Model evaluation and implementation

CREDIT RISK MODELING IN PYTHON

Michael Crabtree
Data Scientist, Ford Motor Company

Comparing classification reports

- Create the reports with `classification_report()` and compare

Logistic Regression

	precision	recall	f1-score	support
Non-Default	0.81	0.98	0.89	9198
Default	0.71	0.17	0.27	2586
micro avg	0.80	0.80	0.80	11784
macro avg	0.76	0.57	0.58	11784
weighted avg	0.79	0.80	0.75	11784

Gradient Boosted Tree

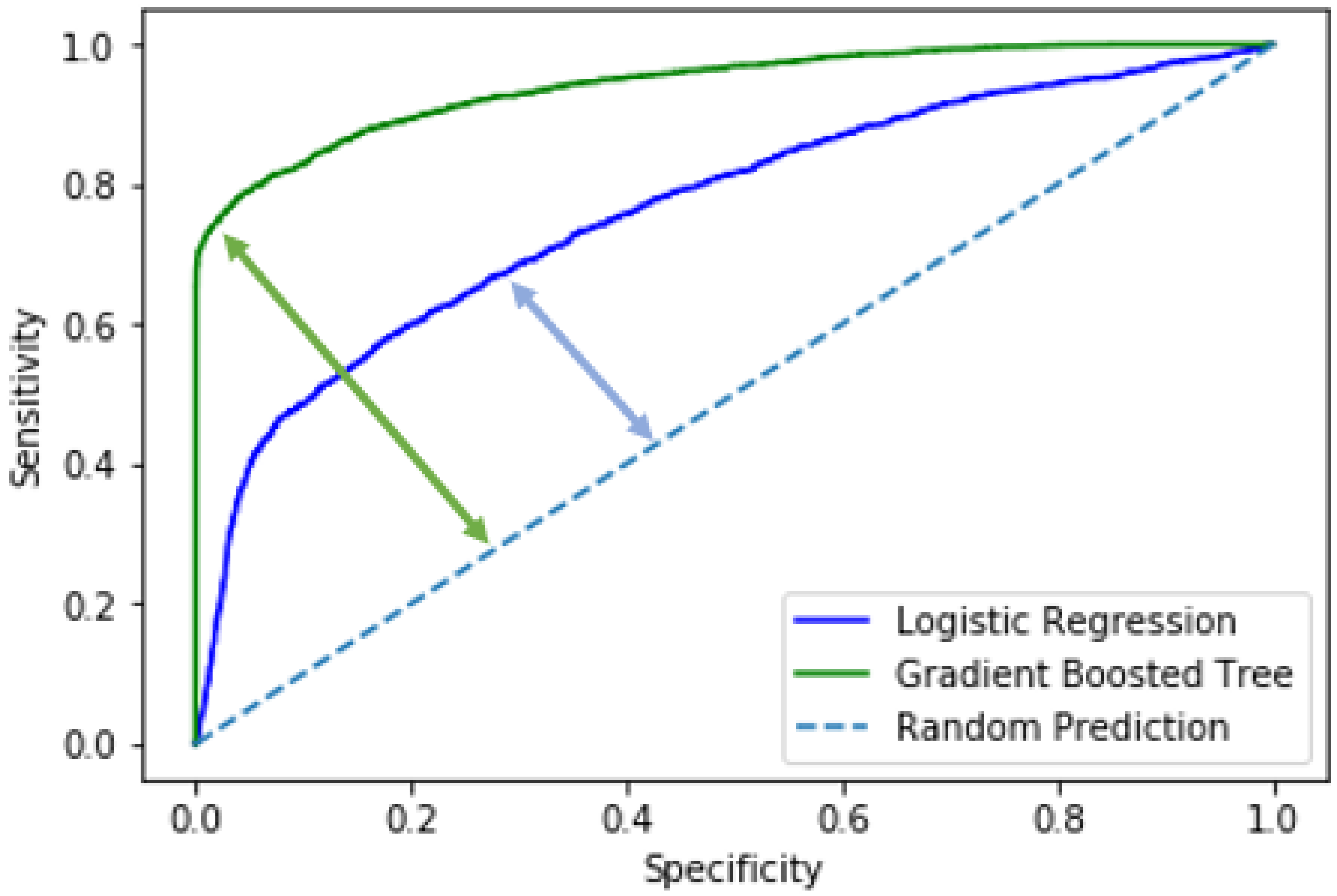
	precision	recall	f1-score	support
Non-Default	0.90	0.98	0.94	9198
Default	0.91	0.63	0.74	2586
micro avg	0.90	0.90	0.90	11784
macro avg	0.91	0.80	0.84	11784
weighted avg	0.90	0.90	0.90	11784

$$F_1 Score = 2 * \left(\frac{precision \; * recall}{precision + recall}\right)$$

$$Macro\ Average = \frac{F_1 Score(Default) + F_1 Score(NonDefault)}{2}$$

ROC and AUC analysis

- Models with better performance will have more lift
- More lift means the AUC score is higher



Model calibration

- We want our probabilities of default to accurately represent the model's confidence level
  - The probability of default has a degree of uncertainty in it's predictions
- A sample of loans and their predicted probabilities of default should be close to the percentage of defaults in that sample

Sample of loans	Average predicted PD	Sample percentage of actual defaults	Calibrated?
10	0.12	0.12	Yes
10	0.25	0.65	No

