Startup Machine Learning: Bootstrapping a fraud detection system

Michael Manapat
Stripe
@mlmanapat

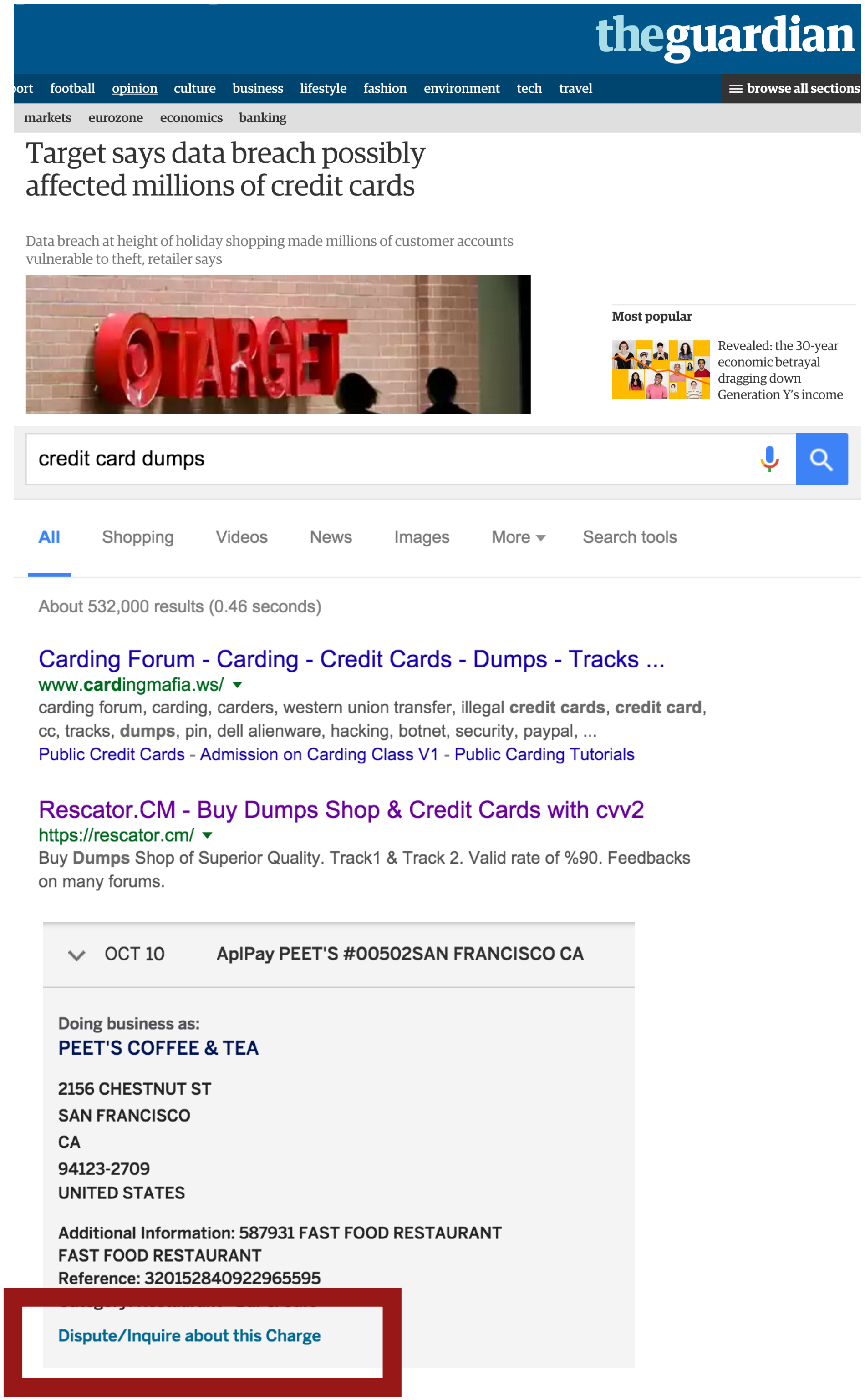
- About me: Engineering Manager of the Machine Learning Products Team at Stripe

