Applied Machine Learning

Week 3

Grant Schoenebeck

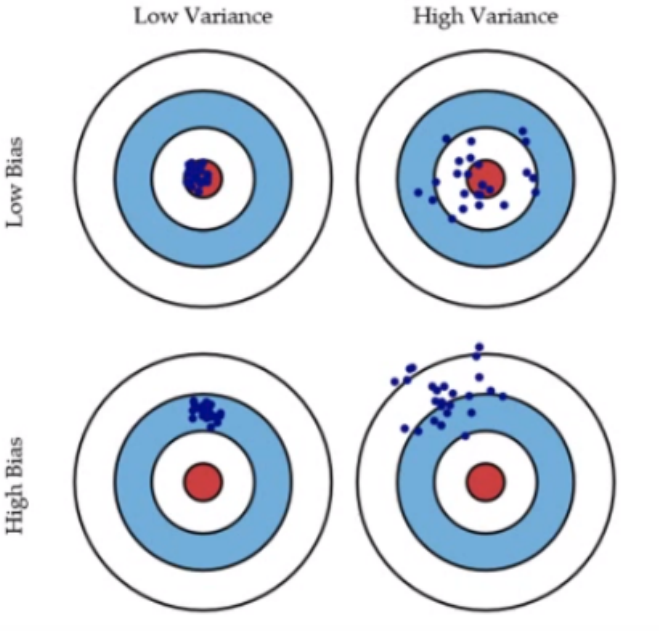
Some slides adapted from material courtesy of Andrew W. Moore @ CMU. See http://www.cs.cmu.edu/~awm/tutorials.html

Outline

- Bias-Variance Decomposition of Error
- Regularization
  - *Ridge, Lasso*
- Robust Regression
- Logistic Regression
- Measuring accuracy
  - *ROC, F1, confusion matrix*

Bias-Variance Decomposition of Error

Bias vs variance around a single "target "point



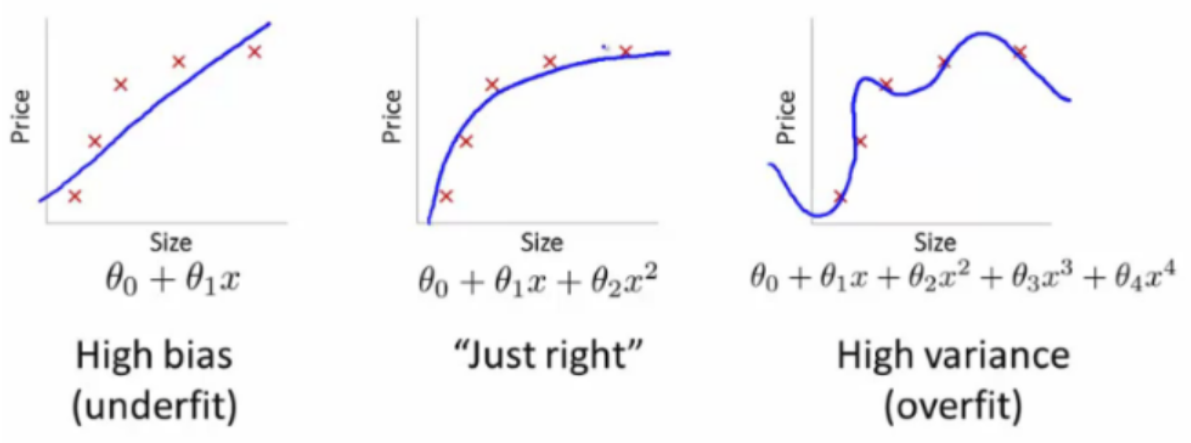
Bias-Variance

- **Error can be decomposed into two orthogonal components: Bias and Variance**
- **Process:**
  - *Fix a point **x*** with label **y***.*
  - *Drawn n points **X, y** . Train predictor $F_{X,y}$. Then $F_{X,y}(x^*)$ is a random variable (over the randomness of training data sample) which predicts **y***.*
  - *Bias =* $E(F_{X,y}(x^*) - y^*)$
  - *Variance =* $Var\left(F_{X,y}(x^*)\right) = E(F_{X,y}(x^*) - E(F_{X,y}(x^*)))$
  - *Mean Squared Error = Variance + Bais*^{2}

Bias-Variance

- *Bias* = $E(F_{\boldsymbol{X},y}(x^*) - y^*)$
  - *Typically due to underfitting. Classifier cannot accurately predict all points simultaneously.*
- *Variance* = $Var\left(F_{\boldsymbol{X},y}(x^*)\right) = E(F_{\boldsymbol{X},y}(x^*) - E\left(F_{\boldsymbol{X},y}(x^*)\right)^2$
  - *Would like that we learn the same $F_{\boldsymbol{X},y}(\cdot)$ each time (the truth).*
  - *Conversely if $F_{\boldsymbol{X},y}(\cdot)$ is highly dependent on which training data were chosen, it cannot consistently predict y^*.*

Bias-variance tradeoffs



How to restrict the complexity of the model family used to fit the data?
We'll learn about something called regularization…

Regularization

Problems with ordinary least-squares regression

- Large coefficients w_i often yield large variance
  - *huge weights can cancel each other out.*
- We prefer models low weights.
- We prefer models with few explanatory variables.

Ridge Regression

- OLS but adds a penalty for large variations in *w* parameters

$$RSS_{RIDGE}(\boldsymbol{w}, b) = \sum_{\{i=1\}}^{N} (\boldsymbol{y}_i - (\boldsymbol{w} \cdot \boldsymbol{x}_i + b))^2 + \alpha \sum_{\{j=1\}}^{p} w_j^2$$

- Penalty only for learning.
- Regularization: prefers models with small complexity
  - *Measured by sum of squares of w*
  - *"L2 regularization"*
- α parameter controls effect of regularization.
  - *Higher α increases preference of simpler models.*

Ridge Regression

- **WARNING: You must normalize inputs when you use Ridge Regression, why?**

$$RSS_{RIDGE}(\boldsymbol{w}, b) = \sum_{\{i=1\}}^{N} (\boldsymbol{y}_i - (\boldsymbol{w} \cdot \boldsymbol{x}_i + b))^2 + \alpha \sum_{\{j=1\}}^{p} w_j^2$$

Think, Pair, Share

- Ridge Regression Trades off:
- A) decreases Variance but may increase Bias
- B) decreases Bias for may increase Variance

Lasso regression

- **OLS but adds a penalty for large variations in *w* parameters**

$$RSS_{LASSO}(\boldsymbol{w}, b) = \sum_{\{i=1\}}^{N} (\boldsymbol{y}_i - (\boldsymbol{w} \cdot \boldsymbol{x}_i + b))^2 + \alpha \sum_{\{j=1\}}^{p} |w_j|$$

- **Penalty only for learning.**
- **Regularization: prefers models with small complexity**
  - *Measured by sum of squares of w*
  - *"L1 regularization"*
- **α parameter controls effect of regularization.**
  - *Higher alpha increases preference of simpler models.*

