

# Measuring regressor and classifier errors

Quantifying model performance

Terence Parr

MSDS program

**University of San Francisco**

# Regressor losses/metrics

# Common regressor loss funcs / metrics

- Mean squared error  
Range  $0..∞$ , units( $y$ )<sup>2</sup>, symmetric  
e.g.,  $y=90$  vs  $95 = 5^2 = 25$


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

- Mean absolute value  
Range  $0..∞$ , units( $y$ ), symmetric  
e.g.,  $y=90$  vs  $95 = 5$

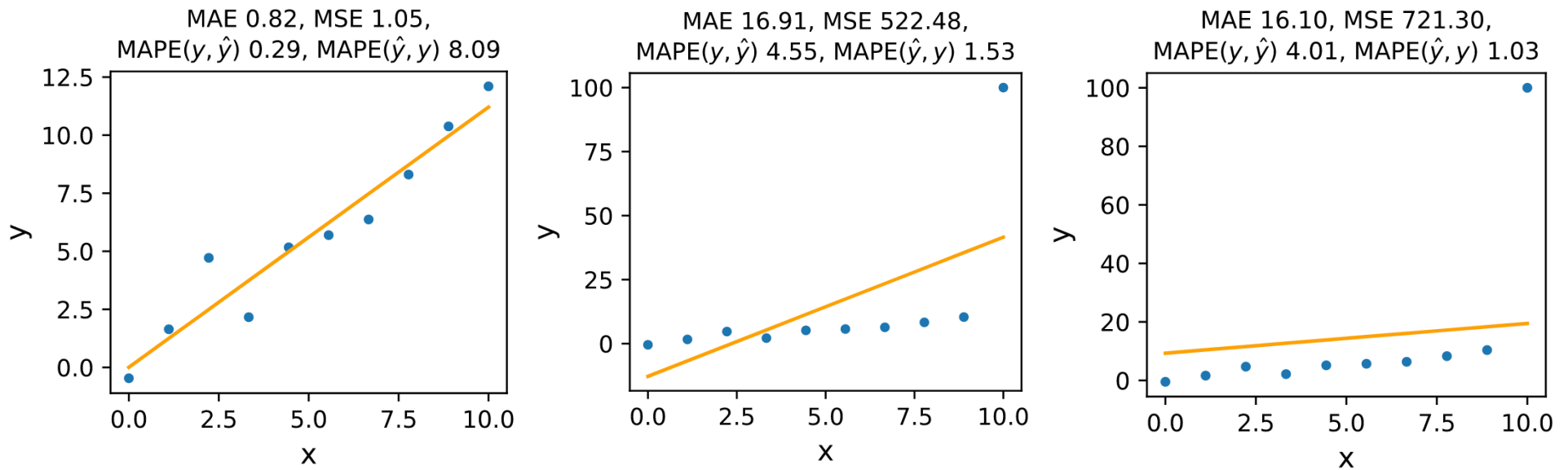
$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

- Mean absolute percentage error  
Range  $0..∞$ , unitless, **asymmetric**  
undefined if  $y=0$ ; e.g.,  $y=90$  vs  $95 = 5/90=0.0555$

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

(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 2<sup>nd</sup> model due to outlier.  
MAE isn't perfect either as it thinks 2<sup>nd</sup> and 3<sup>rd</sup> models are about the same.  
Note MAPE asymmetry!

# R<sup>2</sup>

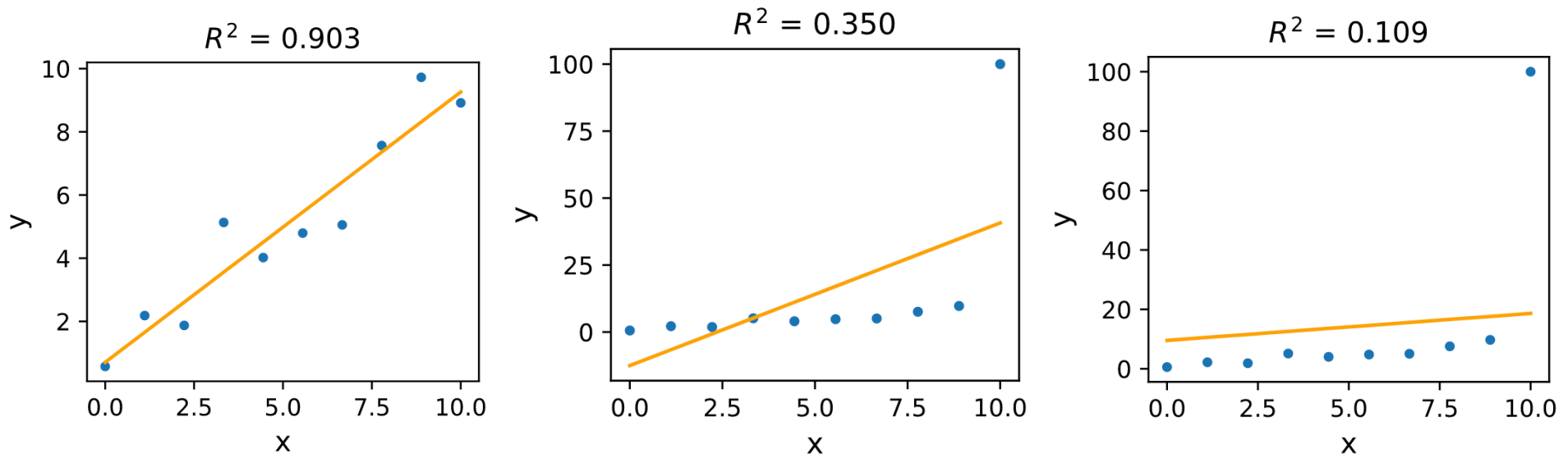
- How well our model performs compared to a “mean model”

