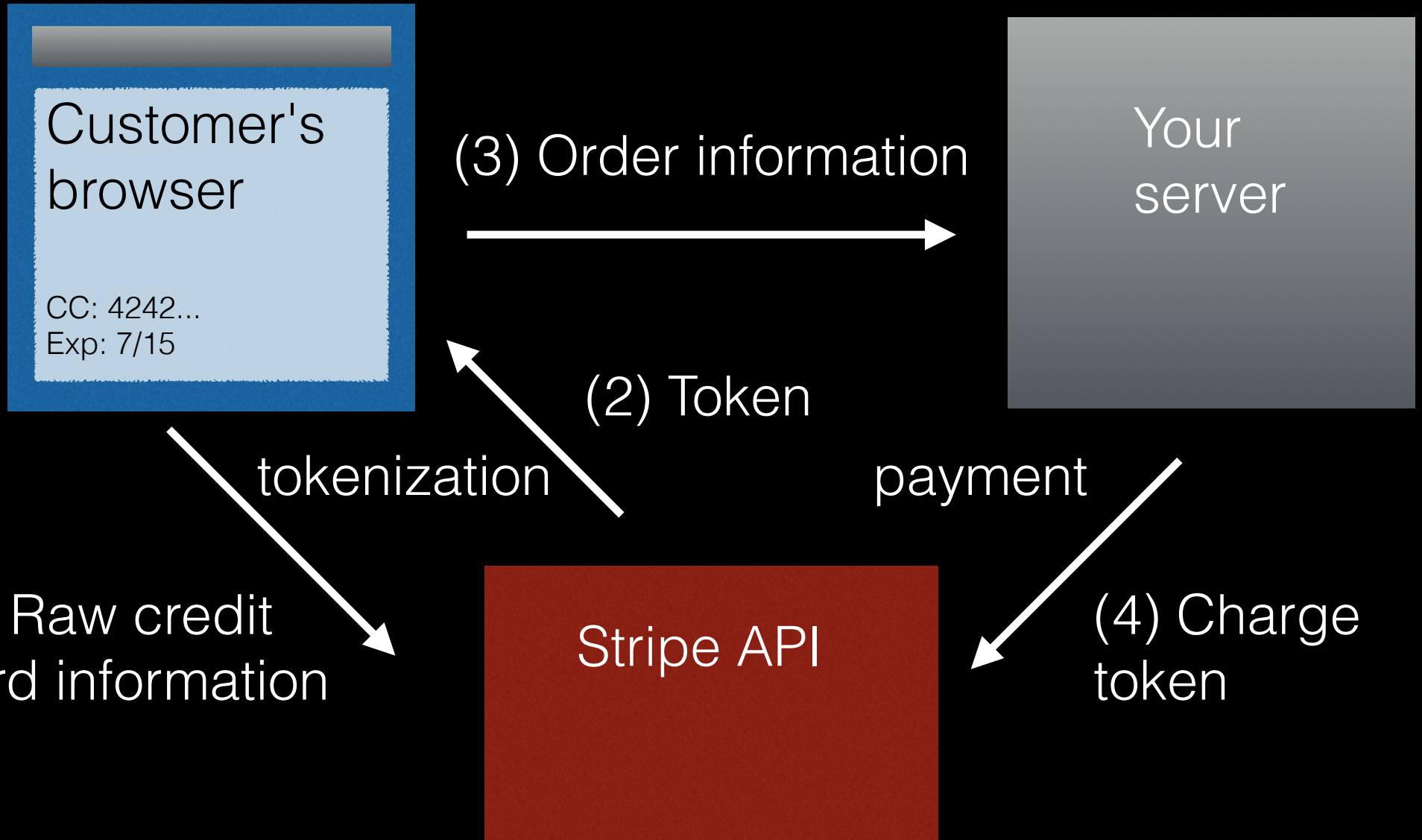
```
import stripe
stripe.api_key = "sk_test_BQokiaJOvBdiI2HdlWgHa4ds429x"

stripe.Charge.create(
    amount=40000,
    currency="usd",
    source={
        "number": '4242424242424242',
        "exp_month": 12,
        "exp_year": 2016,
        "cvc": "123"
    },
    description="PyData registration for mlm@stripe.com"
)
```