1 http://datascienceassn.org/sites/default/files/Predicting%20good%20probabilities%20with%20supervised%20le

Calculating calibration

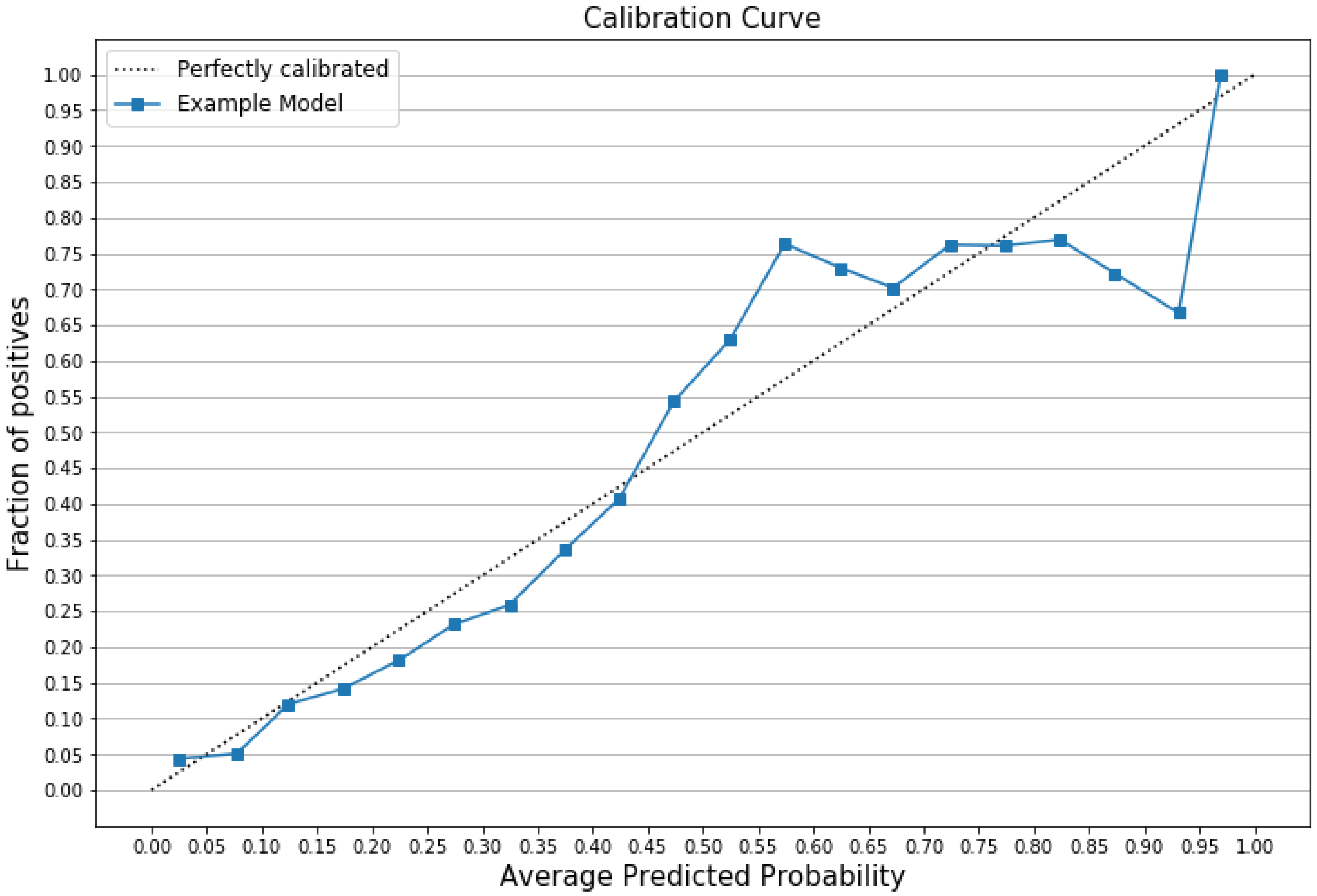
- Shows percentage of true defaults for each predicted probability
- Essentially a line plot of the results of `calibration_curve()`

```
from sklearn.calibration import calibration_curve
calibration_curve(y_test, probabilities_of_default, n_bins = 5)
```

```
# Fraction of positives
(array([0.09602649, 0.19521012, 0.62035996, 0.67361111]),
# Average probability
 array([0.09543535, 0.29196742, 0.46898465, 0.65512207]))
```