- About Stripe: Payments infrastructure for the internet

```
curl https://api.stripe.com/v1/charges \
  -u sk_test_BQokikJOvBiI2HlWgH4olfQ2: \
  -d amount=400 \
  -d currency=usd \
  -d "description=Charge for test@example.com" \
  -d "source[object]=card" \
  -d "source[number]=4242424242424242" \
  -d "source[exp_month]=12" \
  -d "source[exp_year]=2016" \
  -d "source[cvc]=123"
```

Fraud

- Card numbers are stolen by hacking, malware, etc.
- “Dumps” are sold in “carding” forums
- Fraudsters use numbers in dumps to buy goods, which they then resell
- Cardholders dispute transactions
- Merchant ends up bearing cost of fraud



Machine Learning

- We want to detect fraud in real-time
- Imagine we had a black box “classifier” which we fed all the properties we have for a transaction (e.g., amount)
- The black box responds with the probability that the transaction is fraudulent
- We use the black box elsewhere in our system: e.g., Stripe’s API will query it for every transaction and immediately declines a charge if the probability of fraud is high enough

Input data

```
fraudulent,charge_time,amount,card_country,card_use_24h
False,2015-12-31T23:59:59Z,20484,US,0
False,2015-12-31T23:59:59Z,1211,US,0
False,2015-12-31T23:59:59Z,8396,US,1
False,2015-12-31T23:59:59Z,2359,US,0
False,2015-12-31T23:59:59Z,1480,US,3
False,2015-12-31T23:59:59Z,535,US,3
False,2015-12-31T23:59:59Z,1632,US,0
False,2015-12-31T23:59:59Z,10305,US,1
False,2015-12-31T23:59:59Z,2783,US,0
False,2015-12-31T23:59:59Z,939,US,0
```

Choosing the “features” (feature engineering) is a hard problem that we won’t cover here

First attempt

$$\text{Probability(fraud)} = a \times \text{amount} + b \times \text{card\_use\_24h} + \cdots + Z$$

Two issues:

- Probability(fraud) needs to be between 0 and 1
- card_country is not numerical (it's "categorical")

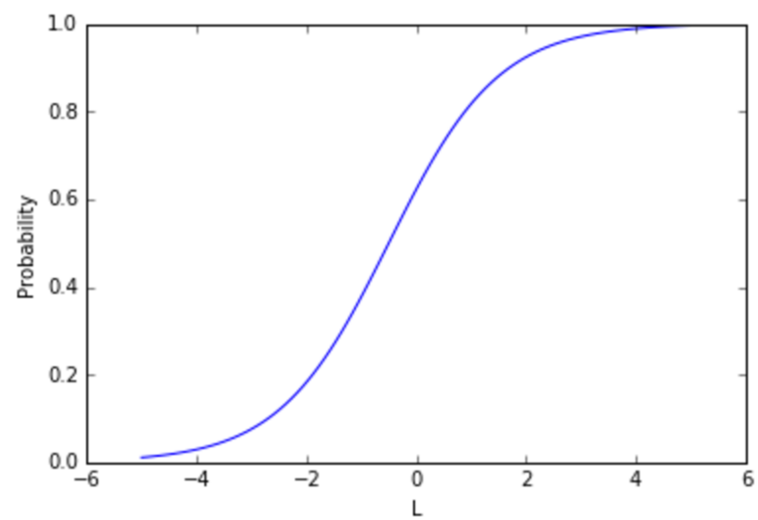
Logistic regression

- Instead of modeling p = Probability(fraud) as a linear function, we model the log-odds of fraud

$$\log\left(\frac{p}{1-p}\right) = a \times \text{amount} + b \times \text{card\_use\_24h} + \cdots + Z$$

- p is a sigmoidal function of the right side

$$p = \frac{\exp(a \times \text{amount} + b \times \text{card\_use\_24h} + \cdots + Z)}{1 + \exp(a \times \text{amount} + b \times \text{card\_use\_24h} + \cdots + Z)}$$



Categorical variables

- If we have a variable that takes one of N discrete values, we “encode” that by adding N - 1 “dummy” variables
- Ex: Let’s say card_country can be “AU,” “GB,” or “US.” We add booleans for “card = AU” and “card = GB”
- We don’t want a linear relationship among variables

Our final model is

$$\log\left(\frac{p}{1-p}\right) = a \times \text{amount} + b \times \text{card\_use\_24h} + c \times (\text{country} = \text{AU}) + d \times (\text{country} = \text{GB}) + Z$$