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

- 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

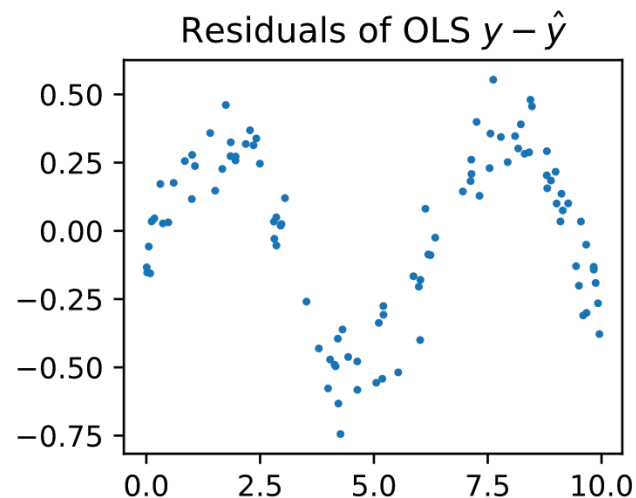
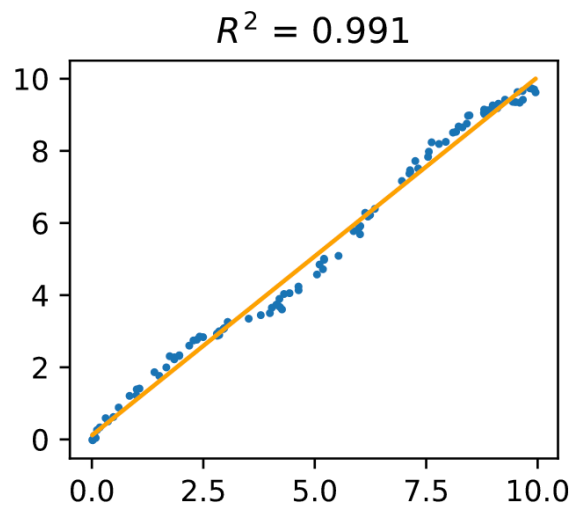
# Low $R^2$ doesn't always imply bad model

- Not best metric; here are 3 graphs with good, bad, decent fits
- Outliers can skew metrics that square deltas
- The 3<sup>rd</sup> has lowest  $R^2$  but it's a way better model than 2<sup>nd</sup>



# 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  $x, y$



But, we want  
residuals to  
look like  
random  
noise!

# 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
- MAE \$500 for a \$1,000 apartment is 0.5x or 1.5x off and a big deal but \$500 error 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 (see next slide)
- $R^2$ , MSE and in general squaring errors yields metrics that are sensitive to big deltas from even a few test records



# Symmetry in metrics

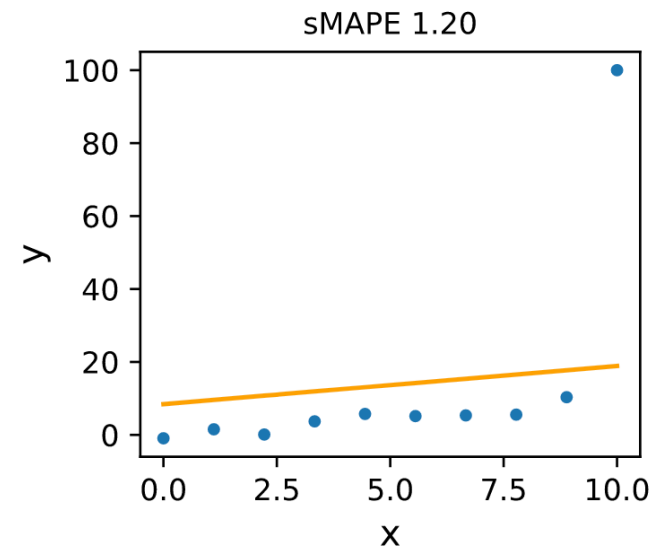
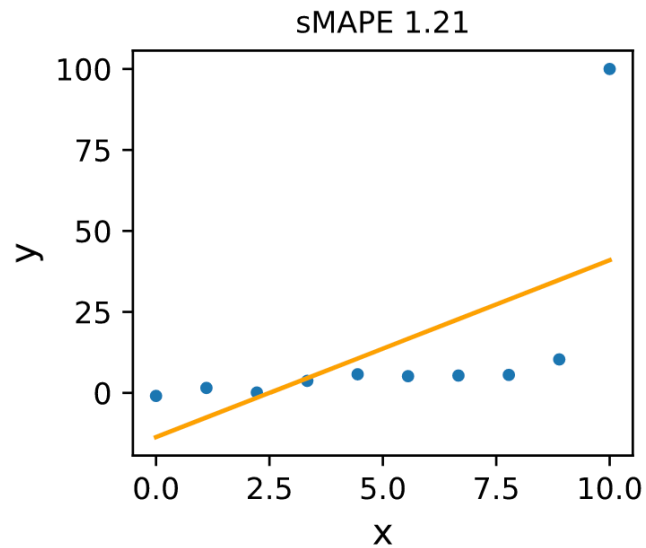
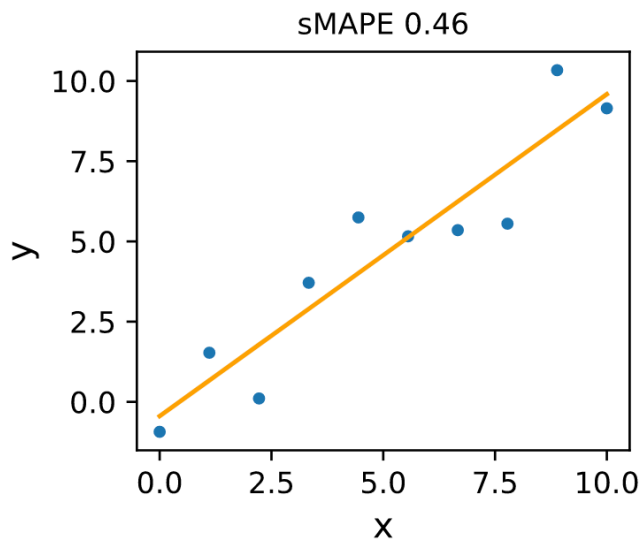
- Most metrics compute  $y - \hat{y}$  but ratio  $y/\hat{y}$  is often better
- But, most ratio-based metrics are asymmetric, which is bad!
  - If  $y=100$ ,  $\hat{y}=0.01$ :  $MAPE = 0.9999$
  - If  $y=0.01$ ,  $\hat{y}=100$ :  $MAPE = 9999$

