

Regressor and classifier metrics

Quantifying model performance

Terence Parr
MSDS program
University of San Francisco

Regressor metrics

Common regressor metrics

- Mean squared error
Range $0..\infty$, units(y)², symmetric


$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- Mean absolute value
Range $0..\infty$, units(y), symmetric

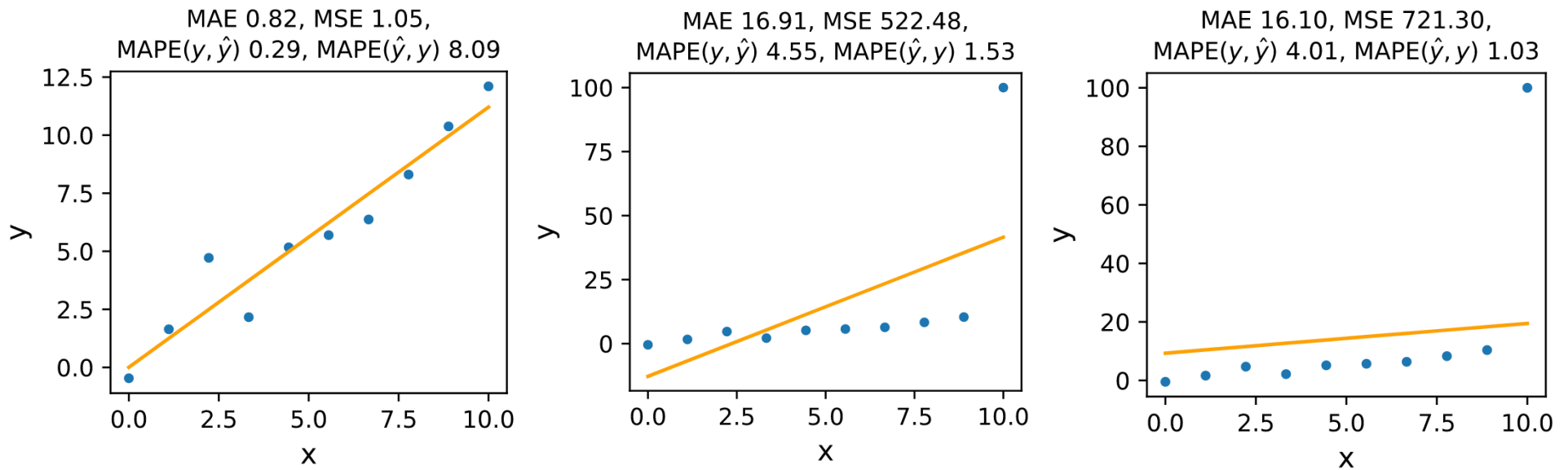
$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

- Mean absolute percentage error
Range $0..\infty$, unitless, **asymmetric**
undefined if $y=0$

$$MAPE = \frac{1}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

(For moment, think of symmetry as $\text{metric}(y, \hat{y}) = \text{metric}(\hat{y}, y)$)  UNIVERSITY OF SAN FRANCISCO

MAE, MSE, MAPE example



MSE incorrectly makes last model look horrible and worse than 2nd model due to outlier.
MAE isn't perfect either as it thinks 2nd and 3rd models are about the same.
Note MAPE asymmetry!

R²

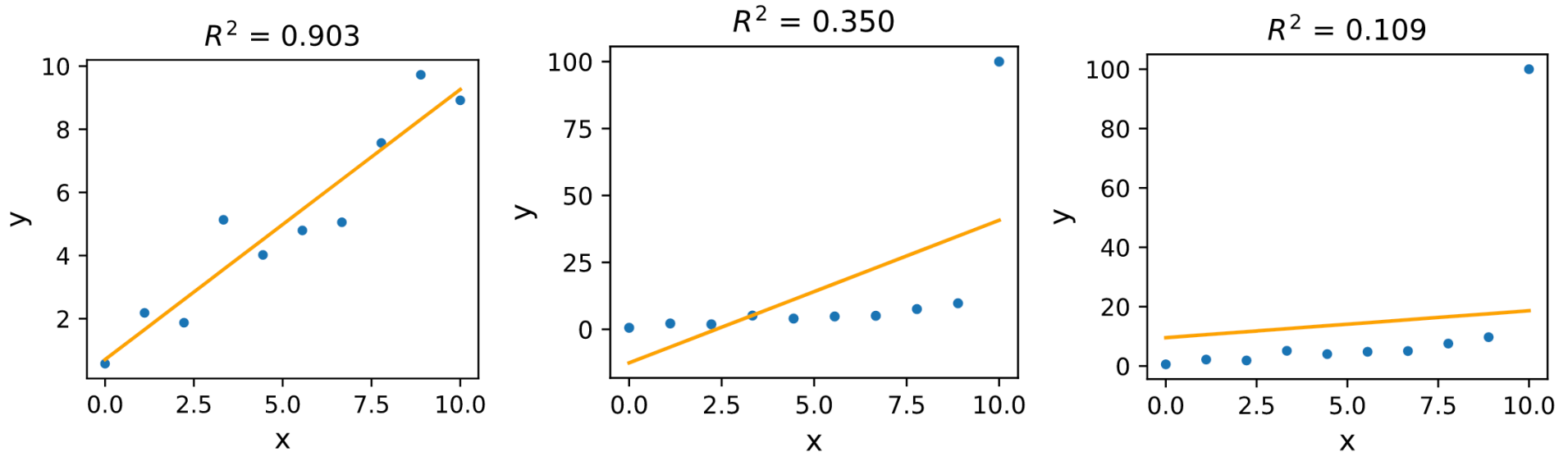
- How well our model performs compared to a “mean model”

$$R^2 = 1 - \frac{\text{Squared error}}{\text{Variation from mean}} = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

- Range of possible values: $(-\infty, 1]$
- Our model could be really bad, giving large negative numbers
- For OLS linear models, R^2 in $[0, 1]$
- R^2 is default regressor metric for sklearn