Lasso Regression

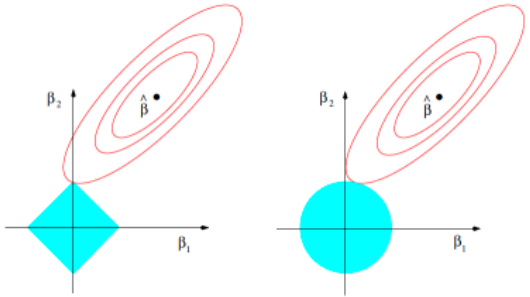
- **WARNING: You must normalize inputs when you use Lasso Regression, why?**

$$RSS_{LASSO}(\boldsymbol{w}, b) = \sum_{\{i=1\}}^{N} (\boldsymbol{y}_i - (\boldsymbol{w} \cdot \boldsymbol{x}_i + b))^2 + \alpha \sum_{\{j=1\}}^{p} |w_j|$$

Ridge vs Lasso

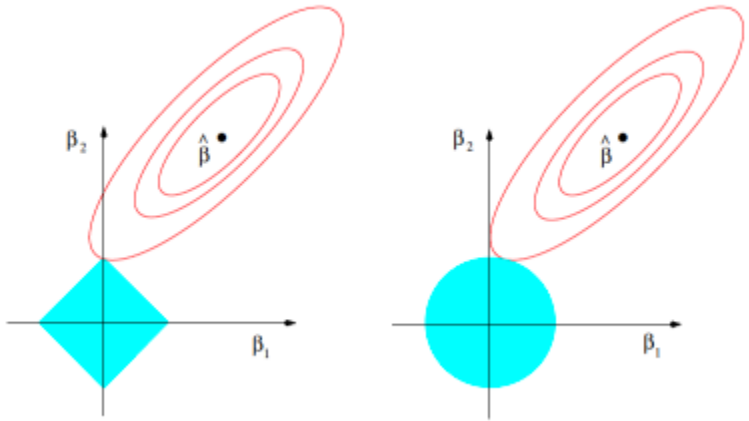
$$RSS_{RIDGE}(\boldsymbol{w}, b) = \sum_{\{i=1\}}^{N} (\boldsymbol{y}_i - (\boldsymbol{w} \cdot \boldsymbol{x}_i + b))^2 + \alpha \sum_{\{j=1\}}^{p} w_j^2$$

$$RSS_{LASSO}(\boldsymbol{w}, b) = \sum_{\{i=1\}}^{N} (\boldsymbol{y}_i - (\boldsymbol{w} \cdot \boldsymbol{x}_i + b))^2 + \alpha \sum_{\{j=1\}}^{p} |w_j|$$



Ridge vs Lasso

- **Use Ridge:**
  - *Many small/medium sized effects*
- **Use Lasso:**
  - *Only a few variables with medium/large effect*
  - *Think "Lasso" the important variables.*



Ridge and Lasso

- Part of a class of methods called "Shrikage" methods
- Can be computed efficiently using convex methods.

Lasso Regression on the Communities and Crime Dataset

For alpha = 2.0, 20 out of 88 features have non-zero weight.

Top features (sorted by abs. magnitude):

```
PctKidsBornNeverMar, 1488.365 # percentage of kids born to people who never married
PctKids2Par, -1188.740 # percentage of kids in family housing with two parents
HousVacant, 459.538  # number of vacant households
PctPersDenseHous, 339.045 # percent of persons in dense housing (more than 1 person/room)
NumInShelters, 264.932 # number of people in homeless shelters
```

Adding non–polynomial Features

Examples:

- Population of a city versus number of billionaires?
- Revenue of a company versus number of vice presidents?
- Number of employees versus layers of management?

Might try to add sublinear features:

Also, can add transformations of data:

- QuantileTransformer (transform to uniform)
  - Can also output normal

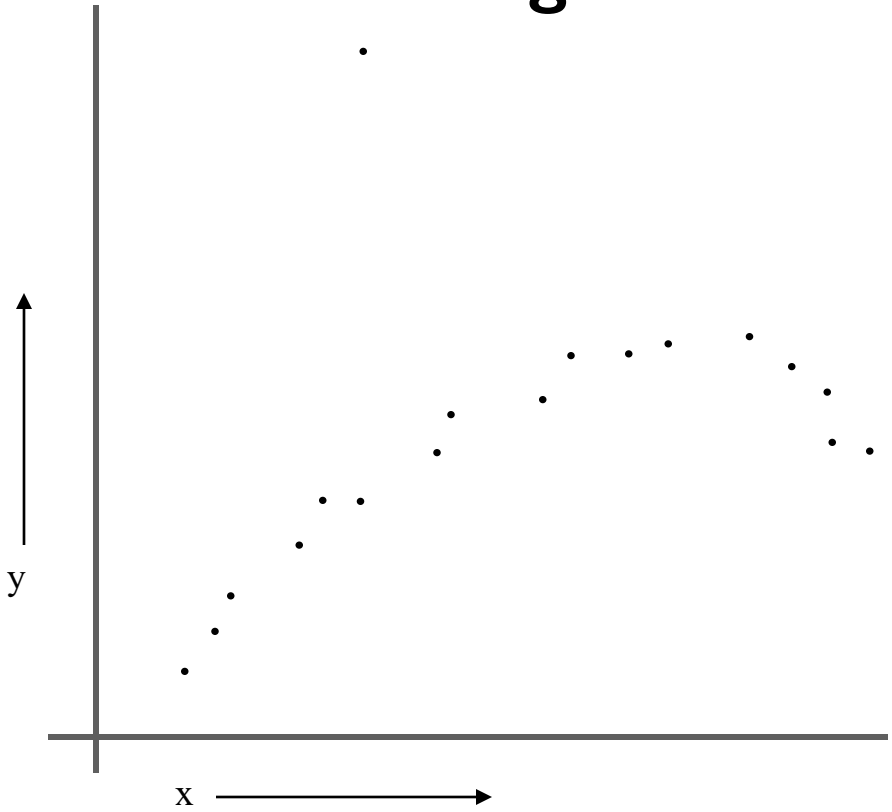
Non-linear Regression

$$y_i \sim N(\sqrt{w + x_i}, \sigma^2)$$

- How to learn w?