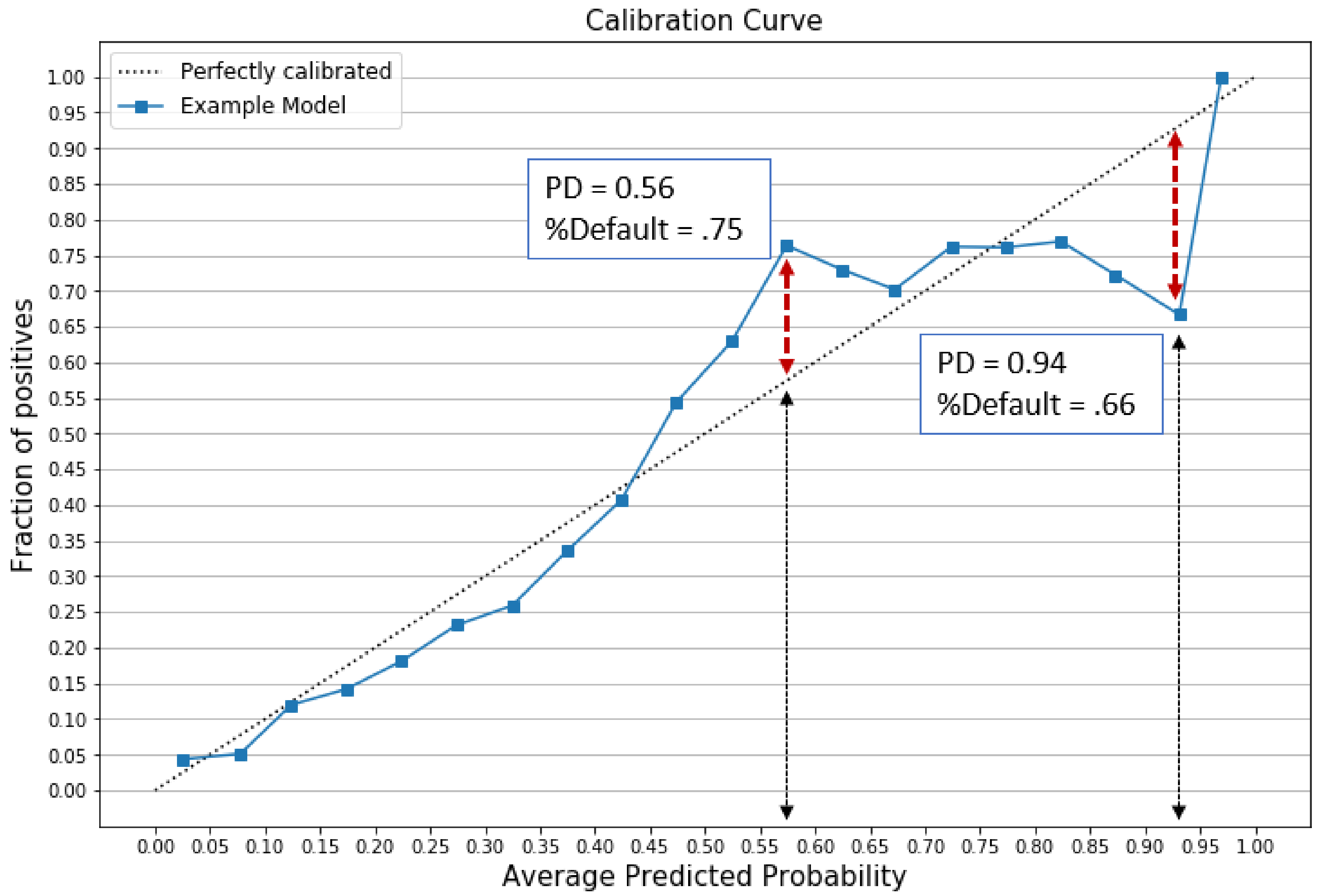
Plotting calibration curves

```
plt.plot(mean_predicted_value, fraction_of_positives, label="%s" % "Example Model")
```