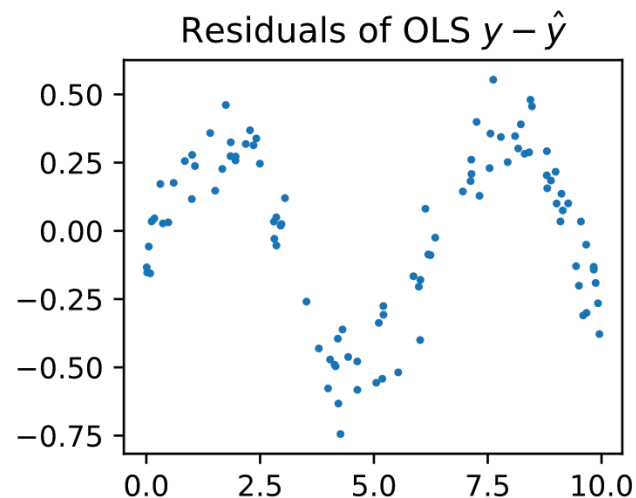
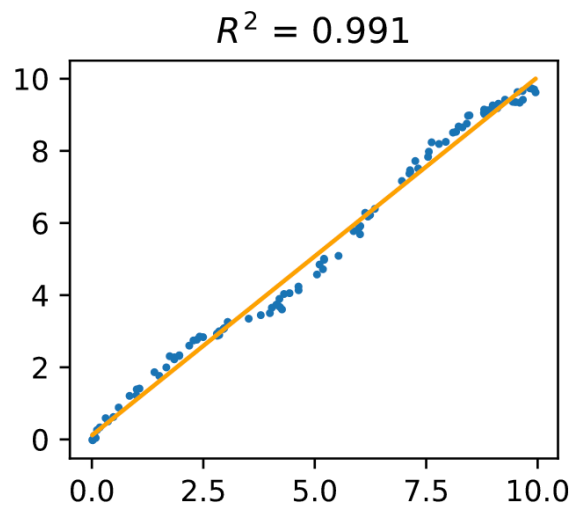
Low R^2 doesn't always imply bad model

- Not best metric; here are 3 graphs with good, bad, decent fits
- The 3rd has lowest R^2 but it's a way better model than 2nd



High R^2 doesn't always imply good model

- This linear model looks pretty good and has $R^2 = 0.991$
- But check the residual plot! Model doesn't capture nature of x, y



Which metric should we use?

- That depends what we care about for business reasons
- For prices, we usually care more about the percentage error than the absolute amount
- \$500 off for a \$1,000 apartment is 0.5x or 1.5x off and a big deal but \$500 off for a \$1 million apartment is a trivial difference
- For things like body temperature that will most likely all be within a small range, the mean absolute error (MAE) is a good and interpretable measure
- The percentage error can be interpretable but there are problems with it: asymmetry
- R^2 , MSE and in general squaring error is super sensitive to big deltas

Symmetry in metrics

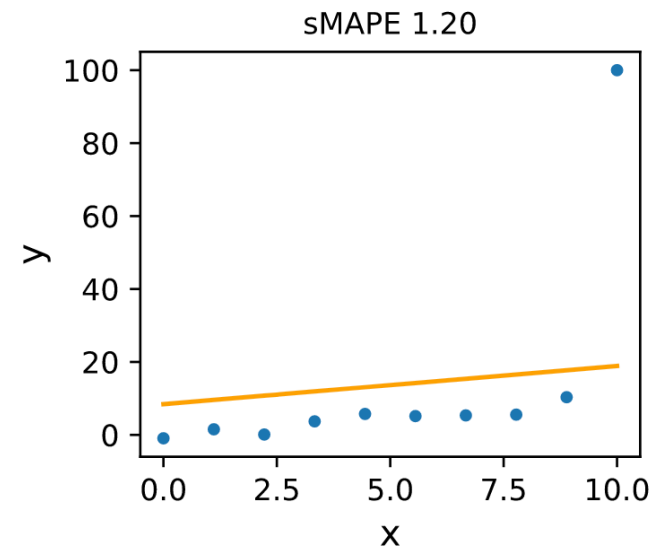
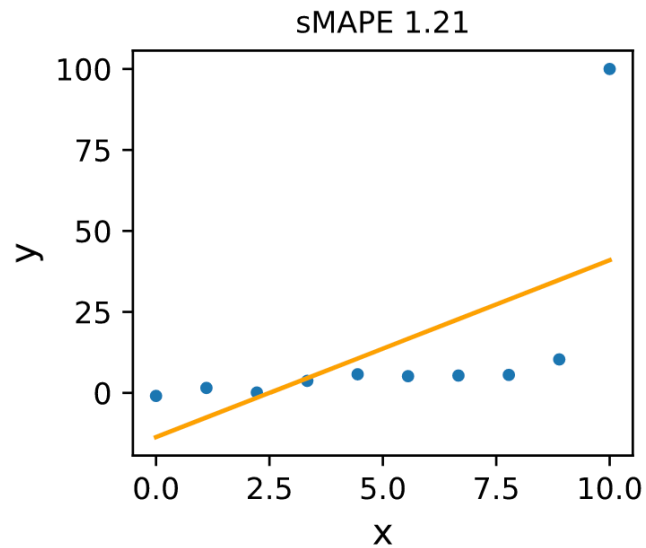
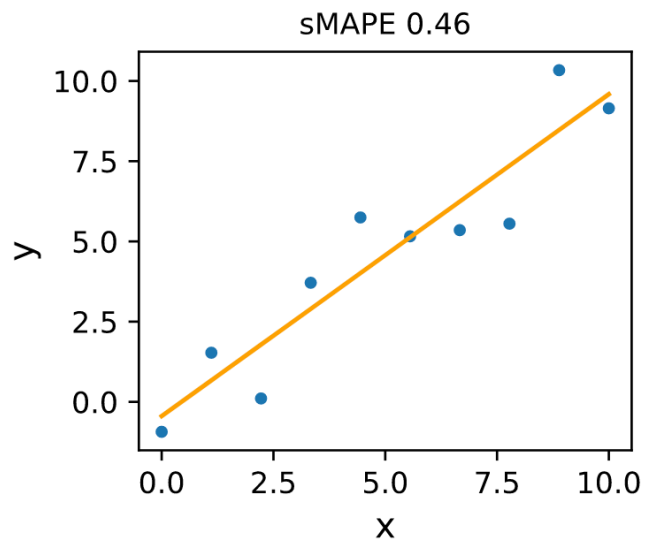
- Most metrics use $y - \hat{y}$ but ratio y/\hat{y} is often better
- But, most ratio-based metrics are asymmetric, which is bad!
 - If $y=100$, $\hat{y}=0.01$: $MAPE = 0.9999$
 - If $y=0.01$, $\hat{y}=100$: $MAPE = 9999$