Machine Learning at Stripe

- Ten engineers/data scientists as of July 2015 (no distinction between roles)
- Focus has been on fraud detection
 - Continuous scoring of businesses for fraud risk
 - **Synchronous classification of charges as fraud or not**

Making a Charge



Charge Outcomes

- Nothing (best case)
- Refunded
- **Charged back ("disputed")**
 - Fraudulent disputes result from "card testing" and "card cashing"
- **Can take > 60 days to get raised**

JUL 5 SAFEWAY STORE 1507SAN FRANCISCO CA

Doing business as:
SAFEWAY

2020 MARKET ST
SAN FRANCISCO
CA
94114-1314
UNITED STATES

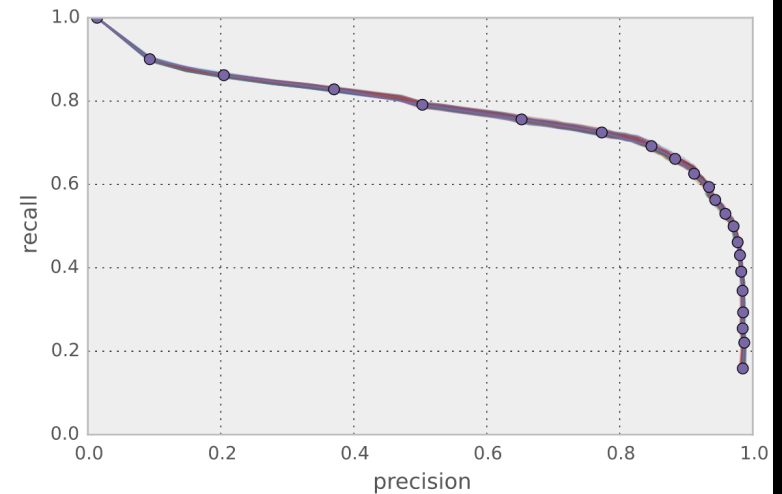
Additional Information: GROCERY STORE
GROCERY STORE
Reference: 320151870334481742
Category: Merchandise & Supplies - Groceries

[Dispute/Inquire about this Charge](#)

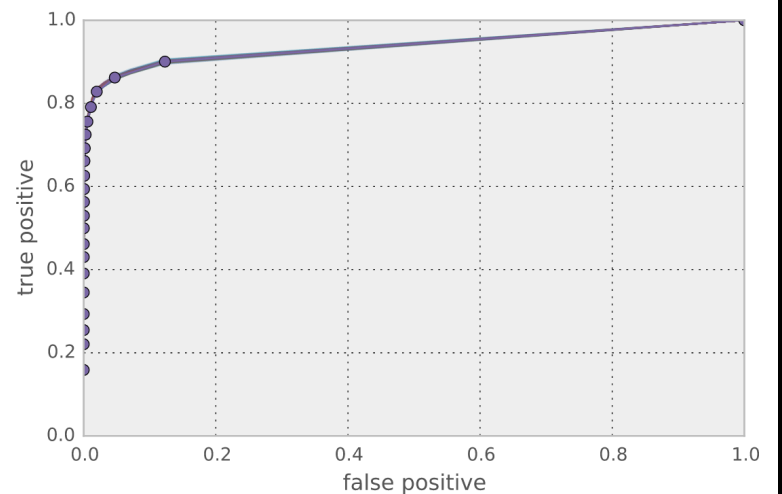
Model Building

- December 31st, 2013
 - Train a **binary classifier** for **disputes** on data from Jan 1st to Sep 30th
 - Validate on data from Oct 1st to Oct 31st (need to wait ~60 days for labels)
- Based on validation data, pick a policy for actioning scores: block if **score > 50**