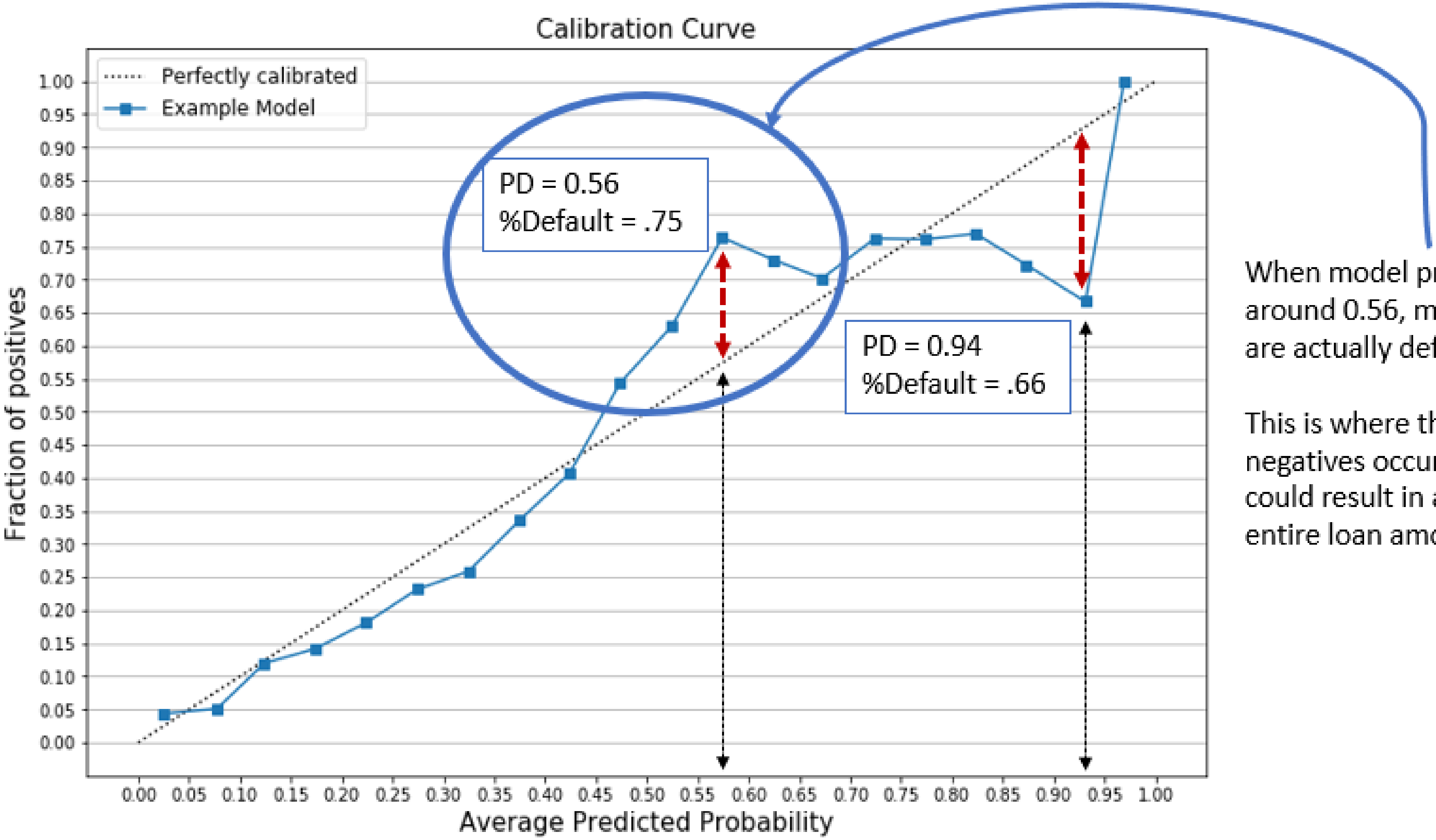


Checking calibration curves

- As an example, two events selected (above and below perfect line)



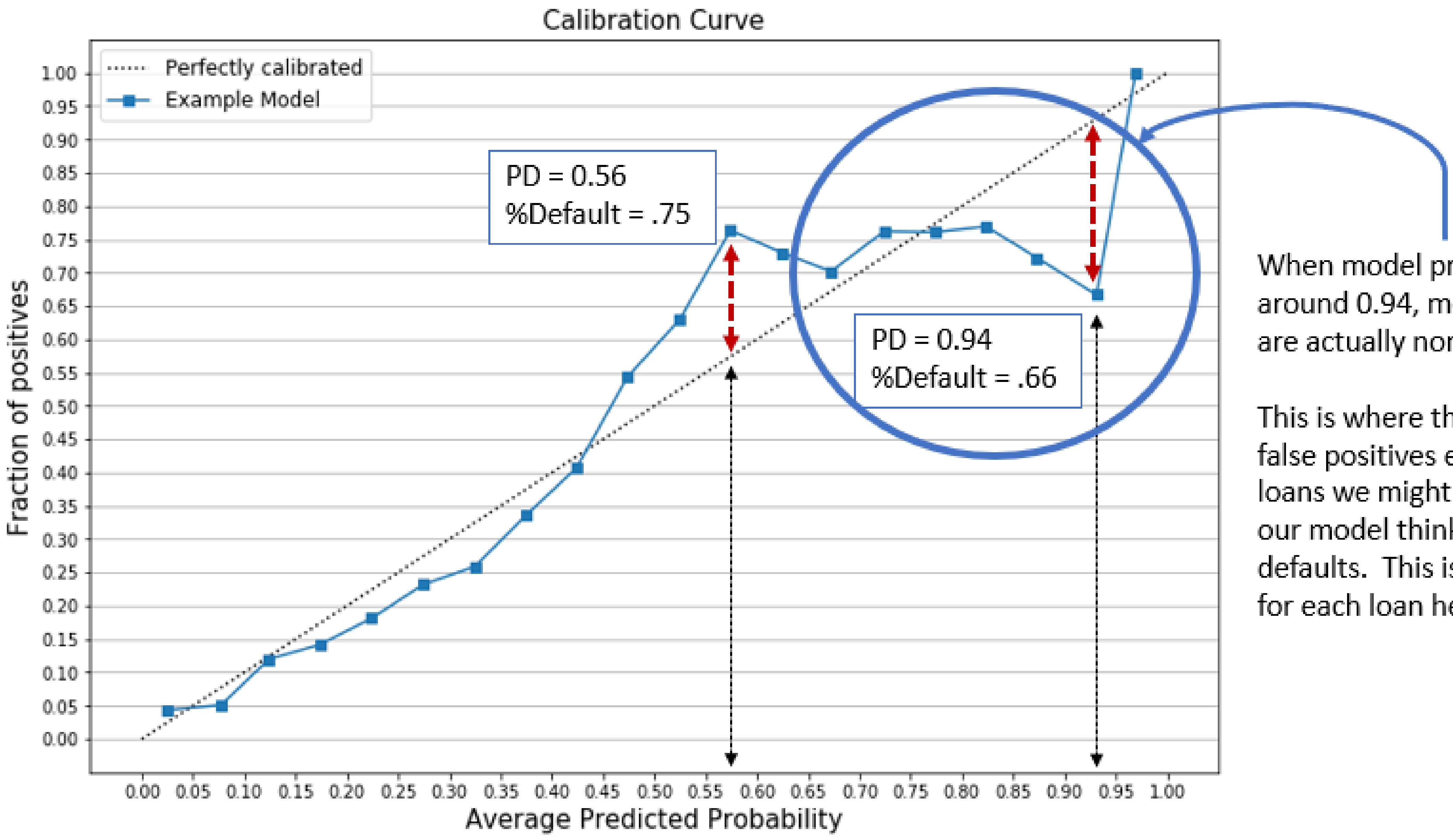
Calibration curve interpretation



When model predictions are around 0.56, more of the loans are actually defaults

This is where those costly false negatives occur, and a default could result in a loss for the entire loan amount

Calibration curve interpretation



When model predictions are around 0.94, more of the loans are actually non-defaults

This is where the majority of the false positives exist. These are loans we might deny because our model thinks they are defaults. This is a loss of profit for each loan here

Let's practice!