Fitting a regression

$$\log\left(\frac{p}{1-p}\right) = a \times \text{amount} + b \times \text{card\_use\_24h} +$$
$$c \times (\text{country} = \text{AU}) + d \times (\text{country} = \text{GB}) + Z$$

- Guess values for *a*, *b*, *c*, *d*, and *Z*
- Compute the “likelihood” of the training observations given these values for the parameters

$$\ell(a,b,c,d,Z) = \prod_{\text{fraud}} p(x_i) \prod_{\text{not fraud}} (1 - p(x_j))$$

- Find *a*, *b*, *c*, *d*, and *Z* that maximize likelihood (optimization problem—gradient descent)

```python
import pandas as pd
```

```python
data = pd.read_csv('data.csv')
```

```python
data.head()
```

	fraudulent	charge_time	amount	card_country	card_use_24h
0	False	2015-12-31T23:59:59Z	20484	US	0
1	False	2015-12-31T23:59:59Z	1211	US	0
2	False	2015-12-31T23:59:59Z	8396	US	1
3	False	2015-12-31T23:59:59Z	2359	US	0
4	False	2015-12-31T23:59:59Z	1480	US	3

```python
data.fraudulent.value_counts()
```

```
False    45174
True     44219
Name: fraudulent, dtype: int64
```

```python
data.card_country.value_counts()
```

```
US    84494
GB     2754
AU     2145
Name: card_country, dtype: int64
```

pandas brings R-like data frames to Python

```
encoded_countries = pd.get_dummies(data.card_country, prefix='cc_')
```

```
encoded_countries.head()
```

	cc__AU	cc__GB	cc__US
0	0	0	1
1	0	0	1
2	0	0	1
3	0	0	1
4	0	0	1

```
data = data.join(encoded_countries)
```

```
data.head()
```

	fraudulent	charge_time	amount	card_country	card_use_24h	cc__AU	cc__GB	cc__US
0	False	2015-12-31T23:59:59Z	20484	US	0	0	0	1
1	False	2015-12-31T23:59:59Z	1211	US	0	0	0	1
2	False	2015-12-31T23:59:59Z	8396	US	1	0	0	1
3	False	2015-12-31T23:59:59Z	2359	US	0	0	0	1
4	False	2015-12-31T23:59:59Z	1480	US	3	0	0	1

```
y = data.fraudulent
```

```
X = data[['amount', 'card_use_24h', 'cc__AU', 'cc__GB']]
```

```python
from sklearn.cross_validation import train_test_split
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)
```

- We want models to generalize well, i.e., to give accurate predictions on new data

- We don't want to "overfit" to randomness in the data we use to train the model, so we evaluate our performance on data not used to generate the model

```python
# Logistic Regression
from sklearn.linear_model import LogisticRegression
```

```python
lr_model = LogisticRegression().fit(X_train, y_train)
```

```python
lr_model.coef_
```

```
array([[  4.62586221e-06,   3.53495554e-02,   4.28936114e-03,
          2.49802503e-03]])
```

```python
lr_model.intercept_
```