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

- Try symmetric MAPE  
Range 0..2, unitless, symmetric,  
undefined when  $y=0$  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 losses/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>	
		<b>F</b>	<b>T</b>
<i>Actual</i>	<b>F</b>	<i>TN</i>	<i>FP</i>
	<b>T</b>	<i>FN</i>	<i>TP</i>

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

(breast cancer RF)

# True/False positive/negative rates

		<i>Predicted</i>	
		<b>F</b>	<b>T</b>
<i>Actual</i>	<b>F</b>	TN	FP
	<b>T</b>	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 2<sup>nd</sup> 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 1<sup>st</sup> 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 (sometimes overall error is high but very low for some records)
- Should focus on features that are predictive of med/high to improve model

	<b>predicted_low</b>	<b>predicted_medium</b>	<b>predicted_high</b>
<b>expected_low</b>	3749	566	83
<b>expected_medium</b>	854	418	119
<b>expected_high</b>	189	174	141

# More classifier metrics

(I like Precision/recall)

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

		Predicted		
		F	T	
Actual	F	35	3	
	T	1	75	

precision =  $75/(75+3)$

		Predicted		
		F	T	
Actual	F	35	3	
	T	1	75	

recall =  $75/(75+1)$

# ROC and PR curves

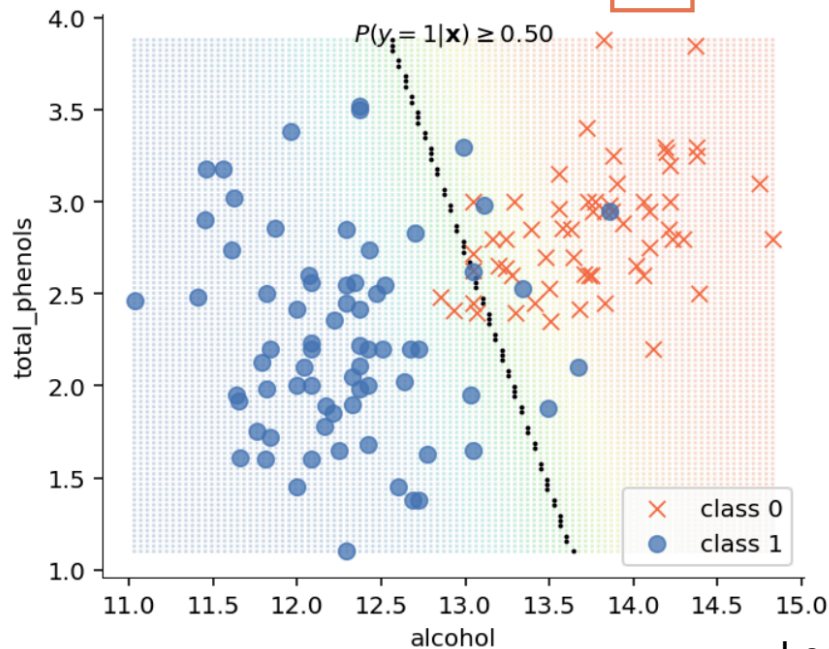


# Classifier: $P(y = 1|\mathbf{x})$ and a threshold

- Different thresholds on the  $P(y = 1|\mathbf{x})$  model output probabilities give different classifiers:

Accuracy 119/130=0.92

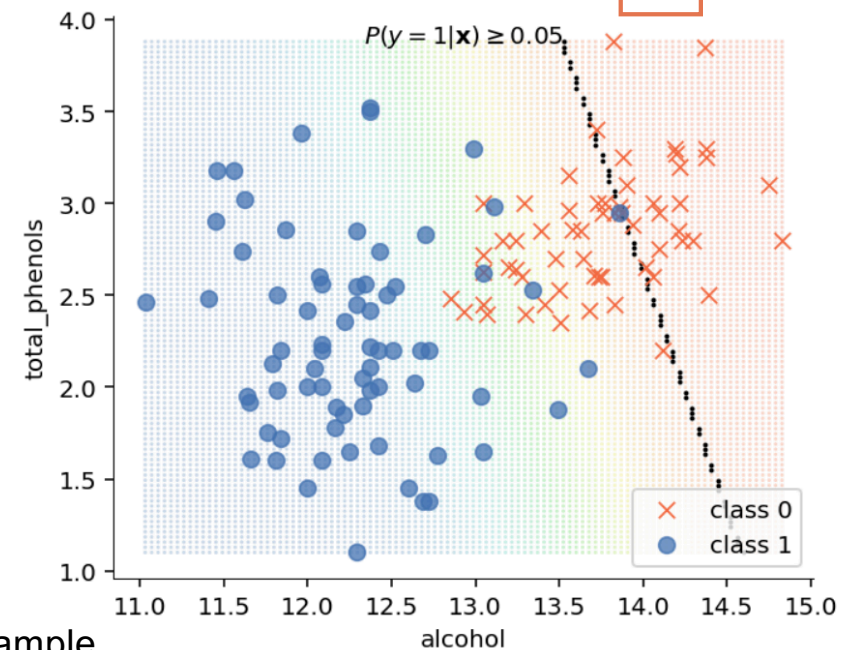
(threshold, precision, recall) = (0.50, 0.941, 0.901)