CREDIT RISK MODELING IN PYTHON

Credit acceptance rates

CREDIT RISK MODELING IN PYTHON

Michael Crabtree
Data Scientist, Ford Motor Company

Thresholds and loan status

- Previously we set a threshold for a range of `prob_default` values
  - This was used to change the predicted `loan_status` of the loan

```
preds_df['loan_status'] = preds_df['prob_default'].apply(lambda x: 1 if x > 0.4 else 0)
```

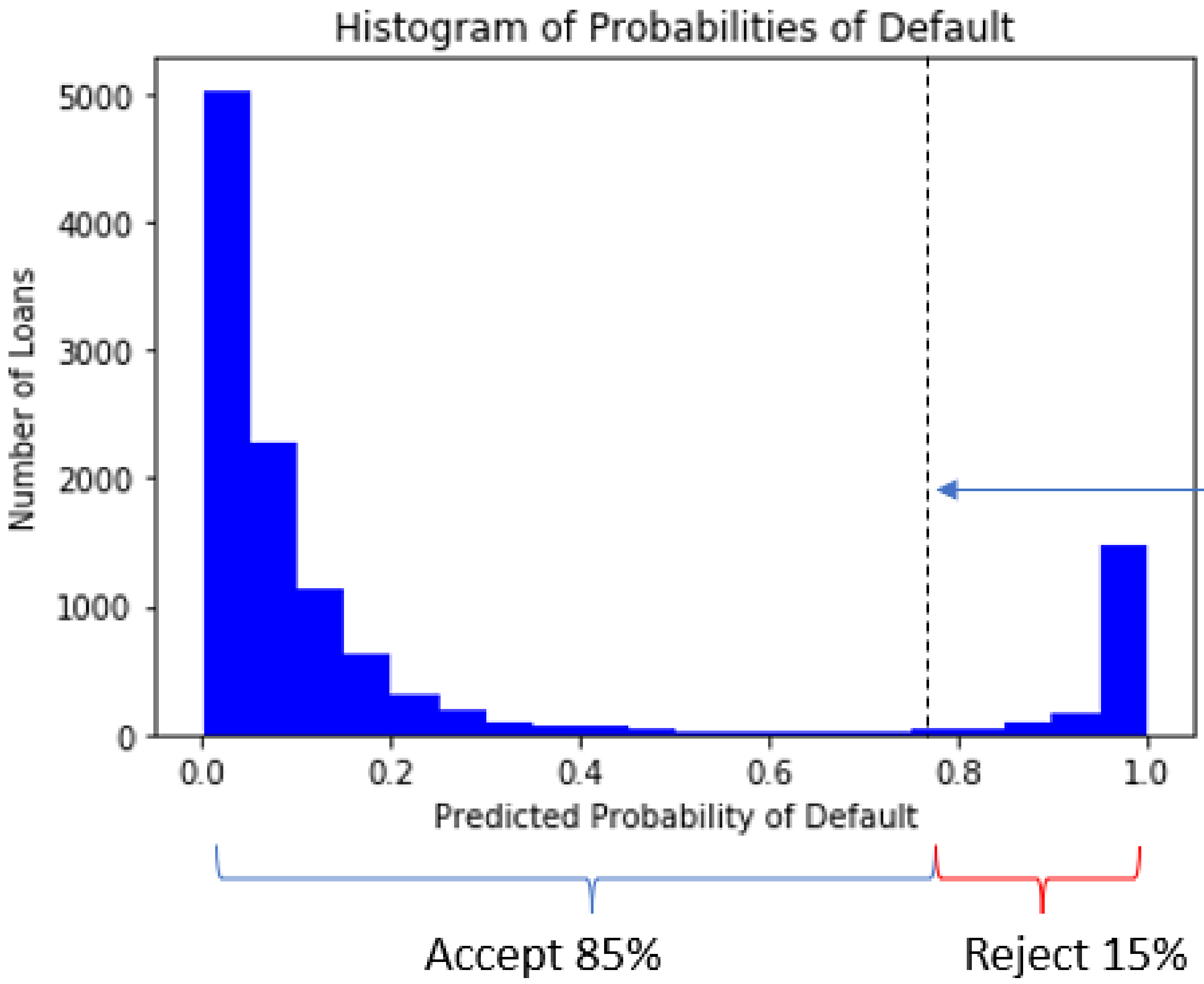
Loan	prob_default	threshold	loan_status
1	0.25	0.4	0
2	0.42	0.4	1
3	0.75	0.4	1

Thresholds and acceptance rate

- Use model predictions to set better thresholds
  - Can also be used to approve or deny new loans
- For all new loans, we want to deny probable defaults
  - Use the test data as an example of new loans
- Acceptance rate: what percentage of new loans are accepted to keep the number of defaults in a portfolio low
  - Accepted loans which are defaults have an impact similar to false negatives

Understanding acceptance rate

- Example: Accept 85% of loans with the lowest `prob_default`



Total Loans: 11,784

Accept 85%: 10,016

Reject 15%: 1,786

What threshold value do we need to reject the top 15%?