$$MAPE = \frac{1}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

- Try symmetric MAPE
Range 0..2, unitless, symmetric,
undefined at $y=0$ and $\hat{y}=0$
(have to work around those landmines)
 - If $y=100$, $\hat{y}=0.01$: $sMAPE = 1.9996$
 - If $y=0.01$, $\hat{y}=100$: $sMAPE = 1.9996$

$$sMAPE = \frac{1}{n} \sum_{i=1}^n \frac{|y_i - \hat{y}_i|}{\frac{1}{2}(|y_i| + |\hat{y}_i|)}$$

sMAPE in action



Classifier metrics

Common classifier metrics

(Ugh; much more complicated than for regressors)

- *TP* = true positive, *TN* = true negative
- *FP* = false positive, *FN* = false negative
- *Confusion matrix* for binary classification to right, but can have larger confusion matrices in general
- The matrices are clear but don't give single metric
- *Accuracy* = correct classification rate = $(TN+TP)/n$
- *Misclassification* rate is $1 - \text{accuracy}$

		<i>Predicted</i>	
		F	T
<i>Actual</i>	F	<i>TN</i>	<i>FP</i>
	T	<i>FN</i>	<i>TP</i>

		<i>Predicted</i>	
		F	T
<i>Actual</i>	F	35	3
	T	1	75

(breast cancer RF)

True/False positive/negative rates

		<i>Predicted</i>	
		F	T
<i>Actual</i>	F	TN	FP
	T	FN	TP

- *True positive rate* is $TP / \text{num-positive}$ (also called *recall*)
Of the actually positive y , how many true positives in \hat{y} ?
 - **TPR** = $TP / \text{true-}y = TP / (TP + FN)$
 - TP over sum of 2nd row in confusion matrix (num true- y is constant)
- *False positive rate* is $FP / \text{num-negative}$
Of the actually false y , how many false positives in \hat{y} ?
 - **FPR** = $FP / \text{false-}y = FP / (FP + TN)$
 - FP over sum of 1st row in confusion matrix (num false- y is constant)

Multi-class confusion matrix

- For C classes, we get C x C matrix
- Optimally, it's a diagonal matrix (correct classifications)
- Example; interest in NYC apartments (lo/med/hi); matrix indicates model is good at predicting low interest apts but not others
- Should focus on features that are predictive of med/high to improve model

	predicted_low	predicted_medium	predicted_high
expected_low	3749	566	83
expected_medium	854	418	119
expected_high	189	174	141

More classifier metrics

(I like Precision/recall)

- Precision/recall useful in binary classification, such as spam
- *Precision* = $TP/(TP+FP)$ “of those predicted as positive, how many did we get right?”
- *Recall* = $TP/(TP+FN)$ “of the positive samples, how many did we find (predict as positive)?”
- *F1* is harmonic mean of precision and recall; $F1 = 2*(P*R)/(P+R)$, which gives equal importance to precision and recall

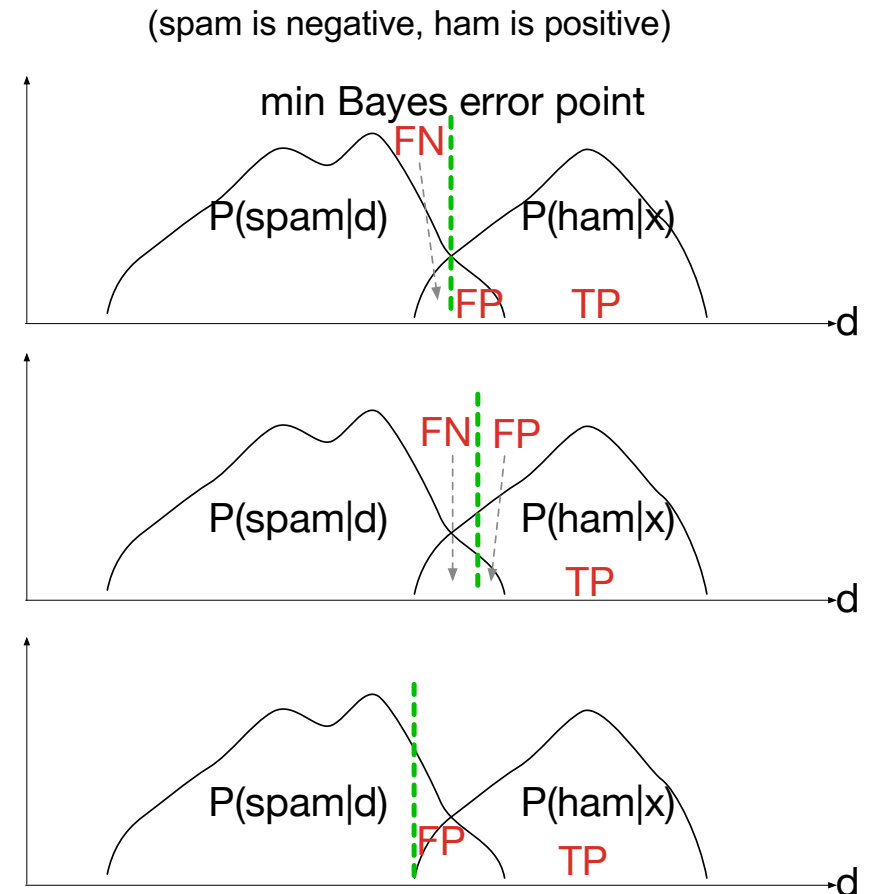
		Predicted		
		F	T	
Actual	F	35	3	precision= 75/(75+3)
	T	1	75	