Logistic example

Accuracy 94/130=0.72

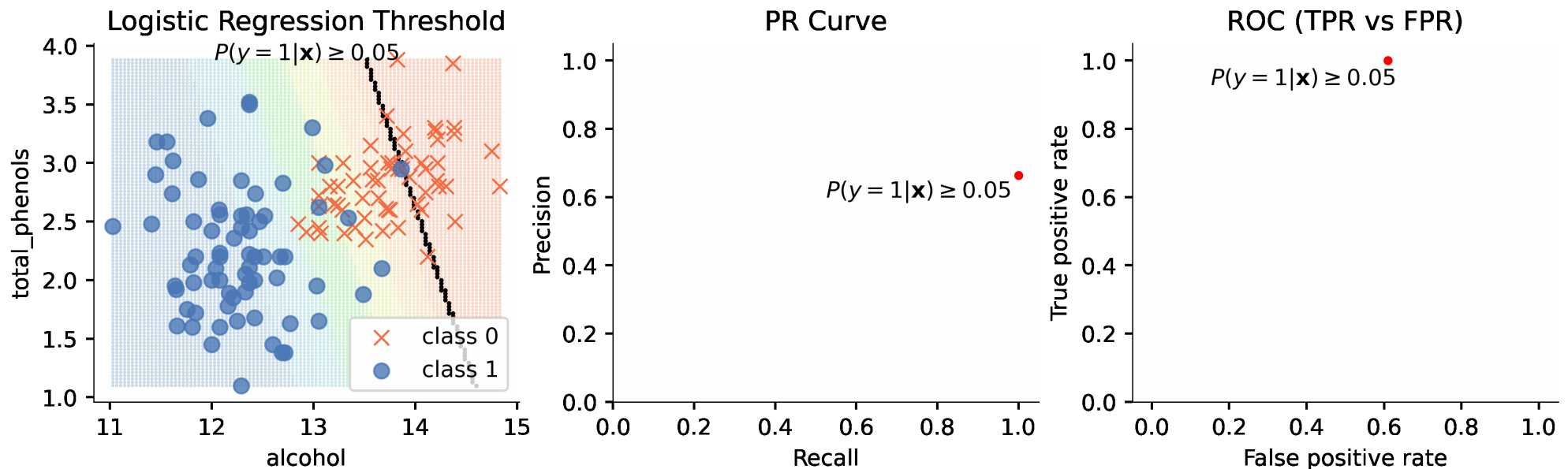
(threshold, precision, recall) = (0.05, 0.664, 1.000)