Precision/Recall curve



ROC curve AUC: 0.939

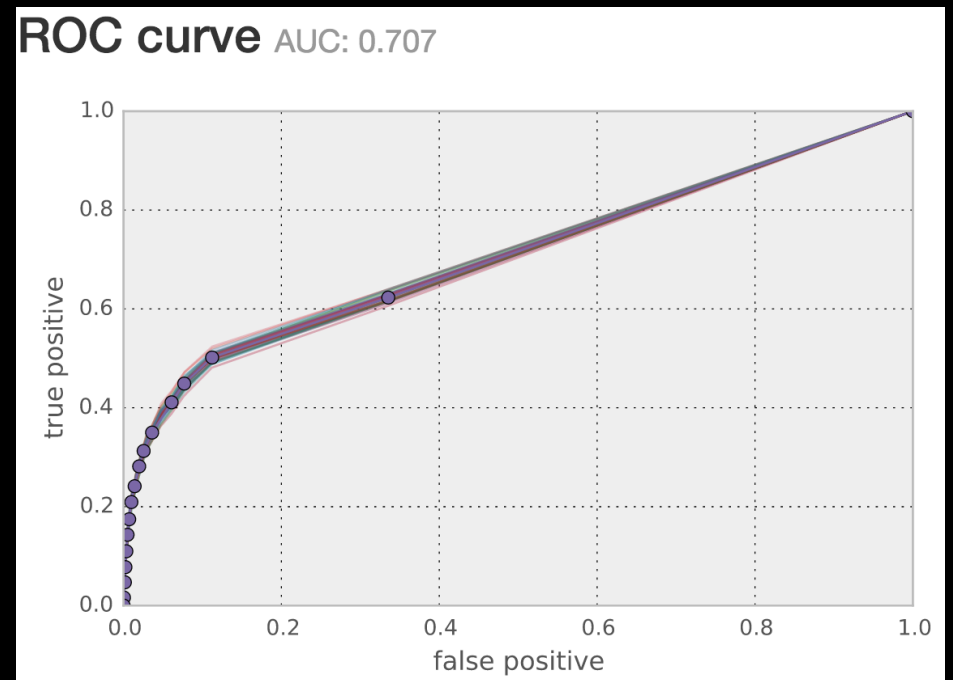


Questions

- Validation data is > 2 months old. How is the model doing?
- What are the **production** precision and recall?
- Business complains about high false positive rate: what would happen if we changed the policy to "block if **score > 70** "?

Next Iteration

- December 31st, 2014. We repeat the exercise from a year earlier
- Train a model on data from Jan 1st to Sep 30th
- Validate on data from Oct 1st to Oct 31st (need to wait ~60 days for labels)
- Validation results look ~ok (but not great)



Next Iteration

- We put the model into production, and the results are terrible
 - From spot-checking and complaints from customers, the performance is worse than even the validation data suggested
- What happened?

Next Iteration

- We already had a model running that was blocking a lot of fraud
- Training and validating only on data for which we had labels: charges that the existing model didn't catch
- Far fewer examples of fraud in number. Essentially building a model to catch the "hardest" fraud
- Possible solution: we could run both models in parallel...

Evaluation and Retraining Require the Same Data

- For **evaluation**, **policy changes**, and **retraining**, we want the same thing:
 - An approximation of the distribution of charges and outcomes that would exist in the absence of our intervention (blocking)

First attempt

- Let through some fraction of charges that we would ordinarily block

```
if score > 50:  
    if random.random() < 0.05:  
        allow()  
    else:  
        block()
```

- Straightforward to compute **precision**

Recall

1,000,000 charges	Score < 50	Score > 50
Total	900,000	100,000
Not Fraud	890,000	1,000
Fraud	10,000	4,000
Unknown	0	95,000

- Total "caught" fraud = $(4,000 * 1/0.05)$
- Total fraud = $(4,000 * 1/0.05) + 10,000$
- Recall = $80,000 / 90,000 = \mathbf{89\%}$

Training

- Train only on charges that were not blocked
- Include weights of $1/0.05 = 20$ for charges that would have been blocked if not for the random reversal

```
from sklearn.ensemble import \
RandomForestRegressor
...
r = RandomForestRegressor(n_estimators=10)
r.fit(X, Y, sample_weight=weights)
```