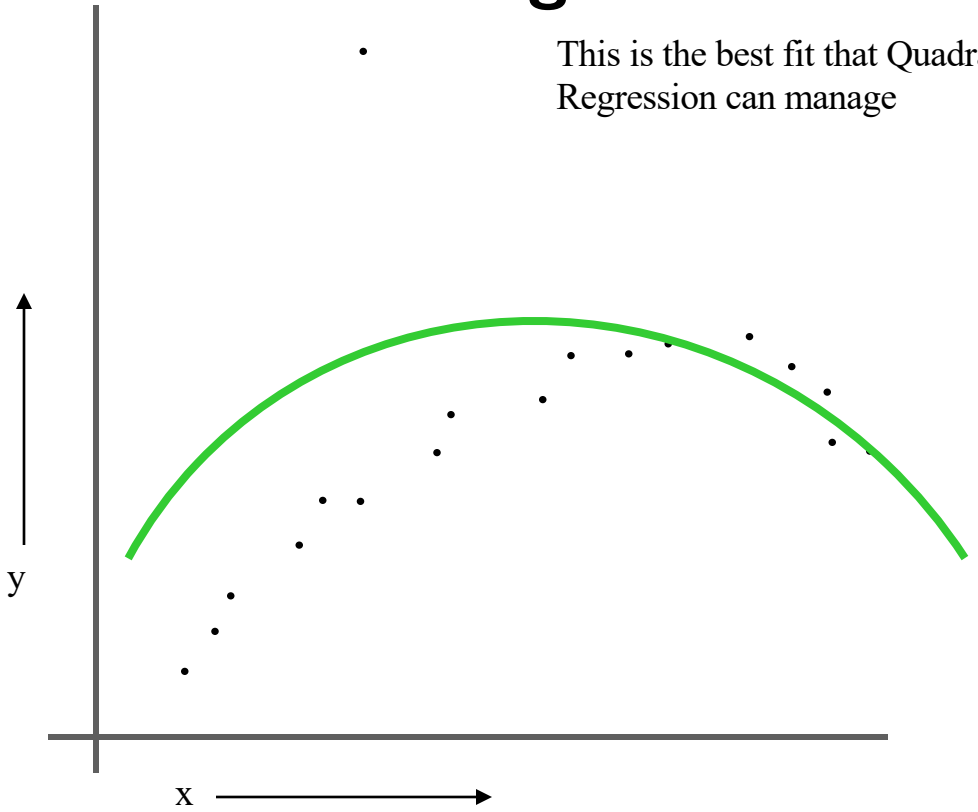
$$y_i \sim N(\sqrt{w + x_i}, \sigma^2)$$