ROC == receiver operator curve  
PR == precision-recall

# Animated ROC or PR curves

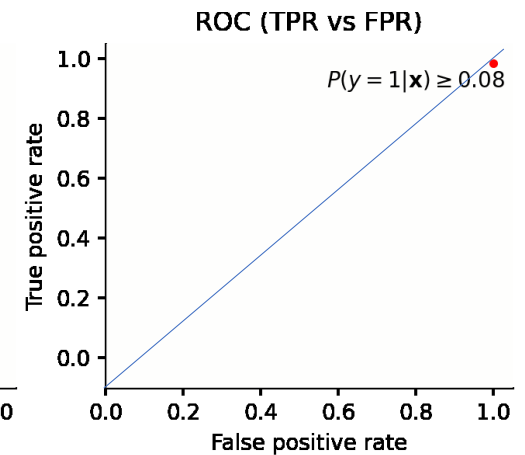
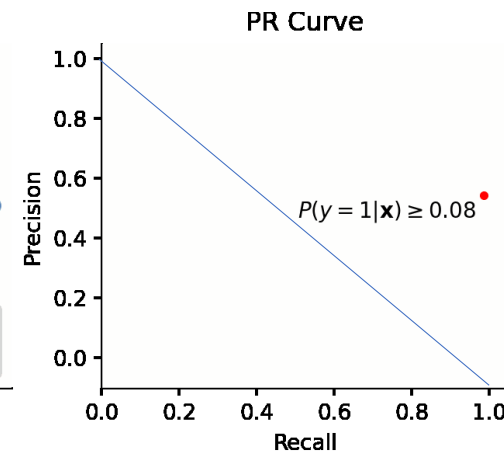
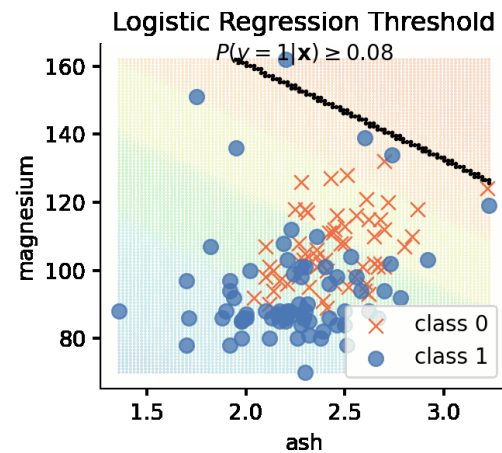
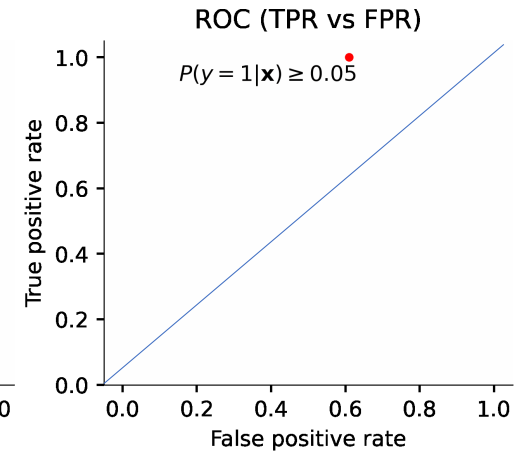
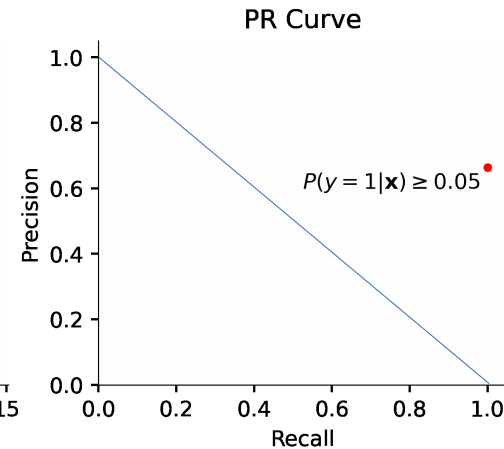
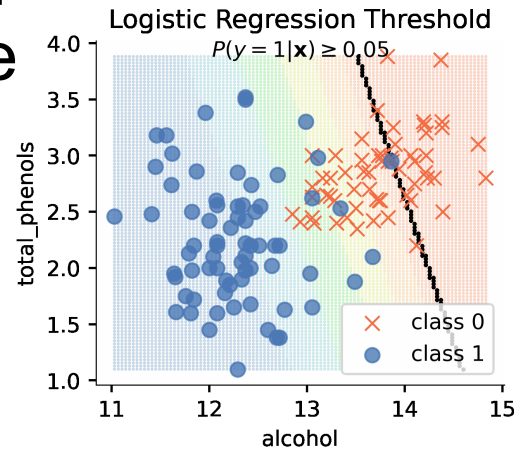
- Shift  $P(y = 1|\mathbf{x})$  threshold from 0.0 to 1.0, compute and plot (Precision, Recall) or (TPR/recall, FPR) coordinates for each classifier resulting from each threshold:



See <https://github.com/parr/msds621/blob/master/notebooks/assessment/PR-ROC-curves.ipynb>

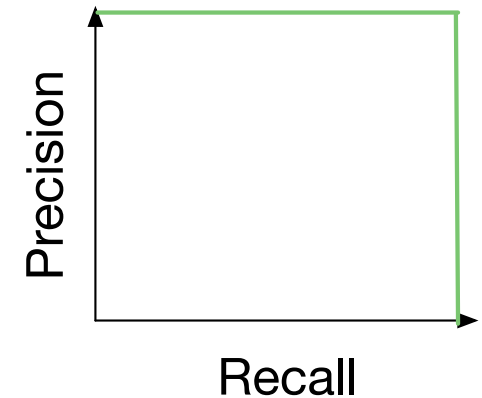
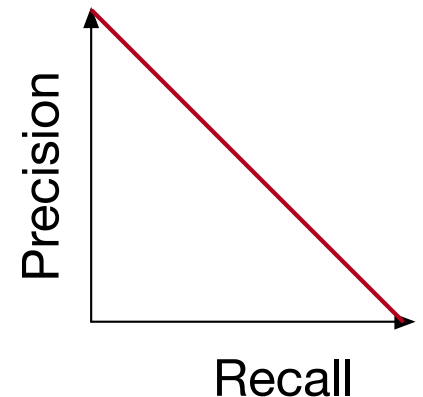
# Compare with less predictive features

- Notice how the less predictive features yield a partition with much more overlap
- The curves for less well separated classes are much closer to the diagonal than for the good features



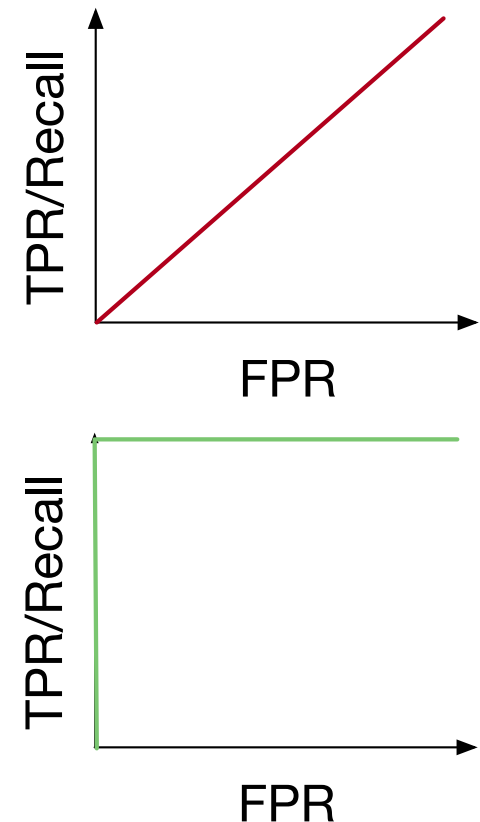
# AUC precision-recall curve

- Goal: Conjure up scalar from curve using AUC (*area under the curve*)
- When distributions overlap exactly, moving threshold around gives -45 degree line, AUC=0.5
- When distributions are completely separate, AUC=1
- Any curve above 45° diagonal is better than overlapping distributions