Training

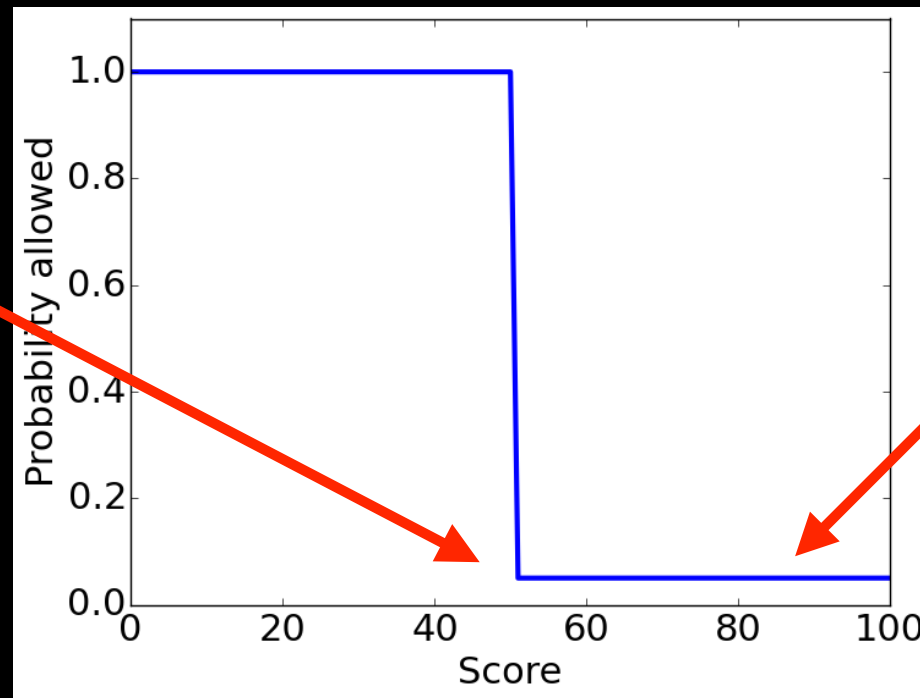
- Use weights in validation (on hold-out set) as well

```
from sklearn import cross_validation
X_train, X_test, y_train, y_test = \
    cross_validation.train_test_split(
        data, target, test_size=0.2)
r = RandomForestRegressor(...)
...
r.score(
    X_test, y_test, sample_weight=weights)
```

Better Approach

- We're letting through 5% of all charges we think are fraudulent. Policy:

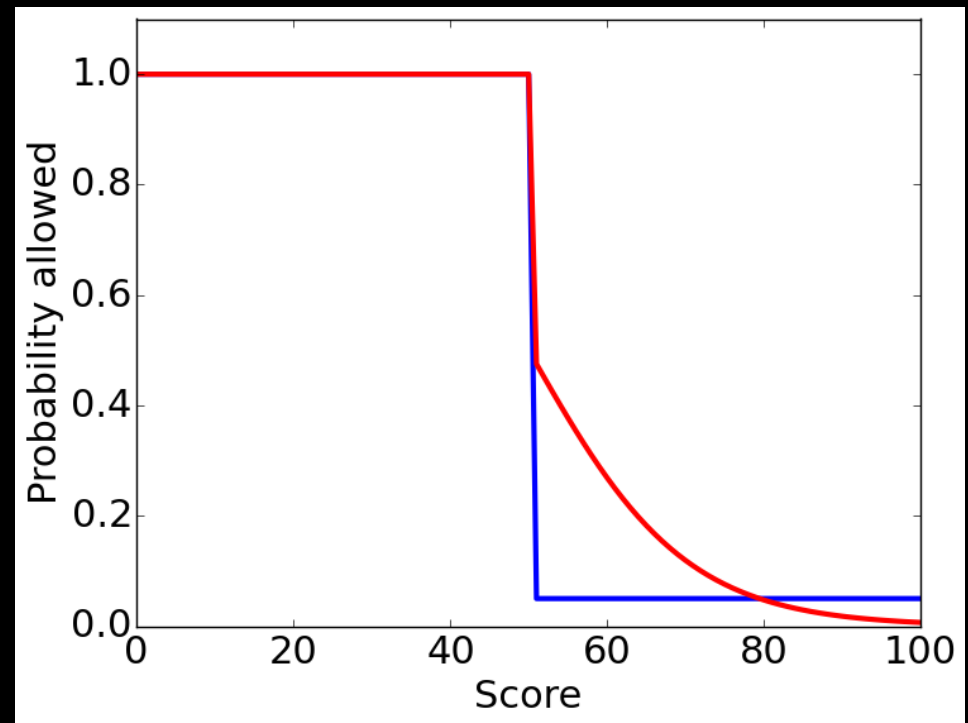
Could go either way



Very likely to be fraud

Better Approach

- **Propensity function:** maps classifier scores to $P(\text{Allow})$
- The higher the score, the lower probability we let the charge through
- Get information on the area we want to improve on
- Letting through less "obvious" fraud ("budget" for evaluation)



Better Approach

```
def propensity(score):  
    # Piecewise linear/sigmoidal  
    ...  
    ps = propensity(score)  
    original_block = score > 50  
    selected_block = random.random() < ps  
    if selected_block:  
        block()  
    else:  
        allow()  
    log_record(  
        id, score, ps, original_block,  
        selected_block)
```

ID	Score	p(Allow)	Original Action	Selected Action	Outcome
1	10	1.0	Allow	Allow	OK
2	45	1.0	Allow	Allow	Fraud
3	55	0.30	Block	Block	-
4	65	0.20	Block	Allow	Fraud
5	100	0.0005	Block	Block	-
6	60	0.25	Block	Allow	OK

Analysis

- In any analysis, we only consider samples that were **allowed** (since we don't have labels otherwise)
- We weight each sample by **$1 / P(\text{Allow})$**
 - cf. weighting by $1/0.05 = 20$ in the uniform probability case

ID	Score	P(Allow)	Weight	Original Action	Selected Action	Outcome
1	10	1.0	1	Allow	Allow	OK
2	45	1.0	1	Allow	Allow	Fraud
4	65	0.20	5	Block	Allow	Fraud
6	60	0.25	4	Block	Allow	OK

Evaluating the "block if **score** > **50**" policy

Precision = $5 / 9 = \mathbf{0.56}$

Recall = $5 / 6 = \mathbf{0.83}$

ID	Score	P(Allow)	Weight	Original Action	Selected Action	Outcome
1	10	1.0	1	Allow	Allow	OK
2	45	1.0	1	Allow	Allow	Fraud
4	65	0.20	5	Block	Allow	Fraud
6	60	0.25	4	Block	Allow	OK

Evaluating the "block if **score** > **40**" policy

Precision = 6 / 10 = **0.60**

Recall = 6 / 6 = **1.00**

ID	Score	P(Allow)	Weight	Original Action	Selected Action	Outcome
1	10	1.0	1	Allow	Allow	OK
2	45	1.0	1	Allow	Allow	Fraud
4	65	0.20	5	Block	Allow	Fraud
6	60	0.25	4	Block	Allow	OK