Robust Regression



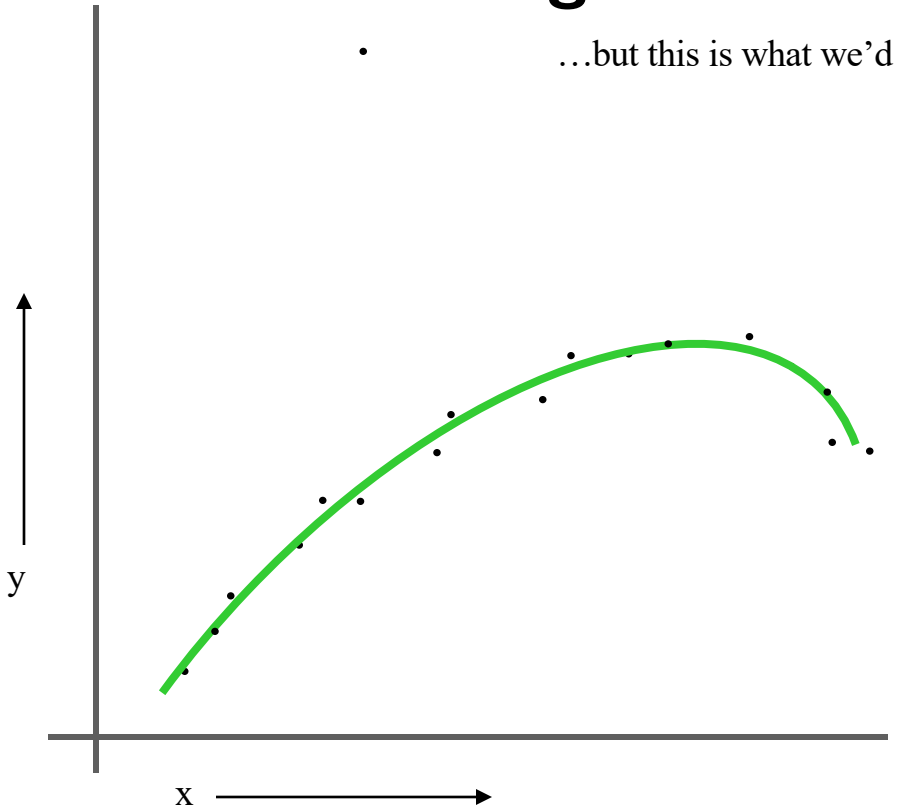
Robust Regression

This is the best fit that Quadratic Regression can manage



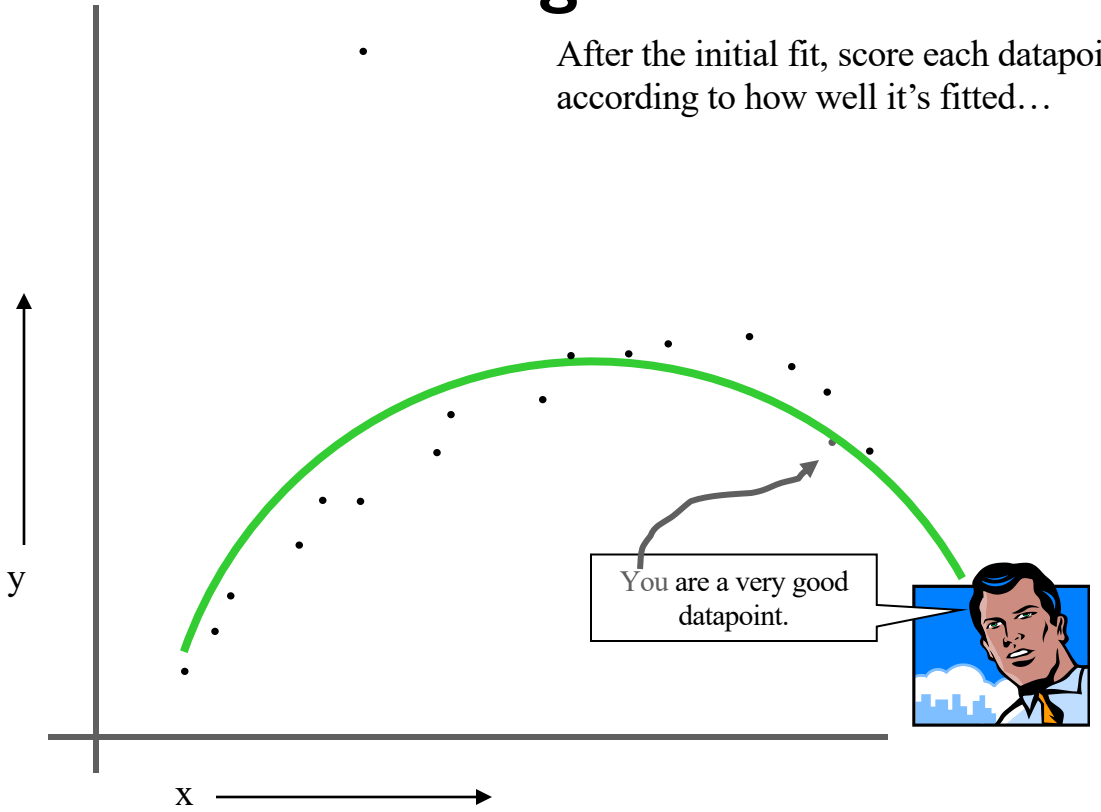
Robust Regression

- …but this is what we'd probably prefer



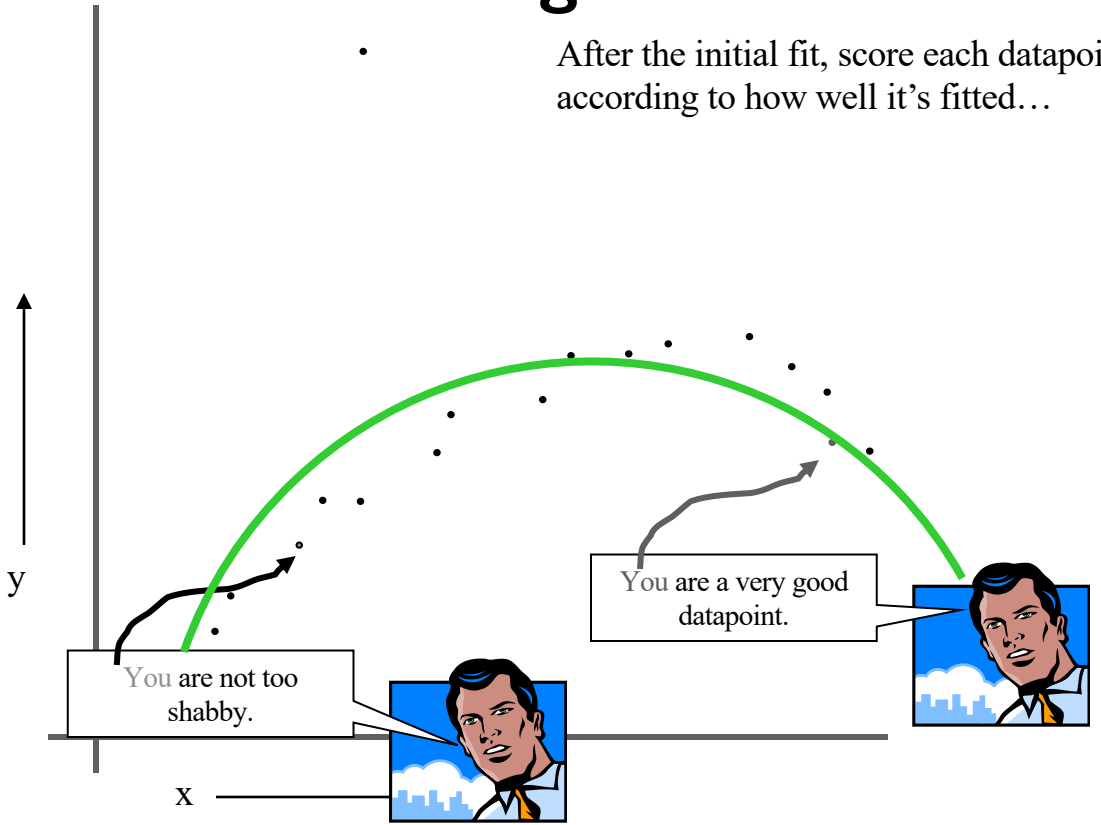
Robust Regression

After the initial fit, score each datapoint according to how well it's fitted…



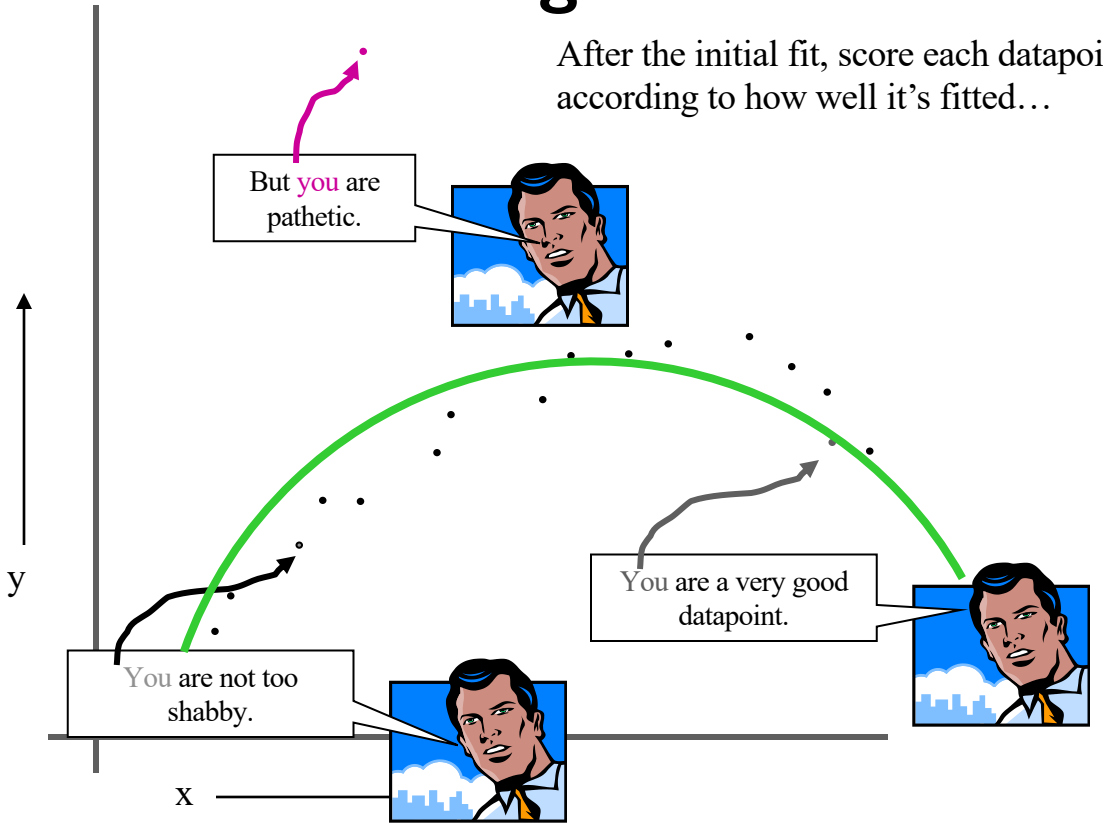
Robust Regression

After the initial fit, score each datapoint according to how well it's fitted…

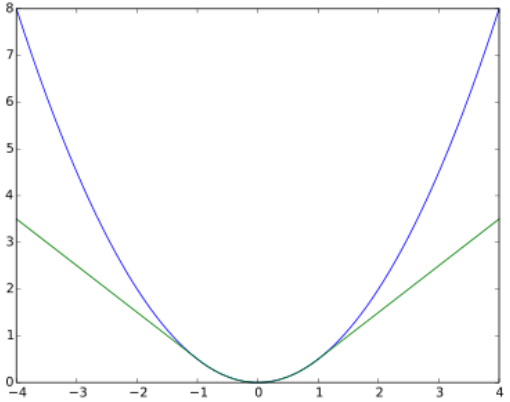


Robust Regression

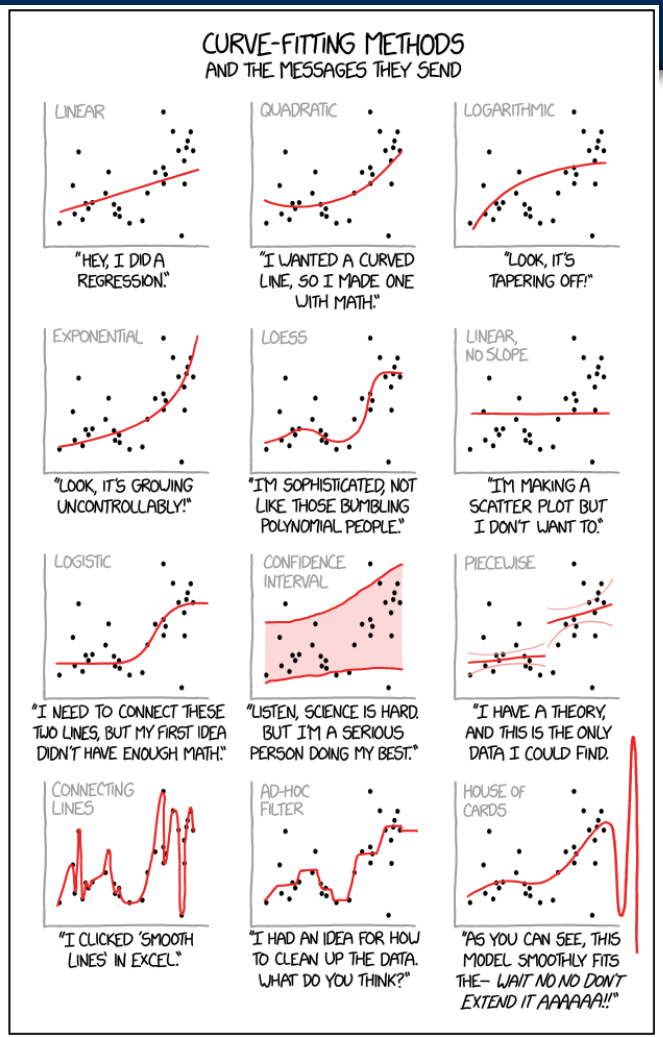
After the initial fit, score each datapoint according to how well it's fitted…



Robust Regression



- **Solutions:**
  - Use HuberRegressor
    - Linear loss for outliers.
    - Implimented in SciKit Learn
  - Use RANSAC (RANdom SAmple Consensus)
    - Implemented in SciKit Learn
    - Fits to subset of points, then only uses "good points"
  - MLE for a different model (LOESS robust regression)
    - *Uses EM algorithms*



https://m.xkcd.com/2048/

alt-text:

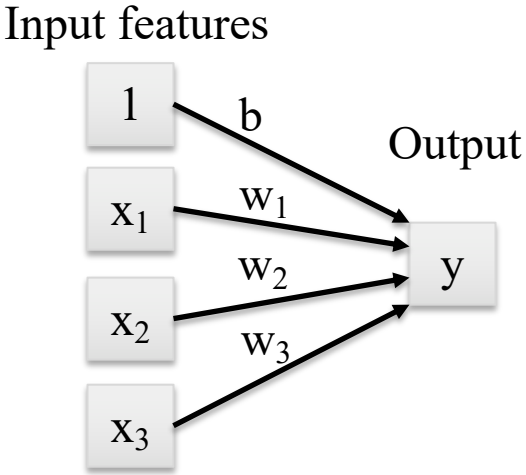
> Cauchy-Lorentz: "Something alarmingly mathematical is happening, and you should probably pause to Google my name and check what field I originally worked in."

Logistic regression

Is Logistic Regression Really Regression?