		Predicted		
		F	T	
Actual	F	35	3	recall= 75/(75+1)
	T	1	75	

AUC ROC

(area under curve, receiver operator curve)

- Measures separability of distributions
- Encapsulates tradeoff in binary classification between **true positive rate** (recall) and **false positive rate** according to score or likelihood threshold
- Recall logistic regression provides probability of class=1 (ham) and we must choose threshold, which then assigns predicted classes
- Shift threshold left/right and you shift (TP, FP) and (TPR, FPR) coordinates



See also <https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc>

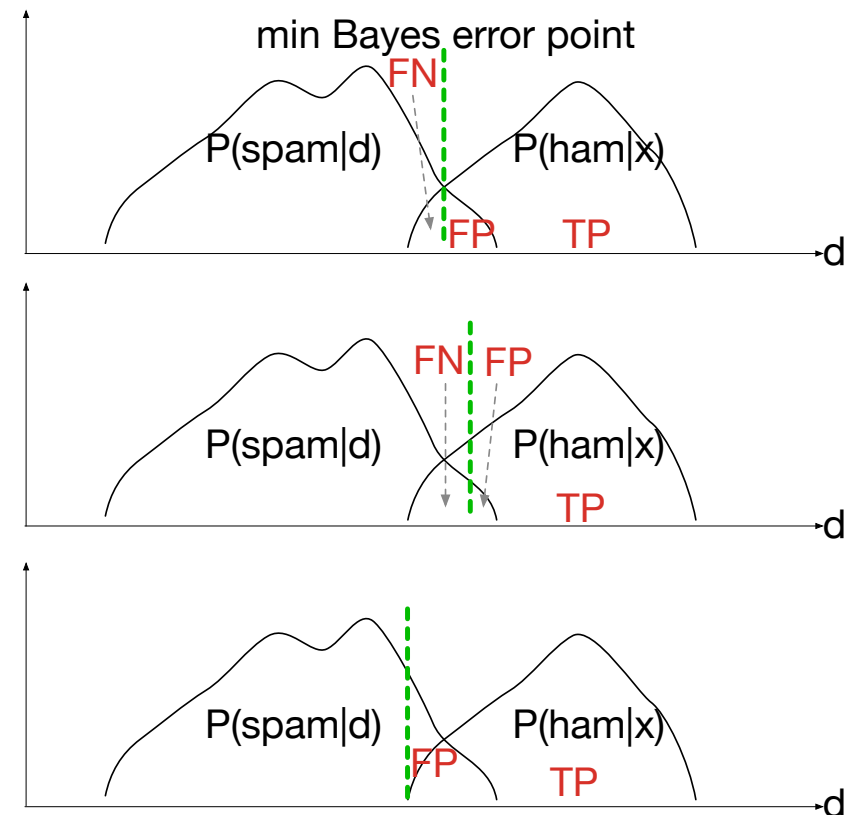
AUC ROC Continued

- For bayes (min error) threshold, we get one (FPR, TPR) coordinate
- Shift threshold to right, FPR, TPR get smaller and we get another coor.
- Shift to left so FN=0, then:
 $TPR = TP / TP = 1.0$, but FPR goes up too