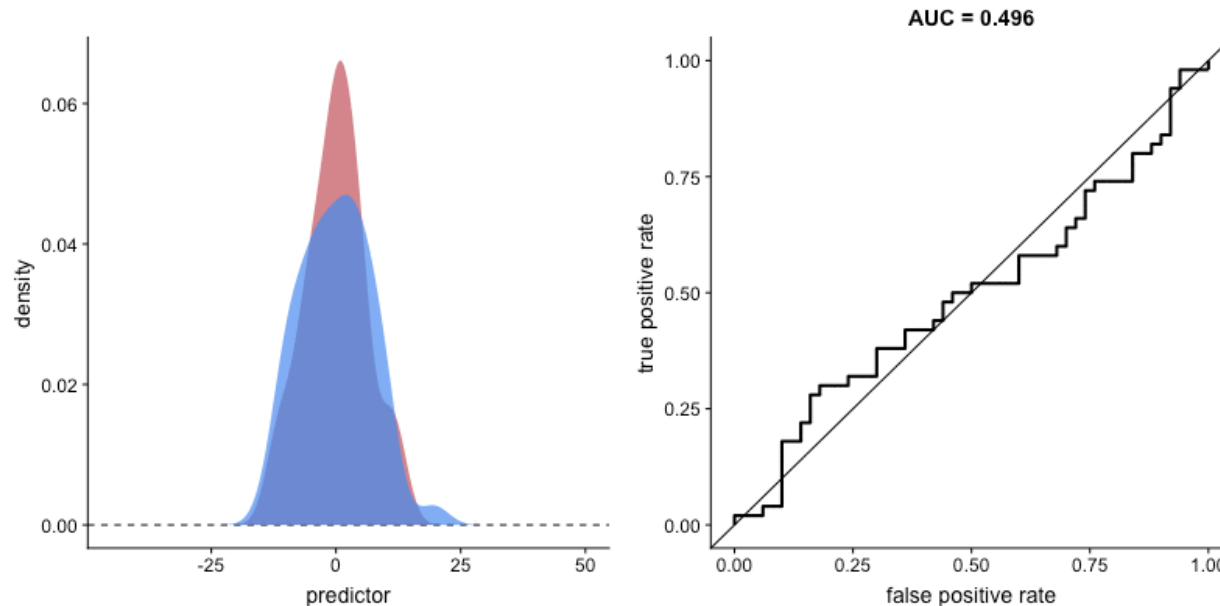
# AUC ROC

- Goal: Conjure up scalar from curve using AUC (*area under the curve*)
- 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,  $AUC=1$
- Any curve above  $45^\circ$  diagonal is better than overlapping distributions



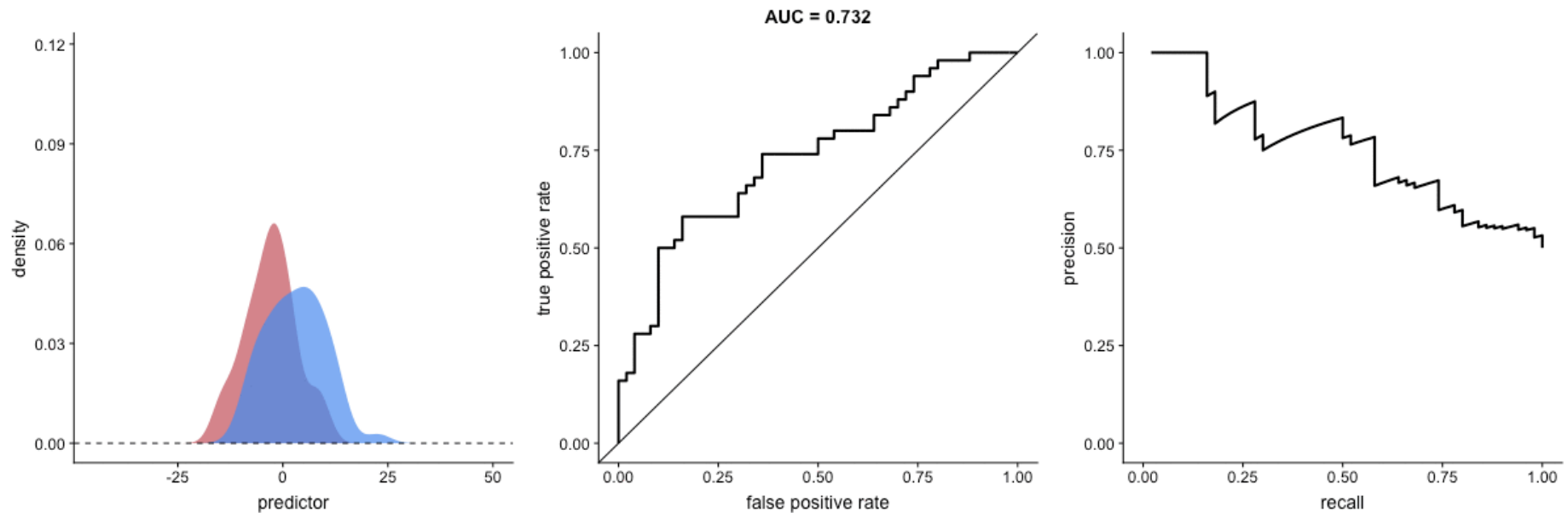
# Why ROC and PR curves?