- **Logistic Regression is a "soft" classifier.**
  - *Like the weather, it provides a forecast.*
  - *Since this is a real number, called a regression.*
  - *You can imagine cases where this is more or less appropriate:*
    - *More: Not enough Given past 4 rock/paper/scissors actions, predict next action.*
    - *Given a picture of a rock, is it quartz?*

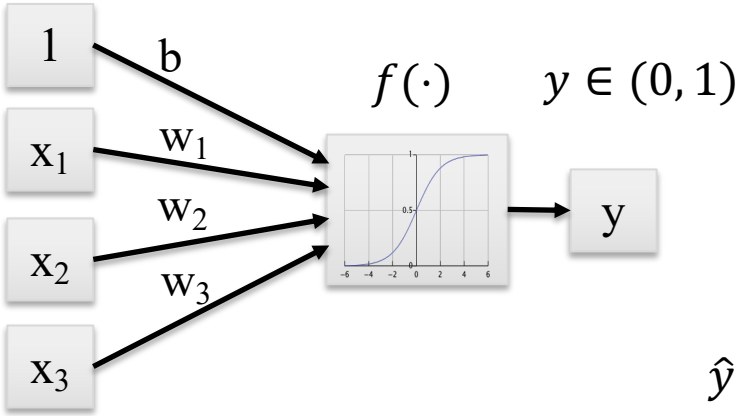
Recall: Linear Regression

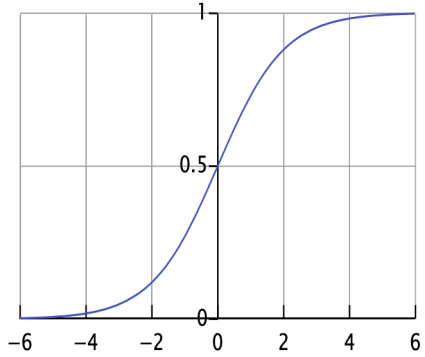


$$\hat{y} = \hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n$$

Linear models for classification: Logistic Regression

Input features



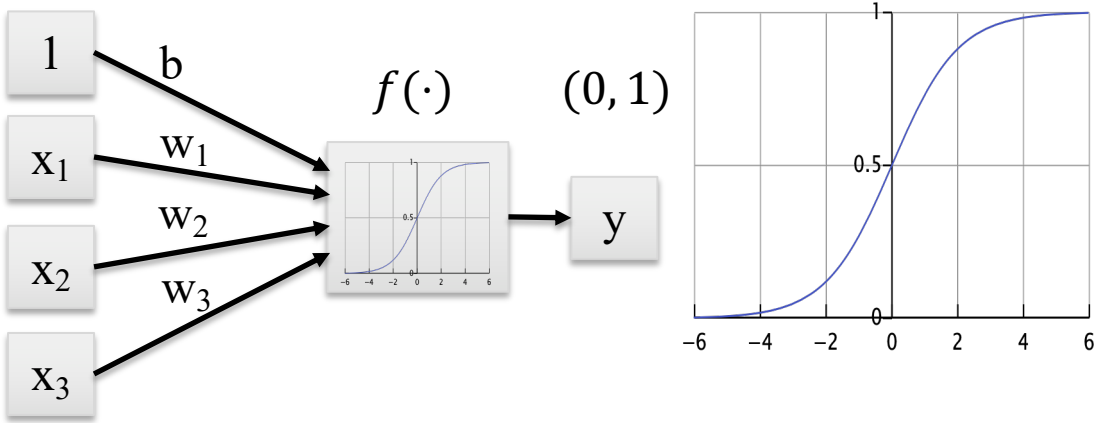


$$\hat{y} = \text{logistic}(\hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n)$$

$$= \frac{1}{1 + \exp\left[-(\hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n)\right]}$$

Linear models for classification: Logistic Regression

Input features



The logistic function transforms real-valued input to an output number y between 0 and 1, interpreted as the probability the input object belongs to the positive class, given its input features $(x_0, x_1, \ldots, x_n)$

$$\hat{y} = \text{logistic}(\hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n)$$

$$= \frac{1}{1 + \exp\left[-(\hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n)\right]}$$

Why?