		Predicted	
		F	T
Actual	F	TN	FP
	T	FN	TP

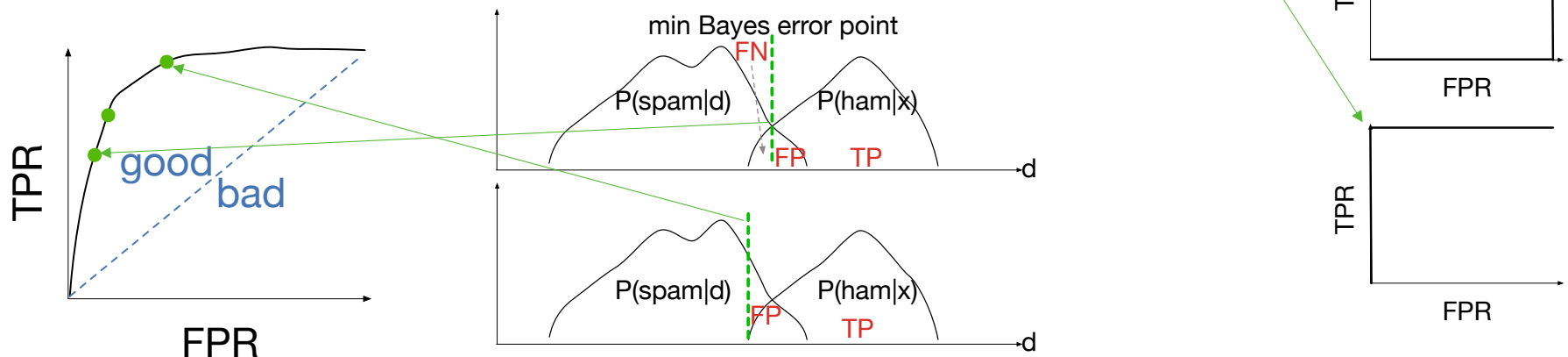
$$TPR = TP / (TP + FN)$$

$$FPR = FP / (FP + TN)$$



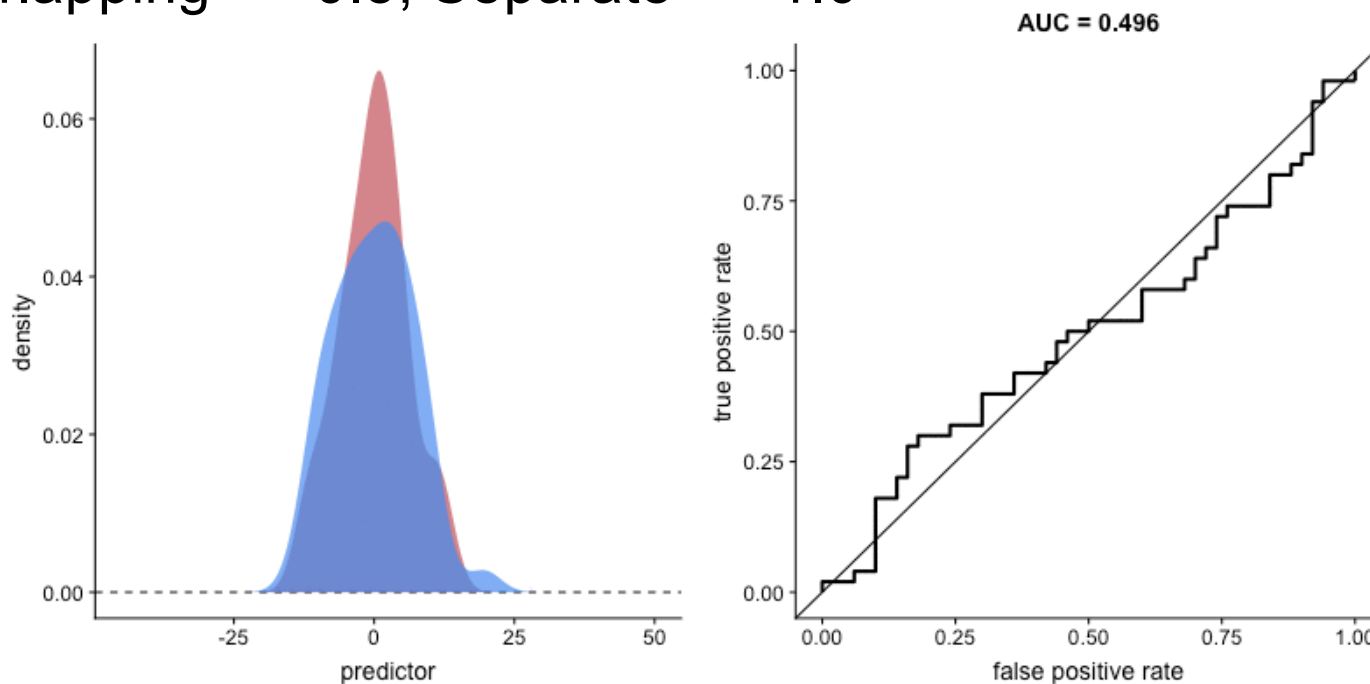
AUC ROC Continued

- When distributions overlap exactly, moving threshold around gives 45 degree line, $AUC=0.5$
- When distributions are reversed, $AUC = 0$
- When distributions are completely separate, $AUC=1$



AUC is a measure of class separability

- Overlapping \Rightarrow 0.5, Separate \Rightarrow 1.0



Computing ROC (or other curves)

- Requires true y and $P(y=1|x)$, which comes from `predict_proba()`
- Sort by probabilities then slide a threshold from top to bottom
- Compute confusion matrix and TPR, FPR at each point, which forms ROC curve
- Example next page

P(\square)	.01	.2	.3	.52	.66	.79	.89	.92	.99
\hat{y}	○	○	○	○	○	○	○	○	○
y	○	○	○	○	□	□	□	□	○

→ P(\square) (TPR,FPR) = (0,0)

P(\square)	.01	.2	.3	.52	.66	.79	.89	.92	.99
\hat{y}	○	○	○	○	○	○	○	○	□
y	○	○	○	○	□	□	□	□	○

→ P(\square) (TPR,FPR) = (0,1/5)

P(\square)	.01	.2	.3	.52	.66	.79	.89	.92	.99
\hat{y}	○	○	○	○	○	○	○	□	□
y	○	○	○	○	□	□	□	□	○

→ P(\square) (TPR,FPR) = (1/4,1/5)

P(\square)	.01	.2	.3	.52	.66	.79	.89	.92	.99
\hat{y}	○	○	○	□	□	□	□	□	□
y	○	○	○	○	□	□	□	□	○

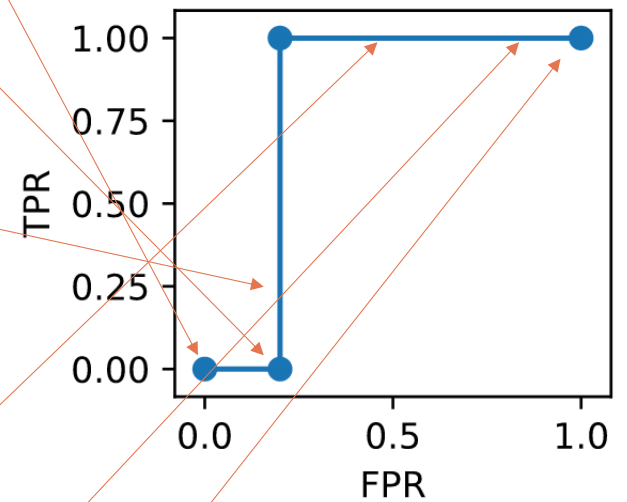
→ P(\square) (TPR,FPR) = (4/4,2/5)

P(\square)	.01	.2	.3	.52	.66	.79	.89	.92	.99
\hat{y}	○	□	□	□	□	□	□	□	□
y	○	○	○	○	□	□	□	□	○

→ P(\square) (TPR,FPR) = (4/4,4/5)

P(\square)	.01	.2	.3	.52	.66	.79	.89	.92	.99
\hat{y}	□	□	□	□	□	□	□	□	□
y	○	○	○	○	□	□	□	□	○

→ P(\square) (TPR,FPR) = (4/4,5/5)