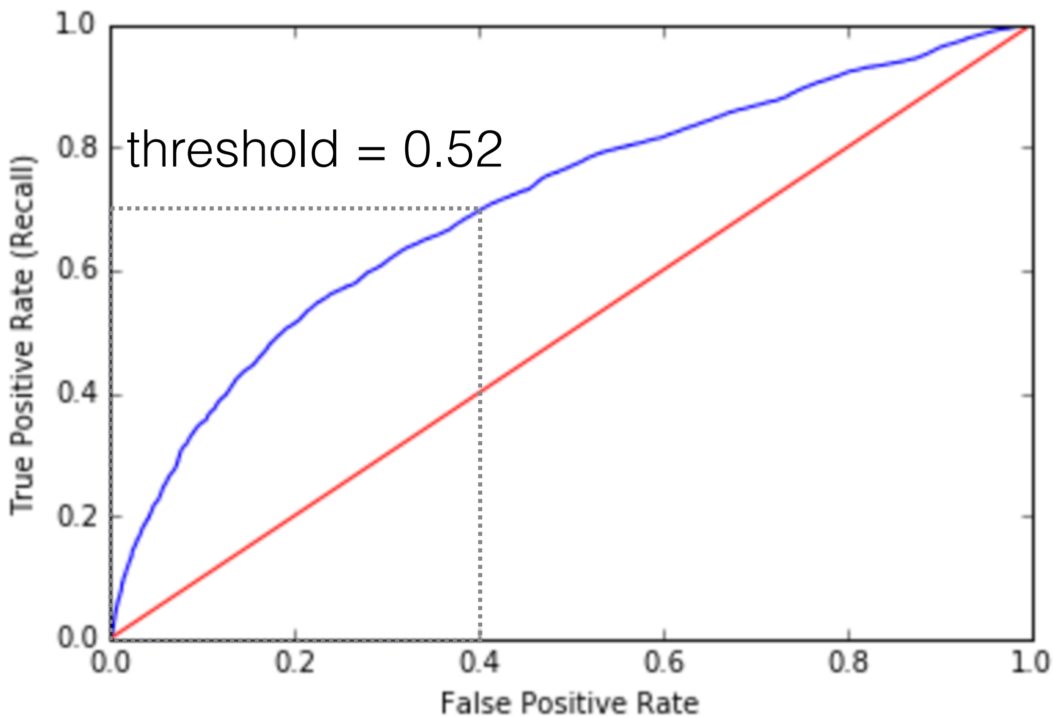
```
array([-0.0157345])
```

$$\log\left(\frac{p}{1-p}\right) = 4.63 \times 10^{-6} \times \text{amount} + 0.035 \times \text{card\_use\_24h} +$$
$$0.0043 \times (\text{cc\_AU} = 1) + 0.0025 \times (\text{cc\_GB} = 1) - 0.016$$

Evaluating the model - ROC, AUC

```
from sklearn.metrics import roc_curve, roc_auc_score
```

```
fpr, tpr, thresholds = roc_curve(y_test, y_test_scores_lr)
```



FPR = fraction of non-fraud predicted to be fraud

TPR = fraction of fraud predicted to be fraud

```
roc_auc_score(y_test, y_test_scores_lr)
```

```
0.70615874763542874
```