- **Likelihood ratio**

$$\frac{\Pr[A]}{\Pr[Not\ A]} = x \text{ implies } \Pr[A] = \frac{1}{1+x} \text{ and } \Pr[not\ A] = \frac{x}{1+x}$$

Type equation here.

- **Thus Log Likelihood ratio of y implies**

$$=\Pr[A] = \frac{1}{1+exp(y)}$$

Why?

- Likelihood ratio (assuming independence)

$$= \frac{\Pr[A|X_1, \ldots, X_k]}{\Pr[not\ A|X_1, \ldots, X_k]} = \frac{\Pr[A]}{\Pr[Not\ A]} \prod \frac{\Pr[X_i|A]}{\Pr[X_i|Not\ A]}$$

Log Likelihood ratio is

$$= \log\left(\frac{\Pr[A]}{\Pr[Not\ A]}\right) + \sum log\left(\frac{\Pr[X_i|A]}{\Pr[X_i|Not\ A]}\right)$$

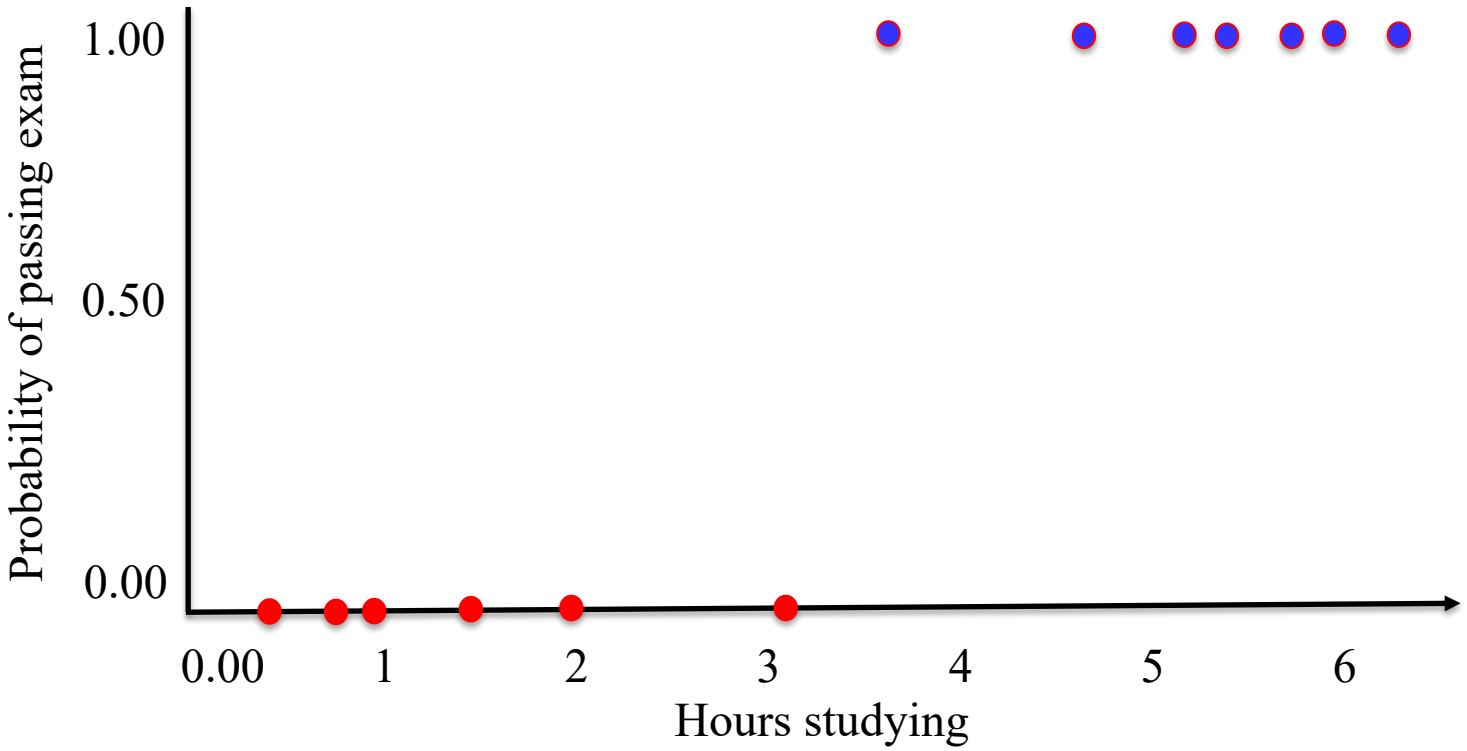
Why?

- **Let** $b = \log(\Pr[A] / \Pr[not\ A])$
- **Let** $w_i\ x_i = \log\left(\frac{Pr[X_i = x_i|A]}{Pr[X_i = x_i|not\ A]}\right)$
- $Pr[A] = \frac{1}{1+ likelihood} = \frac{1}{1+exp(wx+b)}$