Code to plot ROC from previous slide

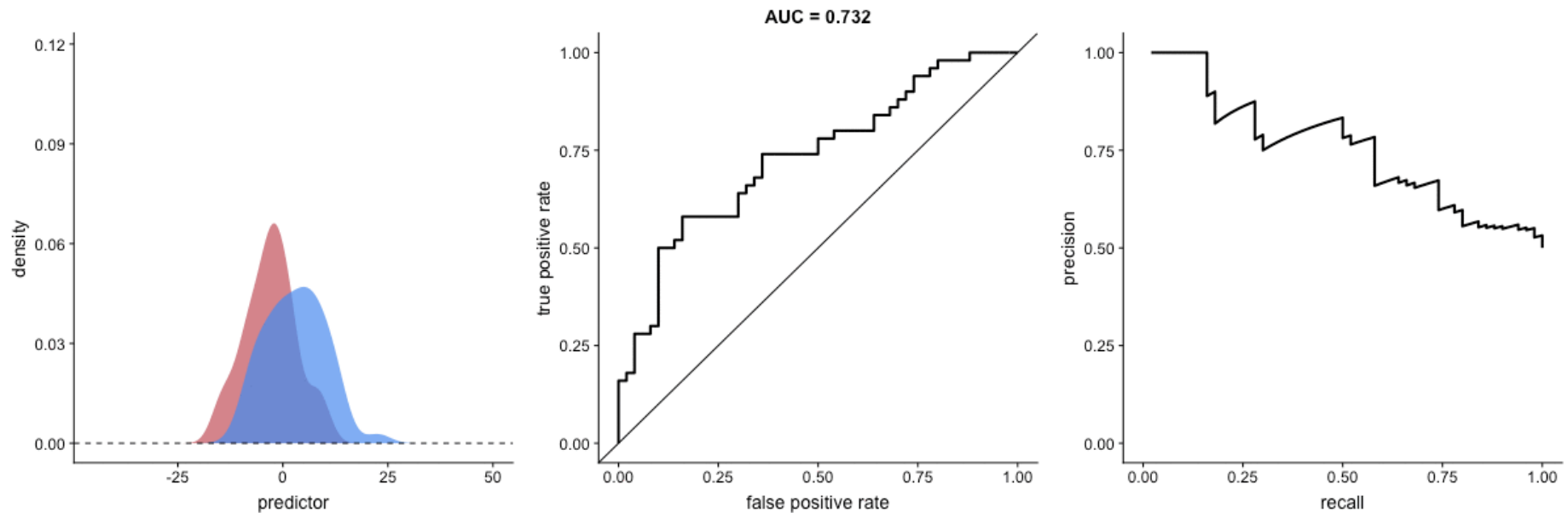
```
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
%config InlineBackend.figure_format = 'svg'

y = np.array([0, 0, 0, 0, 1, 1, 1, 1, 0])
yprob = np.array([0.01,.2,.3,.52,.66,.70,.75,.76,.80])
fpr, tpr, thresholds = metrics.roc_curve(y, yprob, pos_label=1)

plt.figure(figsize=(2.2,2))
plt.plot(fpr,tpr)
plt.scatter(fpr,tpr)
plt.xlabel("FPR")
plt.ylabel("TPR")
```

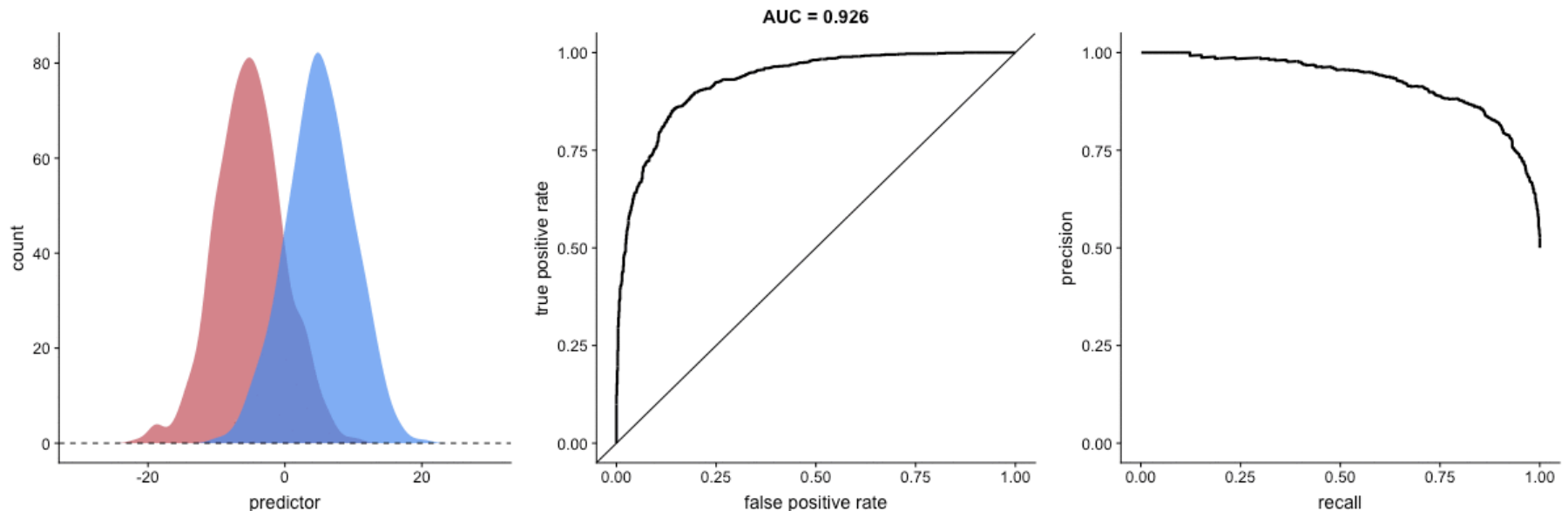
ROC vs Precision-Recall curve

- As variance tightens, ROC goes up whereas PR goes down



Imbalanced classes ROC vs PR curve

- Spam, fraud detection problems are imbalanced; can't trust ROC!
- For same threshold/mean of distribution, only PR curve changes!
- In the end, I favor PR not ROC