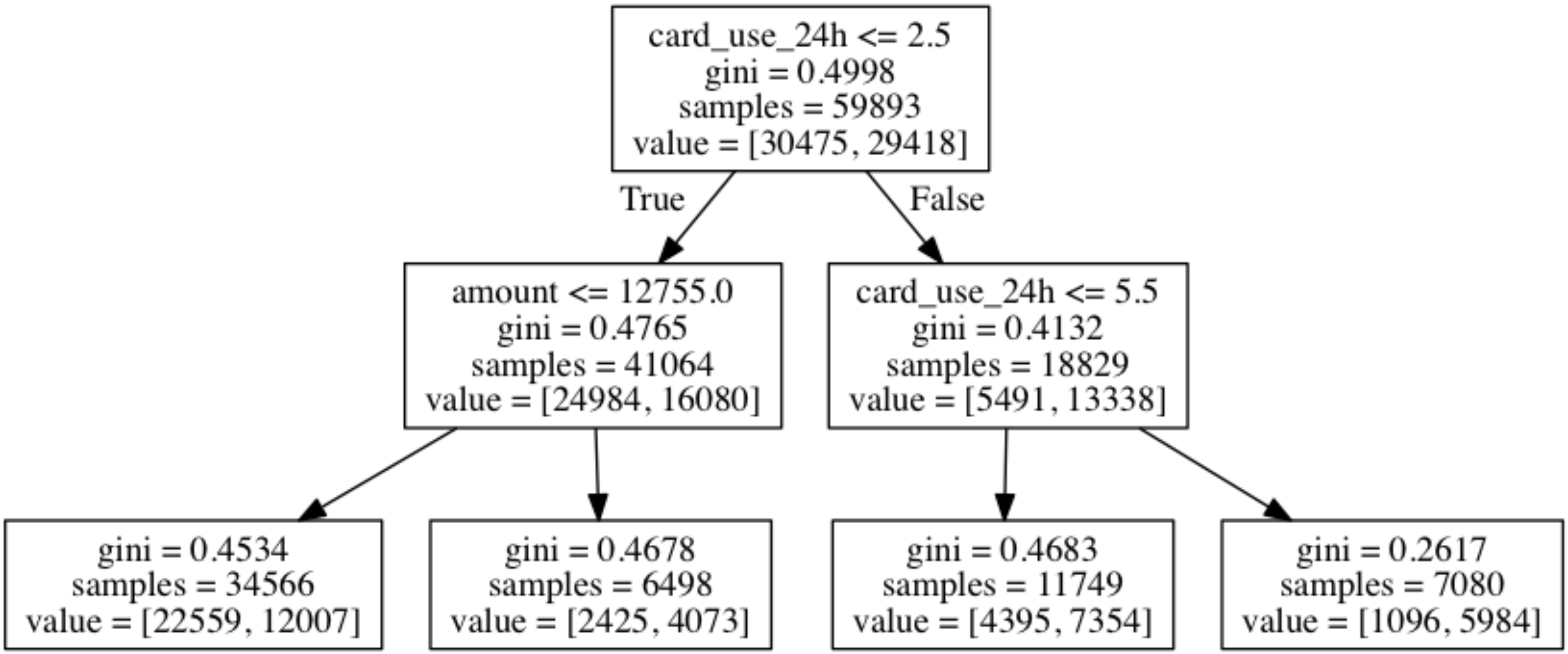
Nonlinear models

- (Logistic) regressions are **linear** models: if you double one input value, the log-odds also double

- What if the impact of amount depends on another variable? For example, maybe larger amounts are more predictive of fraud for GB cards.*

- What if the effect of amount is nonlinear? For example, maybe small and large charges are more likely to be fraudulent than charges with moderate amounts.

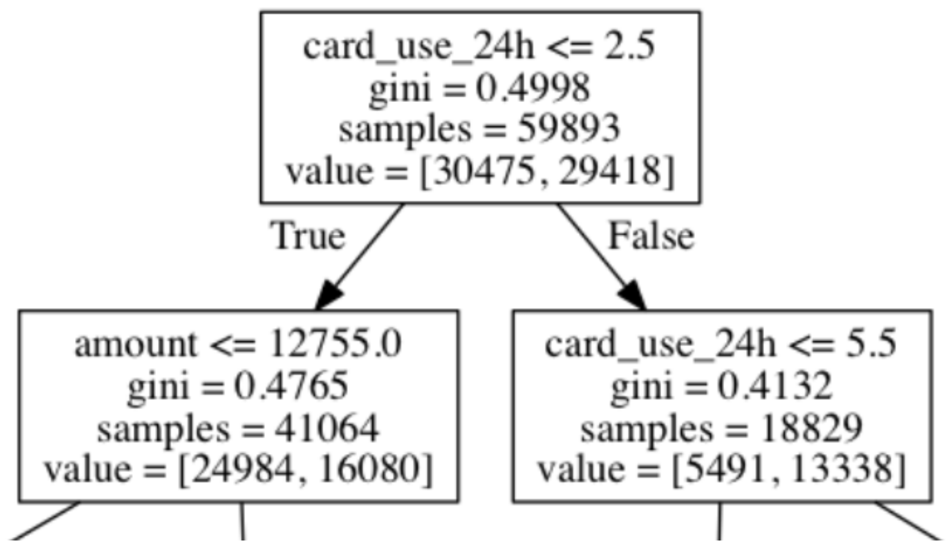
Decision Trees



p = 0.34 p = 0.63 p = 0.63 p = 0.85

Fitting a decision tree

- Start with a node (first node is all the data)
- Pick the split that maximizes the decrease in Gini (weighted by size of child nodes) $2p(1-p)$
- Example gain:
  (0.4998) - (
  (41064/59893) * 0.4765 +
  (18829/59893) * 0.4132)
  = 0.043
- Continue recursively until stopping criterion reached



Random forests

- Decision trees are “easy” to overfit
- We train N trees, each on a (bootstrapped) sample of the training data
- At each split, we only consider a subset of the available features—say, sqrt(total # of features) of them
- This reduces correlation among the trees
- The score is the average of the score produced by each tree

```python
# Decision Tree
from sklearn.tree import DecisionTreeClassifier
```

```python
dt_model = DecisionTreeClassifier(
    max_depth=3, min_samples_split=20).fit(X_train, y_train)
```

```python
y_test_scores_dt = [x[1] for x in dt_model.predict_proba(X_test)]
```

```python
roc_auc_score(y_test, y_test_scores_dt)
```

```
0.6928942419967035
```

```python
# Random Forest
from sklearn.ensemble import RandomForestClassifier
```

```python
rf_model = RandomForestClassifier(
    n_estimators=100, min_samples_leaf=100).fit(X_train, y_train)
```

```python
y_test_scores_rf = [x[1] for x in rf_model.predict_proba(X_test)]
```