Evaluating the "block if **score** > **62**" policy

Precision = $5 / 5 = \mathbf{1.00}$

Recall = $5 / 6 = \mathbf{0.83}$

Analysis

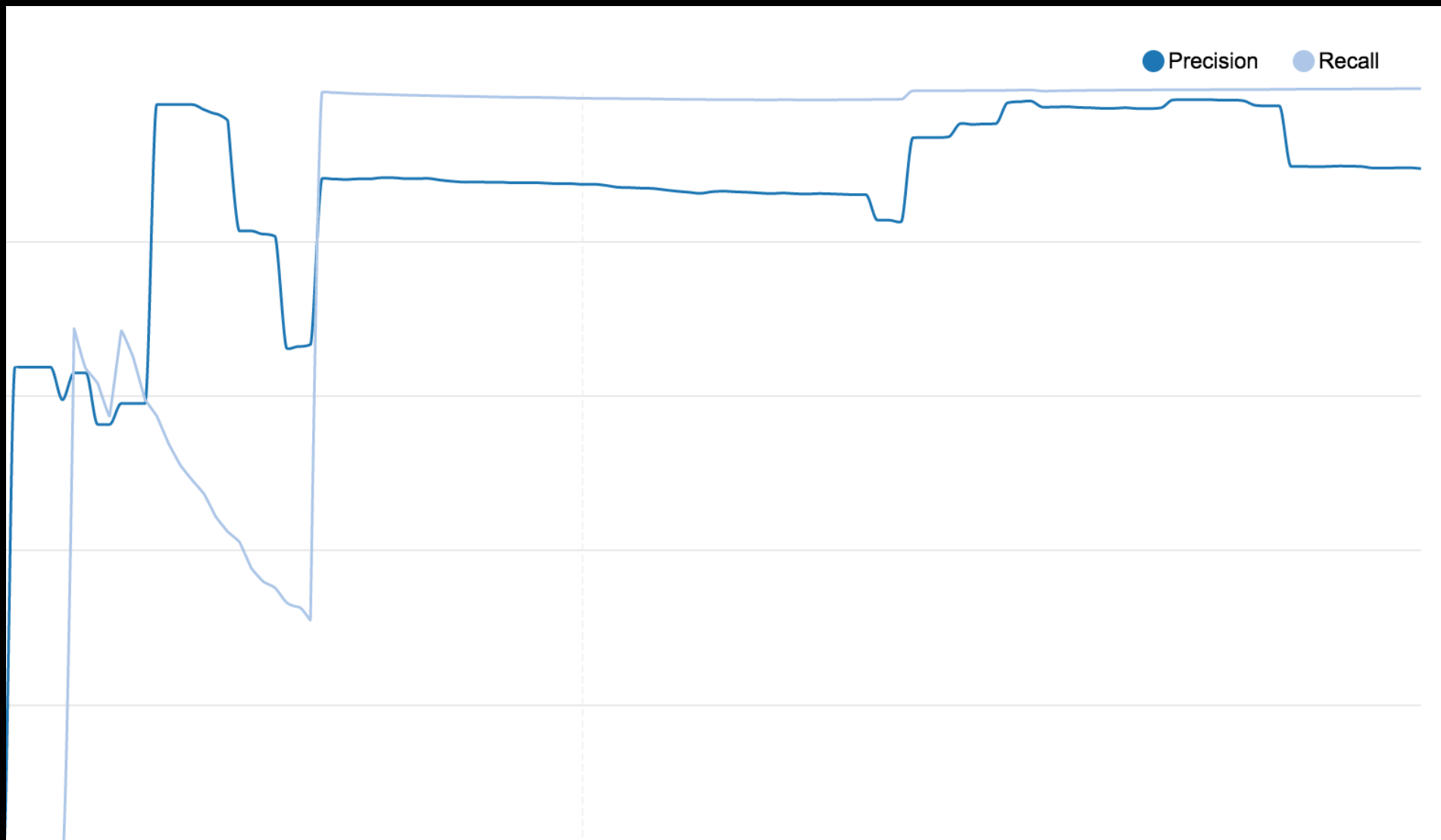
- The propensity function controls the "exploration/exploitation" payoff
- Precision, recall, etc. are estimators
 - Variance of the estimators decreases the more we allow through
- Bootstrap to get error bars (pick rows from the table uniformly at random with replacement)

New models

- Train on weighted data
 - Weights => approximate distribution without any blocking
- Can test arbitrarily many models and policies offline
- Li, Chen, Kleban, Gupta: "Counterfactual Estimation and Optimization of Click Metrics for Search Engines"

Technicalities

- Independence and random seeds
- Application-specific issues



Conclusion

- If some policy is actioning model scores, you can inject randomness in production to understand the counterfactual
- Instead of a "champion/challenger" A/B test, you can evaluate arbitrarily many models and policies in this framework

Thanks!

- Work by Ryan Wang (@ryw90), Roban Kramer (@robanhk), and Alyssa Frazee (@acfrazee)



- We're hiring engineers/data scientists!
- mlm@stripe.com (@mlmanapat)