Upsampling minority class(es) in imbalanced data sets

Naïve use of `train_test_split()` @ 20%

Imbalanced dataset: Kaggle credit card fraud

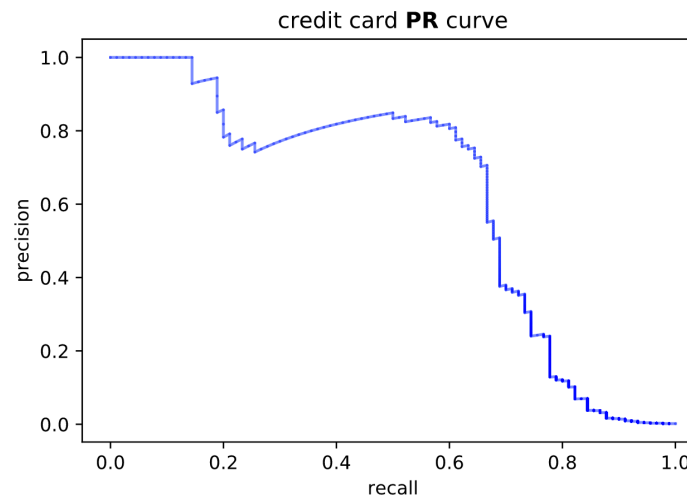
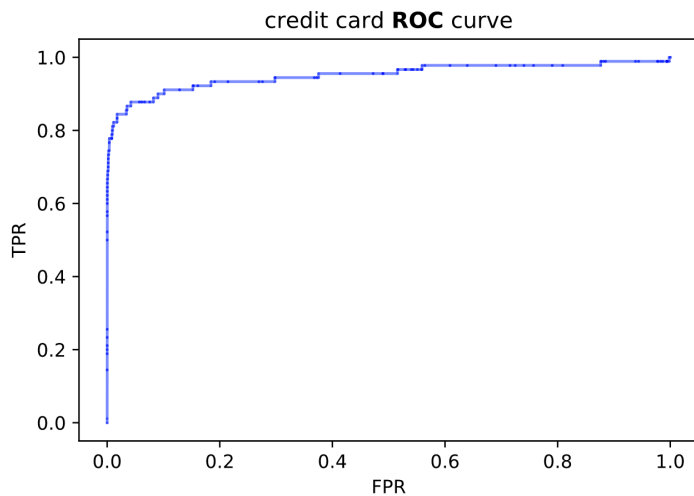
- Num anomalies 492/284807 = 0.17%
- L2 Regularized **LogisticRegression** model
- Accuracy 1.00, AUC ROC 0.93
- F1 0.69, AUC PR 0.62

bad ➔
good ➔

		predicted	
		F	T
actual	F	56840	16
	T	42	64

20% test set

*Why is accuracy=1.0?
(That's rounded up)*



First: Proper split to ensure 20% minority class

- Must split out test set before modeling but get 20% from each class at original ratio of class 1 / class 0 (particularly important for small n)
 - Accuracy 1.00, AUC ROC 0.93
 - F1 0.70, AUC PR 0.65
- If num anomalies $492/284807 = 0.17\%$, then need:
 - TRAIN num fraud $393/227845 = 0.17\%$
 - TEST num fraud $99/56962 = 0.17\%$

	F	T
F	56848	15
T	37	62

```
df_good = df[df['Class']==0]
df_fraud = df[df['Class']==1]
```

```
df_train_good, df_test_good = train_test_split(df_good, test_size=0.20)
df_train_fraud, df_test_fraud = train_test_split(df_fraud, test_size=0.20)
```

```
df_train = pd.concat([df_train_good, df_train_fraud], axis=0)
df_test = pd.concat([df_test_good, df_test_fraud], axis=0)
```

Then: upsample minority in training set

- Upsampled TRAIN num fraud $3930/231382 = 1.70\%$
- TEST num fraud $99/56962 = 0.17\%$

```
df_good = df_train[df_train['Class']==0]
df_fraud = df_train[df_train['Class']==1]

df_fraud_balanced = df_fraud.sample(int(len(df_fraud)*10), replace=True)
df_train_upsampled = pd.concat([df_good, df_fraud_balanced], axis=0)
```

Tweet/paper on how this can cause data leakage

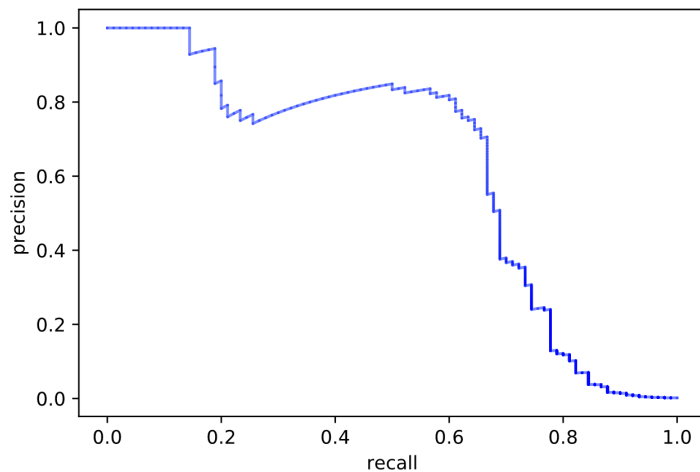
<https://arxiv.org/abs/2001.06296>

<https://twitter.com/Gillesvdwiele/status/1219194600994283520>

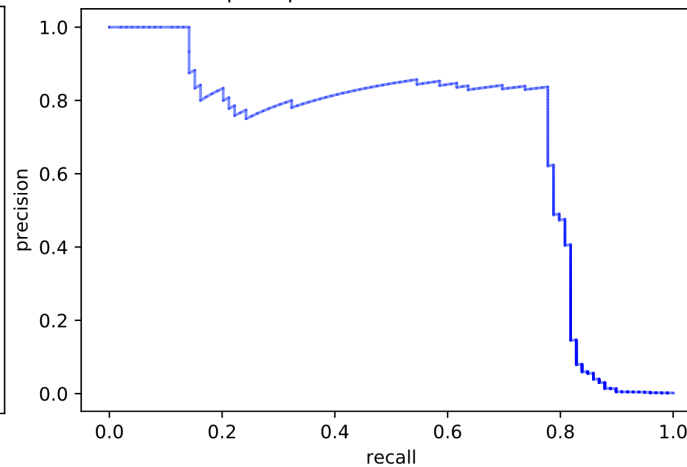
Upsampling fraud 10x in training data

- We get improvement with upsampling for logistic regression, at least for this Kaggle credit card data set