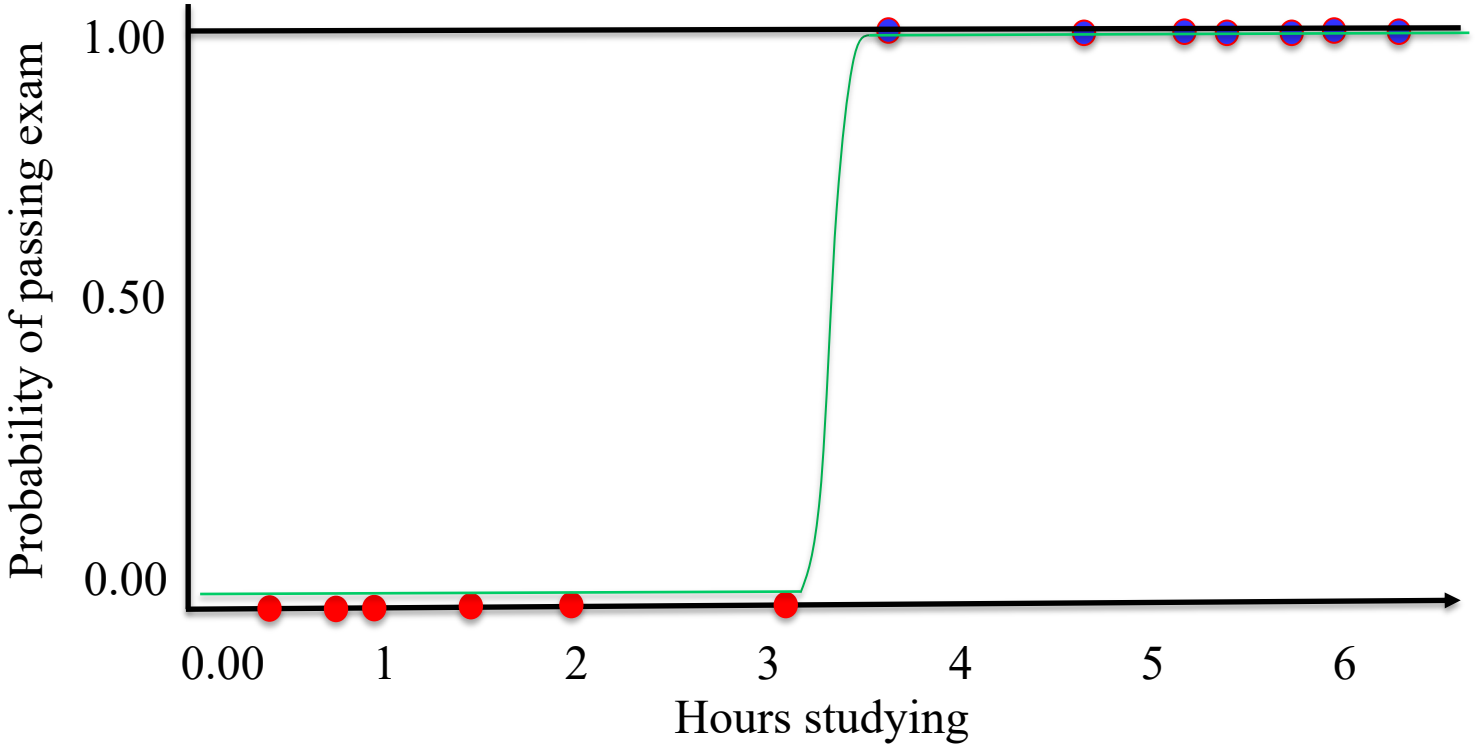
Take Away

- Assumes loglikelihood = $log\left(\frac{Pr[A|X=x]}{Pr[not\ A|X=x]}\right)$ is linear in x

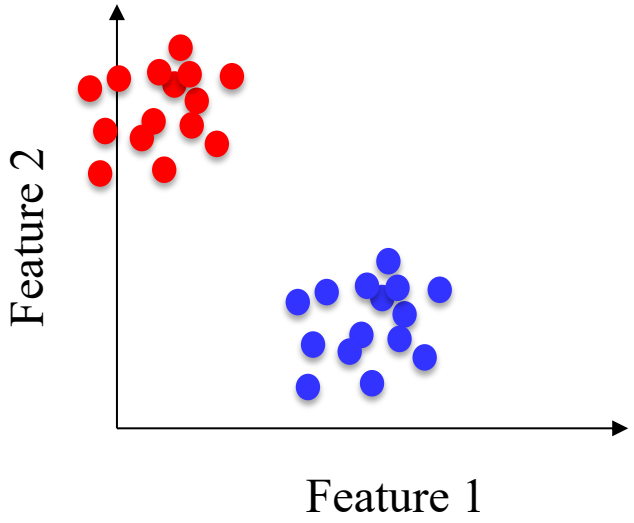
Linear models for classification: Logistic Regression



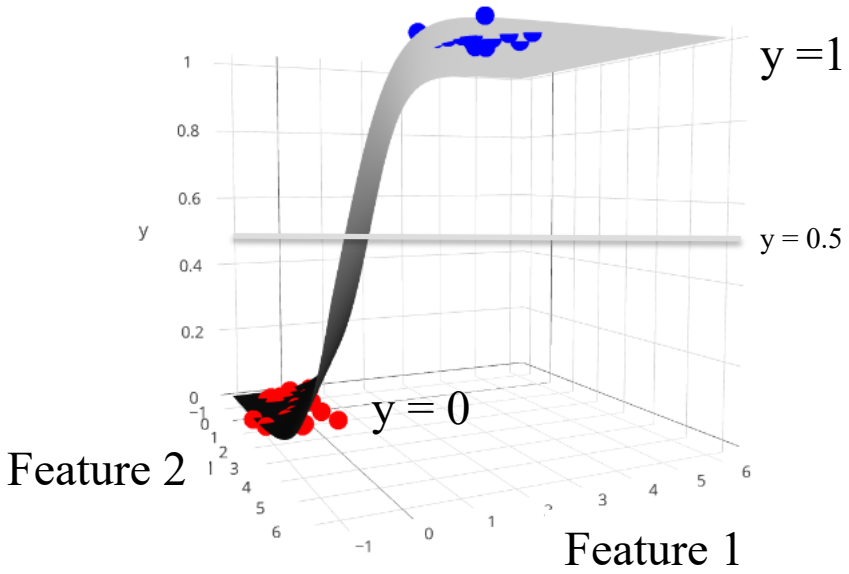
Linear models for classification: Logistic Regression



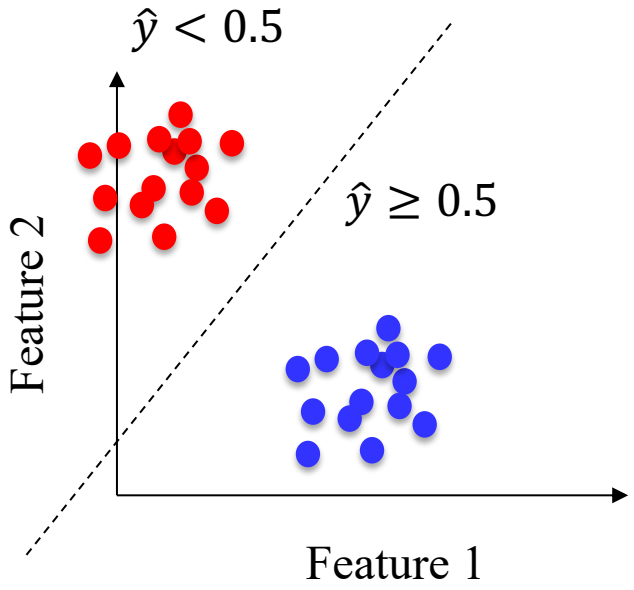
Logistic Regression for binary classification



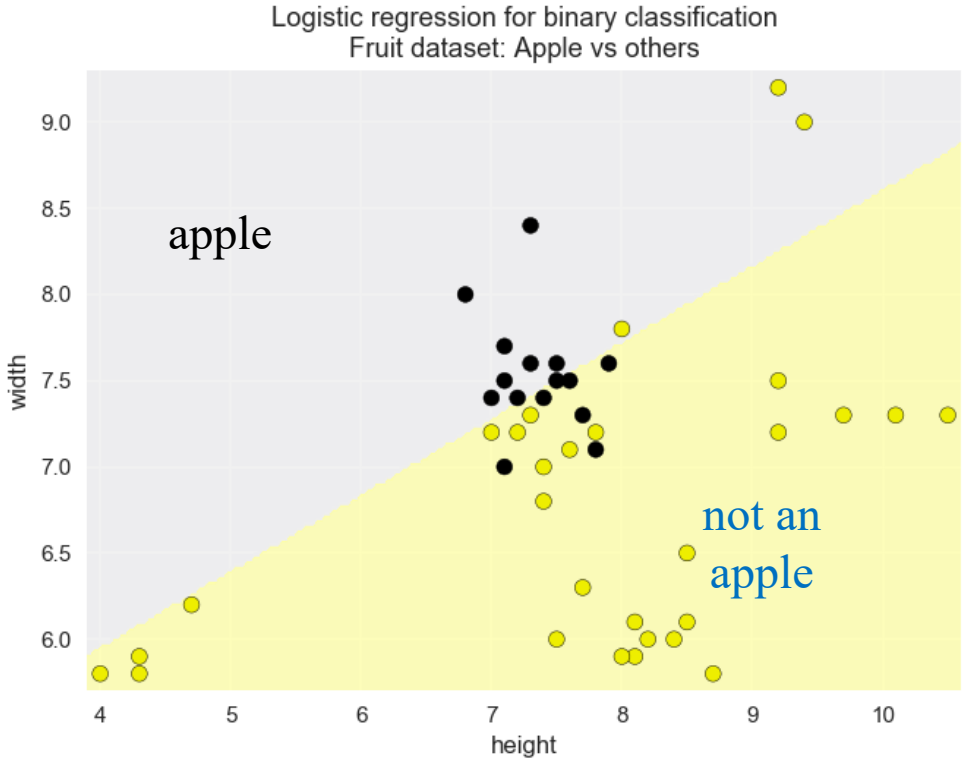
Logistic Regression for binary classification