- Curves indicate quality of model
- More specifically, curves indicate how well/easily a model can separate class 0 from class 1 instances



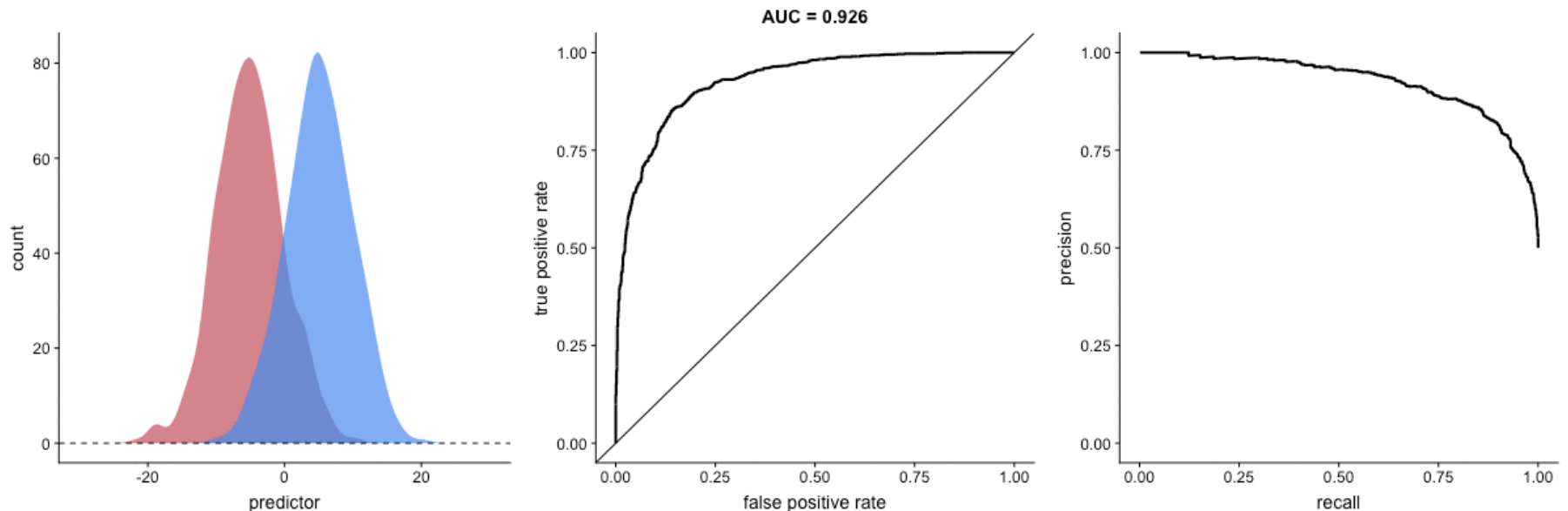
# ROC vs Precision-Recall curve

- As variance tightens, with more overlap, ROC AUC goes up whereas PR goes down; ROC doesn't have good behavior



# ROC vs PR curve: Imbalanced classes

- 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



Animation credits: [https://github.com/dariyasdykova/open\\_projects/blob/master/ROC\\_animation/README.md](https://github.com/dariyasdykova/open_projects/blob/master/ROC_animation/README.md)



# Upsampling minority class(es) in imbalanced data sets

Naïve use of `train_test_split()` @ 20%

# Imbalanced dataset: Kaggle credit card fraud

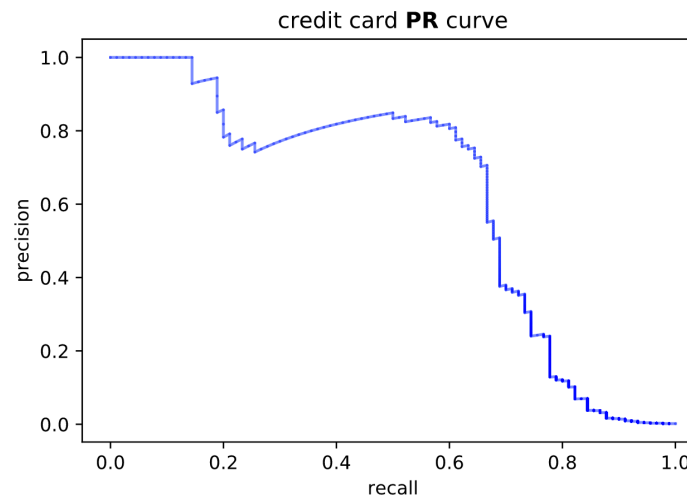
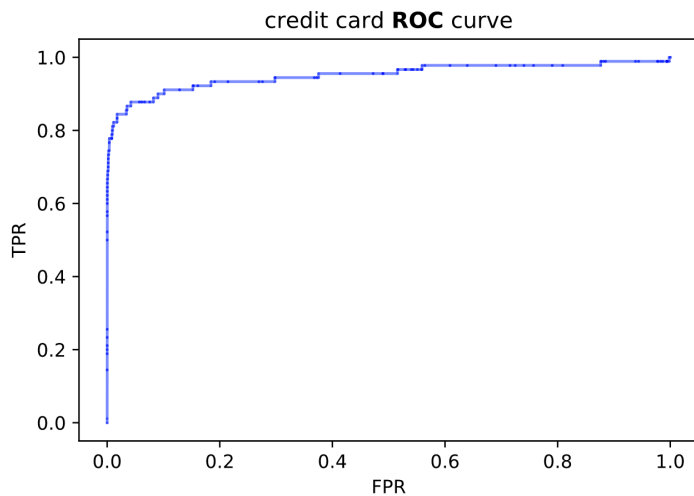
- Num anomalies 492/284807 = 0.17%
- L2 Regularized **LogisticRegression** model
- Accuracy 1.00, AUC ROC 0.93
- F1 0.69, AUC PR 0.62