Calculating the threshold

- Calculate the threshold value for an 85% acceptance rate

```
import numpy as np
# Compute the threshold for 85% acceptance rate
threshold = np.quantile(prob_default, 0.85)
```

```
0.804
```

Loan	prob_default	Threshold	Predicted loan_status	Accept or Reject
1	0.65	0.804	0	Accept
2	0.85	0.804	1	Reject

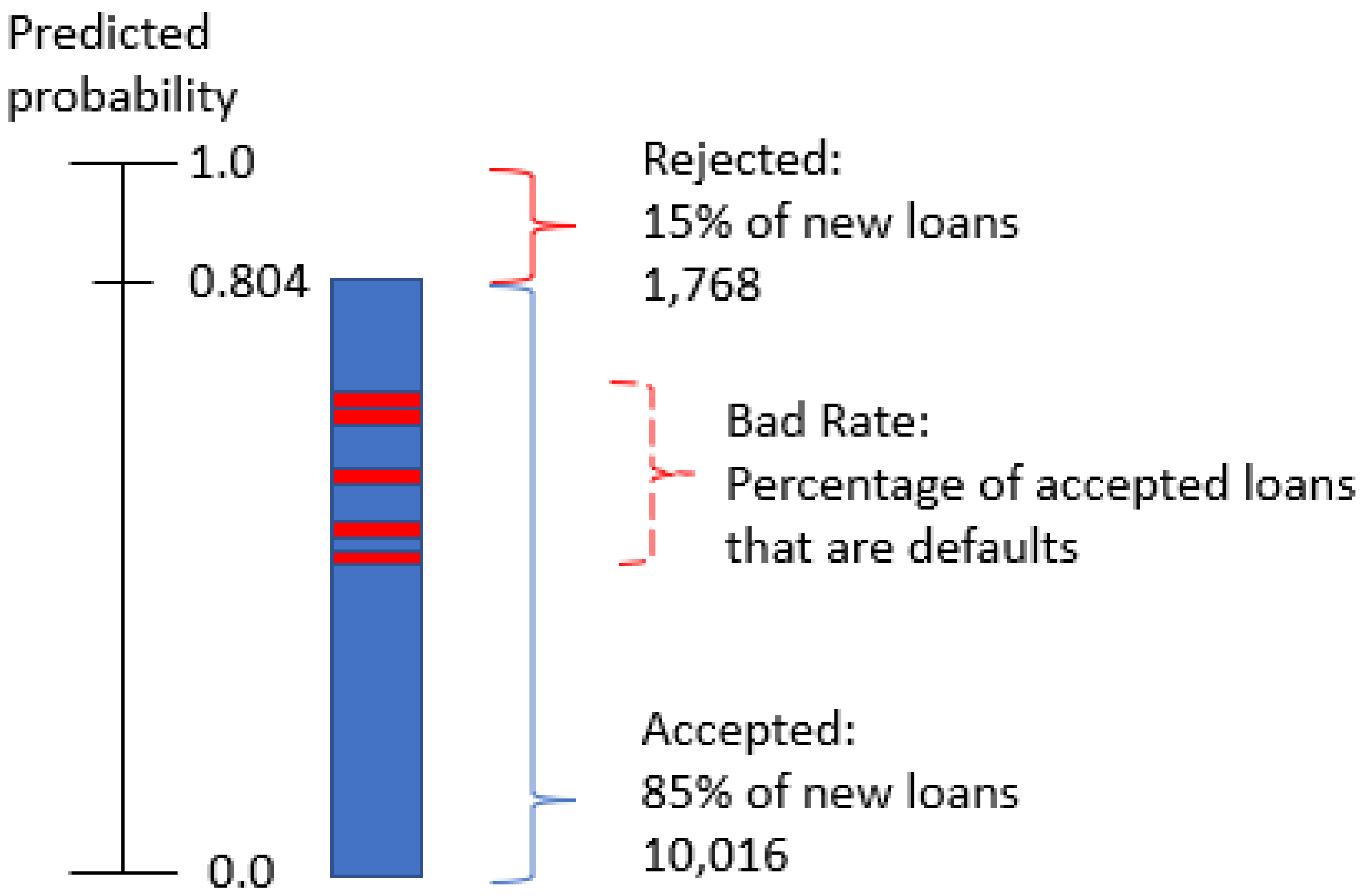
Implementing the calculated threshold

- Reassign `loan_status` values using the new threshold

```
# Compute the quantile on the probabilities of default
preds_df['loan_status'] = preds_df['prob_default'].apply(lambda x: 1 if x > 0.804 else 0)
```

Bad Rate

- Even with a calculated threshold, some of the accepted loans will be defaults
- These are loans with `prob_default` values around where our model is not well calibrated



Bad rate calculation

$$Bad\ Rate = \frac{Accepted\ Defaults}{Total\ Accepted\ Loans}$$

```
#Calculate the bad rate
np.sum(accepted_loans['true_loan_status']) / accepted_loans['true_loan_status'].count()
```

- If non-default is `0` , and default is `1` then the `sum()` is the count of defaults
- The `.count()` of a single column is the same as the row count for the data frame

Let's practice!