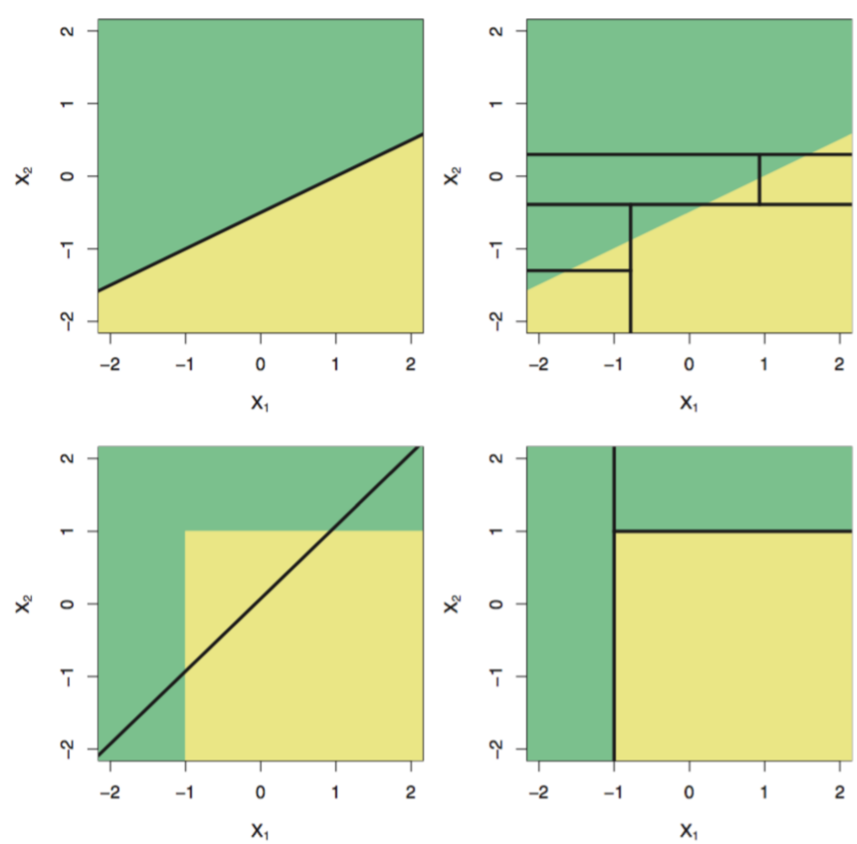
```python
roc_auc_score(y_test, y_test_scores_rf)
```

```
0.7361136032908384
```

Choosing methods

- **Use regression if**: the relationship between the target and the inputs is linear, or you want to be able to isolate the impact of each variable on the target
- **Use a tree/forest if**: there are complex dependencies between inputs or the impact on the target of an input is nonlinear

James, Witten, Hastie, Tibshirani
Introduction to Statistical Learning



Where do you stick the model?

- Make model scoring a service: work common to all model evaluations happens in one place (e.g., logging of scores and feature values for later analysis)

- Easier option: save Python model objects and have scoring be a Python service (e.g., with Tornado)

    - Advantages: easy to set-up

    - Disadvantages: all the problems with pickling, another production runtime (if you're not already using Python), GIL (no concurrent model evaluation)

Other option: create (custom) serialization format, save models in Python, and load in a service in a different language (e.g., Scala/Go)

- Advantages: Runtime consistency, fun evaluation optimizations (e.g, concurrently scoring all the trees in a forest), type checking

- Disadvantages: Have to write serializer/deserializer (PMML is a “standard” but no scikit support)

Better if your RPC protocol supports type-checking (e.g. protobuf or thrift)!

Harder problems

- Feature engineering: figuring out what inputs are valuable to the model (e.g., the “card_use_24h” input)
- Getting data into the right format in production: say you generate training data on Hadoop—what do you do in production?
- Evaluating the production model performance and training new models? (Counterfactual evaluation)

Thanks

@mlmanapat

Slides, Jupyter notebook, data, and related talks at
http://mlmanapat.com

Shameless plug:
Stripe is hiring engineers and data scientists