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)
```

Do NOT upsample before splitting out test set; you'll leak test data into training set!  
Tweet/paper on how this can cause data leakage:

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

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



UNIVERSITY OF SAN FRANCISCO

# Upsampling fraud 10x in training data

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

No upsampling

	F	T
F	56840	16
T	42	64

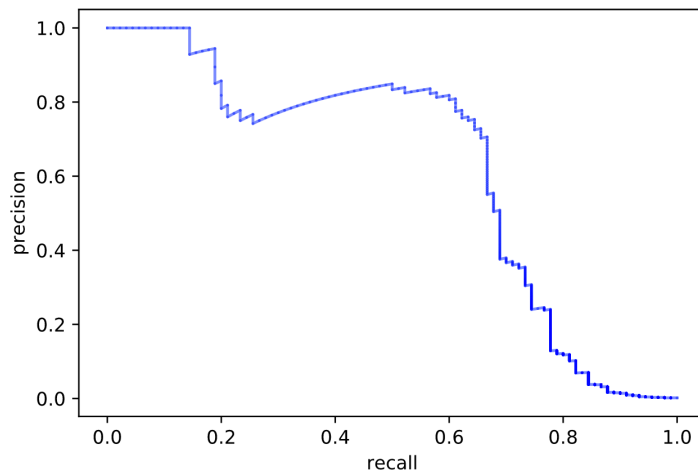
F1 0.69, AUC PR 0.62

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

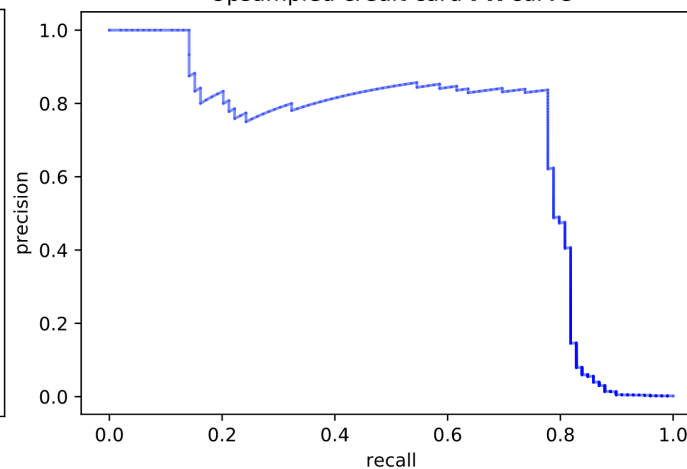
predicted

	F	T
actual F	56827	36
actual T	22	77

credit card PR curve



Upsampled credit card PR curve



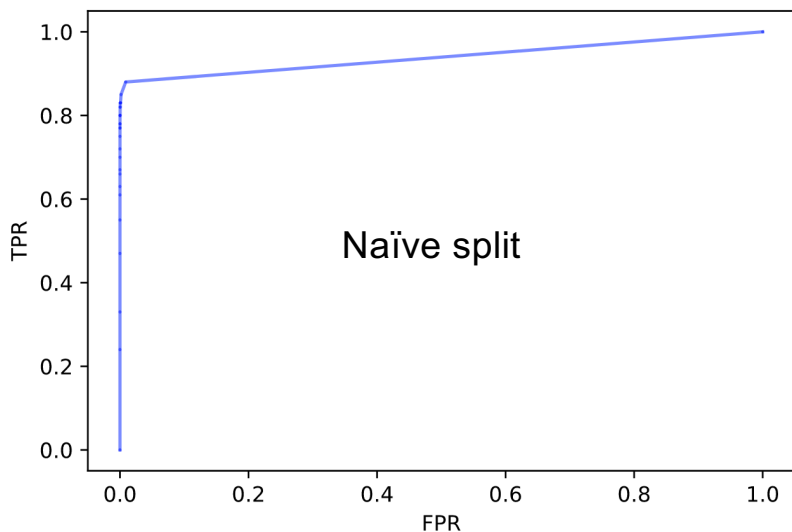
# L2 Logistic Regression summary

- 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)

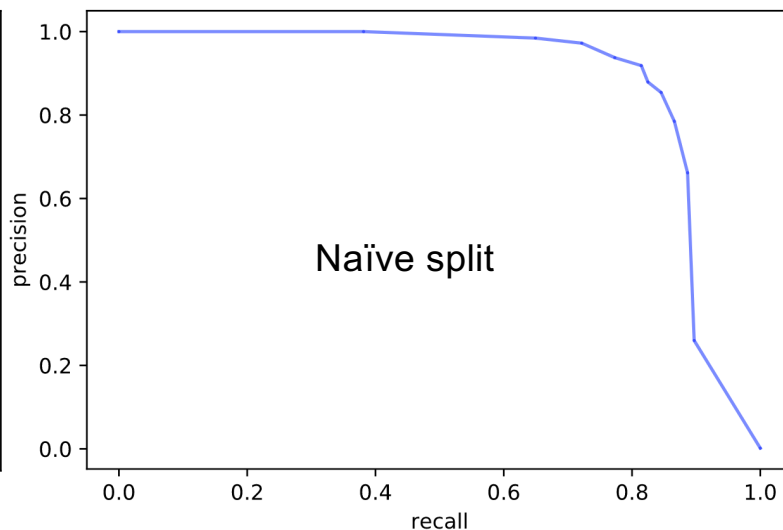
# Credit card fraud with RF (10 trees)

- 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  $penalty(1-p)$  if  $y=0$   
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