CREDIT RISK MODELING IN PYTHON

Credit strategy and minimum expected loss

CREDIT RISK MODELING IN PYTHON

Michael Crabtree
Data Scientist, Ford Motor Company

Selecting acceptance rates

- First acceptance rate was set to 85%, but other rates might be selected as well
- Two options to test different rates:
  - Calculate the threshold, bad rate, and losses manually
  - Automatically create a table of these values and select an acceptance rate
- The table of all the possible values is called a strategy table

Setting up the strategy table

- Set up arrays or lists to store each value

```
# Set all the acceptance rates to test
accept_rates = [1.0, 0.95, 0.9, 0.85, 0.8, 0.75, 0.7, 0.65, 0.6, 0.55,
                0.5, 0.45, 0.4, 0.35, 0.3, 0.25, 0.2, 0.15, 0.1, 0.05]
# Create lists to store thresholds and bad rates
thresholds = []
bad_rates = []
```

Calculating the table values

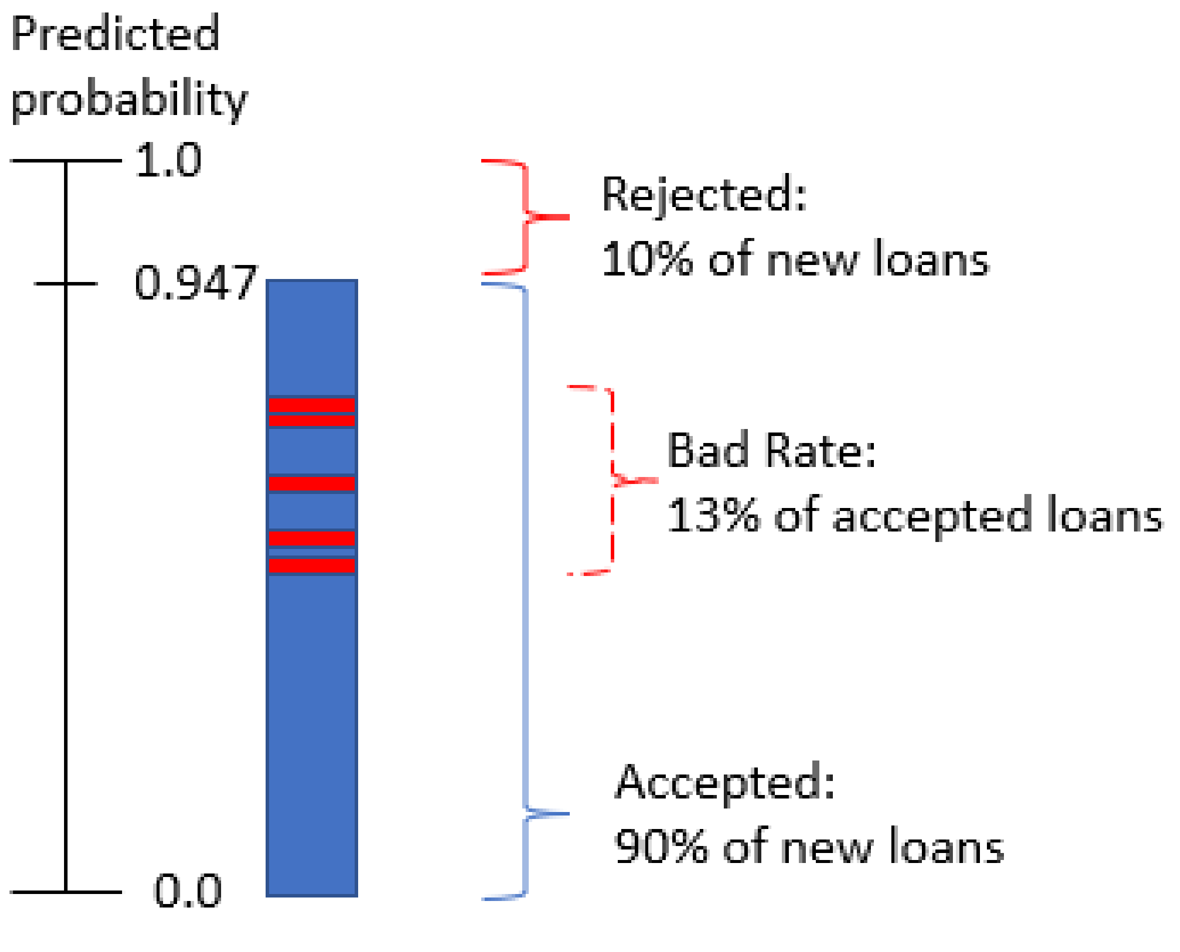
- Calculate the threshold and bad rate for all acceptance rates

```
for rate in accept_rates:
    # Calculate threshold
    threshold = np.quantile(preds_df['prob_default'], rate).round(3)
    # Store threshold value in a list
    thresholds.append(np.quantile(preds_gbt['prob_default'], rate).round(3))
    # Apply the threshold to reassign loan_status
    test_pred_df['pred_loan_status'] = \
        test_pred_df['prob_default'].apply(lambda x: 1 if x > thresh else 0)
    # Create accepted loans set of predicted non-defaults
    accepted_loans = test_pred_df[test_pred_df['pred_loan_status'] == 0]
    # Calculate and store bad rate
    bad_rates.append(np.sum((accepted_loans['true_loan_status'])
             / accepted_loans['true_loan_status'].count()).round(3))
```