credit card **PR** curve



Upsampled credit card **PR** curve



- Accuracy 1.00, AUC ROC 0.95
- F1 0.73, AUC PR 0.70

	F	T
F	56827	36
T	22	77

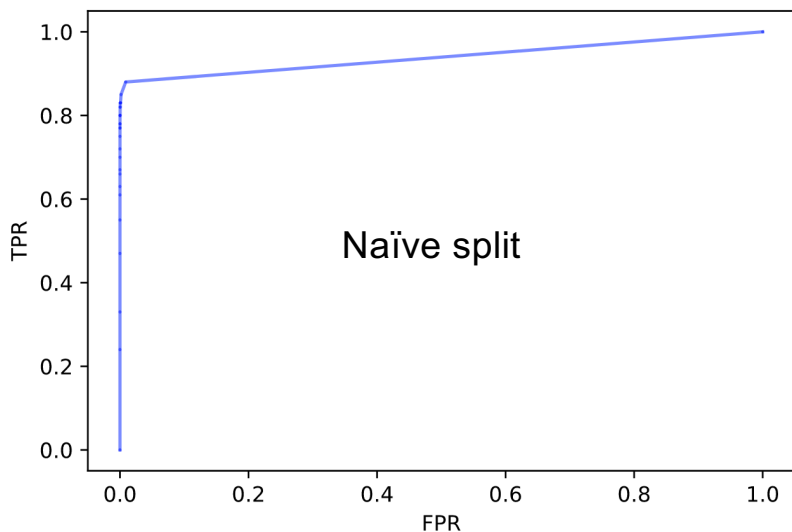
L2 Logistic Regression summary

- Naive split
 - F1 0.69, Accuracy 1.00
 - ROC 0.93, AUC PR 0.62
- Proper split (just making sure train/test balance is right can help)
 - F1 0.70, Accuracy 1.00
 - AUC ROC 0.93, AUC PR 0.65
- Upsample:
 - F1 0.73, Accuracy 1.00
 - AUC ROC 0.95, AUC PR 0.70
- (Varies depending on actual test set held out)

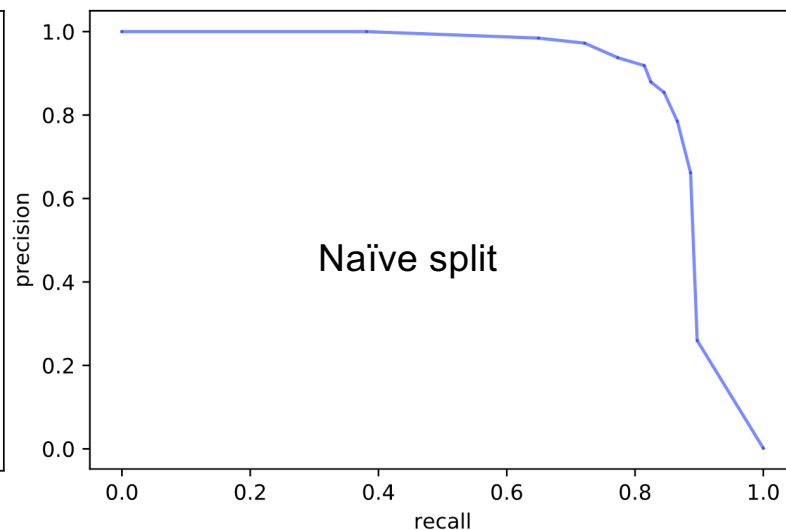
Credit card fraud with RF (10 trees)

- RFs do better than logistic: RF better at isolating minority samples
 - Naïve: ROC 0.92, AUC PR 0.84
 - Balanced split: AUC ROC 0.97, AUC PR 0.93
 - 10x upsampled training: AUC ROC 0.97, AUC PR 0.94 (doesn't really help)

credit card **ROC** curve



credit card **PR** curve



	F	T
F	56860	3
T	13	86

TY OF SAN FRANCISCO

Penalizing inappropriate confidence

- Terence's cousin went to the doctor with suspected skin cancer
- Doc was "100% positive mole was benign"
- 6 months later cousin loses most of her tricep / upper arm
- Doc's model was seriously flawed but not necessarily because of diagnosis
- The biggest mistake was the certainty of the incorrect diagnosis, as it allowed the cancer to spread
- Less certainty would've provided opportunities for more tests...
- Same concept of model assessment applies to ML models

Log loss (is both metric and loss function)

- Measures model performance when model gives probabilities, like logistic regression but RFs can give probabilities too
- Works for any number of classes; we'll do binary only
- Penalizes very confident misclassifications strongly
- Perfect score is 0 log loss, imperfection gives unbounded scores
- Let p be probability of true class, such as fraud or cancer; $1-p$ is then probability of false class, such as not fraud, not cancer
- Log loss is function of actual y and estimated p , not predicted class

Log loss continued

- Assume model gives us p (prob cancer) predicted from some x
- Case 1: true y is cancer
 - If $p=0.9$, we are confident it's cancer and rightly so: loss should be low
 - If $p=0.01$, we are confident it's NOT cancer and wrongly so: **penalize** with high (i.e., bad) loss value
- Case 2: true y is benign (not cancer)
 - If $p=0.9$, we are confident it's cancer and wrongly so: **penalize**
 - If $p=0.01$, we are confident it's NOT cancer and rightly so: loss is low
- Let loss = $penalty(p)$ if $y=1$ else $penalty(1-p)$
where $penalty(p)$ should be very high at low p (confident FP)

Log loss penalty

- $\text{loss} = \text{penalty}(p)$ if $y=1$ else $\text{penalty}(1-p)$
- Let $\text{penalty}(p) = -\log(p)$

$$\text{loss} = \frac{1}{n} \sum_{i=1}^n \begin{cases} -\log(p_i) & y = 1 \\ -\log(1 - p_i) & y = 0 \end{cases}$$

$$\text{loss} = -\frac{1}{n} \sum_{i=1}^n y_i \log(p) + (1 - y_i) \log(1 - p_i)$$

So log loss is average penalty where penalty is very high for confidence in wrong answer