Strategy table interpretation

```
strat_df = pd.DataFrame(zip(accept_rates, thresholds, bad_rates),
                        columns = ['Acceptance Rate','Threshold','Bad Rate'])
```

Acceptance Rate	Threshold	Bad Rate
1.00	0.999	0.219
0.95	0.988	0.177
0.90	0.947	0.133
0.85	0.503	0.097
0.80	0.330	0.078
0.75	0.227	0.066
0.70	0.163	0.055



Adding accepted loans

- The number of loans accepted for each acceptance rate
  - Can use `len()` or `.count()`

```
len(test_pred_df[test_pred_df['prob_default'] < np.quantile(test_pred_df['prob_default'], accept_rate)])
```

Acceptance Rate	Threshold	Bad Rate	Num Accepted Loans	Avg Loan Amnt	Estimated Value
1.00	0.999	0.219	11379	9556.28	61112391.49
0.95	0.988	0.177	10591	9556.28	65382022.72
0.90	0.947	0.133	10025	9556.28	70318452.94
0.85	0.503	0.097	9390	9556.28	72325176.18
0.80	0.330	0.078	8857	9556.28	71436136.33
0.75	0.227	0.066	8229	9556.28	68258329.21
0.70	0.163	0.055	7685	9556.28	65361610.50

Adding average loan amount

- Average `loan_amnt` from the test set data

Acceptance Rate	Threshold	Bad Rate	Num Accepted Loans	Avg Loan Amnt	Estimated Value
1.00	0.999	0.219	11379	9556.28	61112391.49
0.95	0.988	0.177	10591	9556.28	65382022.72
0.90	0.947	0.133	10025	9556.28	70318452.94
0.85	0.503	0.097	9390	9556.28	72325176.18
0.80	0.330	0.078	8857	9556.28	71436136.33
0.75	0.227	0.066	8229	9556.28	68258329.21
0.70	0.163	0.055	7685	9556.28	65361610.50

```
np.mean(test_pred_df['loan_amnt'])
```

Estimating portfolio value

- Average value of accepted loan non-defaults minus average value of accepted defaults
- Assumes each default is a loss of the `loan_amnt`

Acceptance Rate	Threshold	Bad Rate	Num Accepted Loans	Avg Loan Amnt	Estimated Value
1.00	0.999	0.219	11379	9556.28	61112391.49
0.95	0.988	0.177	10591	9556.28	65382022.72
0.90	0.947	0.133	10025	9556.28	70318452.94
0.85	0.503	0.097	9390	9556.28	72325176.18
0.80	0.330	0.078	8857	9556.28	71436136.33
0.75	0.227	0.066	8229	9556.28	68258329.21
0.70	0.163	0.055	7685	9556.28	65361610.50

```
((strat_df['Num Accepted Loans'] * (1 - strat_df['Bad Rate'])) * strat_df['Avg Loan Amnt'])
    - (strat_df['Num Accepted Loans'] * strat_df['Bad Rate'] * strat_df['Avg Loan Amnt'])
```

Total expected loss

- How much we expect to lose on the defaults in our portfolio

$$Total\ Expected\ Loss = \sum_{x=1}^{n} PD_x * LGD_x * EAD_x$$

```
# Probability of default (PD)
test_pred_df['prob_default']
# Exposure at default = loan amount (EAD)
test_pred_df['loan_amnt']
# Loss given default = 1.0 for total loss (LGD)
test_pred_df['loss_given_default']
```

Let's practice!

CREDIT RISK MODELING IN PYTHON

Course wrap up

CREDIT RISK MODELING IN PYTHON

Michael Crabtree
Data Scientist, Ford Motor Company

Your journey...so far

- Prepare credit data for machine learning models
  - Important to understand the data
  - Improving the data allows for high performing simple models
- Develop, score, and understand logistic regressions and gradient boosted trees
- Analyze the performance of models by changing the data
- Understand the financial impact of results
- Implement the model with an understanding of strategy

Risk modeling techniques

- The models and framework in this course:
  - Discrete-time hazard model (point in time): the probability of default is a point-in-time event
  - Stuctural model framework: the model explains the default even based on other factors
- Other techniques
  - Through-the-cycle model (continuous time): macro-economic conditions and other effects are used, but the risk is seen as an independent event
  - Reduced-form model framework: a statistical approach estimating probability of default as an independent Poisson-based event

Choosing models

- Many machine learning models available, but logistic regression and tree models were used
  - These models are simple and explainable
  - Their performance on probabilities is acceptable
- Many financial sectors prefer model interpretability
  - Complex or "black-box" models are a risk because the business cannot explain their decisions fully
  - Deep neural networks are often too complex

Tips from me to you

- Focus on the data
  - Gather as much data as possible
  - Use many different techniques to prepare and enhance the data
  - Learn about the business
  - Increase value through data
- Model complexity can be a two-edged sword
  - Really complex models may perform well, but are seen as a "black-box"
  - In many cases, business users will not accept a model they cannot understand
  - Complex models can be very large and difficult to put into production

Thank you!

CREDIT RISK MODELING IN PYTHON