Logistic Regression for binary classification



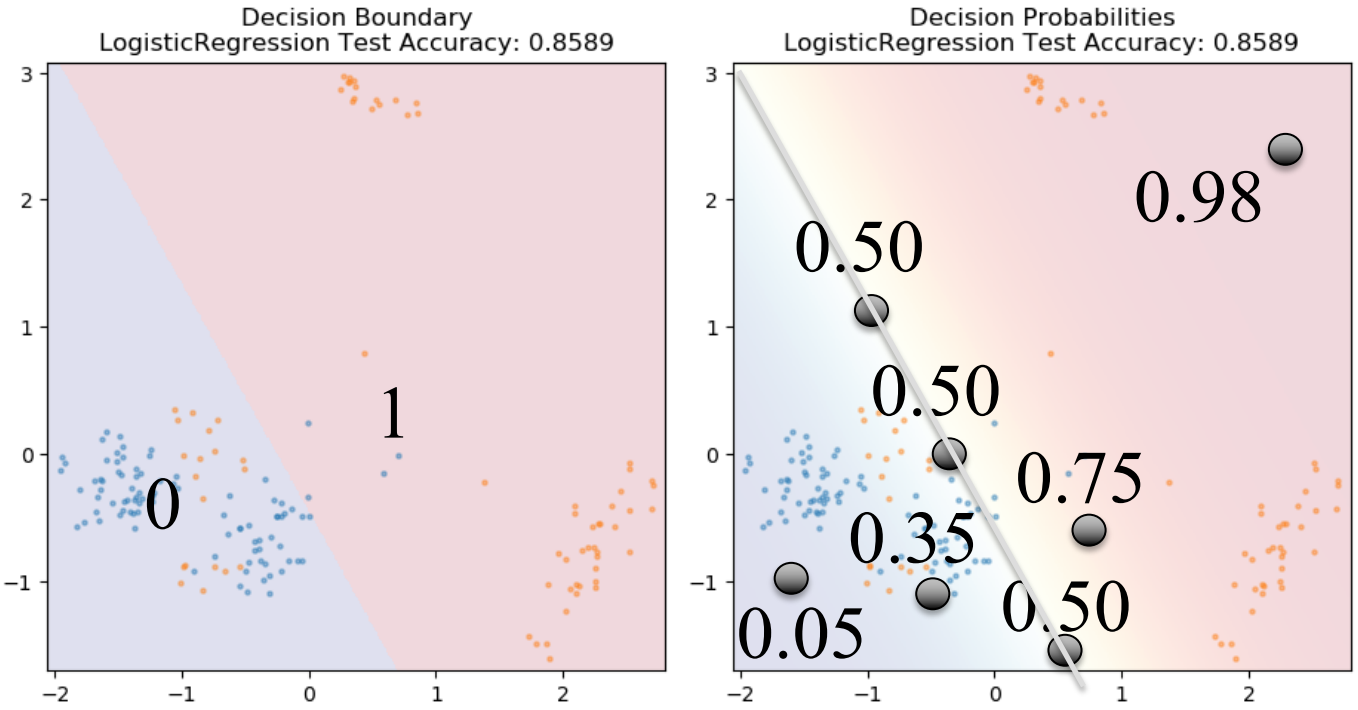
Simple logistic regression problem: two-class, two-feature version of the fruit dataset



What is output?

- .predict(x)
  - *category*
- .predict_proba(x)
  - *Probability estimates*
  - *Soft classifier*
- .predict_log_proba(x)
  - *Log probability estimates*

Hard vs Soft Classification



What is being optimized?

- **Again two views:**
- **MLE: maximum likelihood estimator**
  - $\Pr(W|X,Y)$ ***where*** $\Pr(Y|X,W) = \prod_i logistic\,(x^i \cdot w)$
- **Empirical Risk Minimization:**
  - ***Log loss:*** $-\sum_i \{\log(logistic\,(x^i \cdot w))\;\; if\;\; y_i = 1;$
    $\left(\log(1 - logistic\,(x^i \cdot w)\right) if\;\; y_i = 0\}$
- **Output = "Pr[y_i = 1]"**
  - ***Unlike k-NN; SVM***

Logistic Regression

- Intuition
- Parameters
  - *Regularization (on by default), L_1 or L_2*
  - *Regularization Parameter c (smaller is more regularized)*
  - *class_weight*
- Computation:
  - *Convex space*
- Evaluation
  - *Coming up!*

Model Evaluation for Logistic Regression

Beyond Accuracy

Evaluation

- Different applications have very different goals
- Accuracy is widely used, but many others are possible, e.g.:
  - *User satisfaction (Web search)*
  - *Amount of revenue (e-commerce)*
  - *Increase in patient survival rates (medical)*

Evaluation

- It's very important to choose evaluation methods that match the goal of your application.
- Compute your selected evaluation metric for multiple different models.
- Then select the model with 'best' value of evaluation metric.

Accuracy with imbalanced classes

- Suppose you have two classes:
  - *Relevant (R): the positive class*
  - *Not_Relevant (N): the negative class*
- Out of 1000 randomly selected items, on average
  - *One item is relevant and has an R label*
  - *The rest of the items (999 of them) are not relevant and labelled N.*
- Recall that:

$$\text{Accuracy} = \frac{\#\text{correct predictions}}{\#\text{total instances}}$$

Accuracy with imbalanced classes

- You build a classifier to predict relevant items, and see that its accuracy on a test set is 99.9%.
- Wow! Amazingly good, right?
- For comparison, suppose we had a "dummy" classifier that didn't look at the features at all, and always just blindly predicted the most frequent class (i.e. the negative N class).

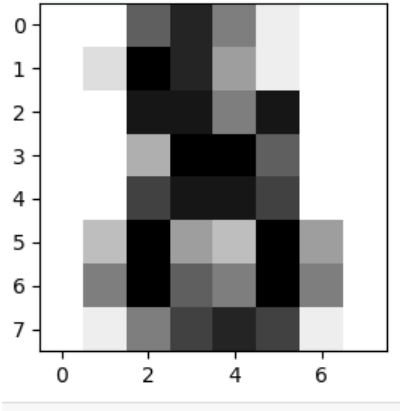
Accuracy with imbalanced classes

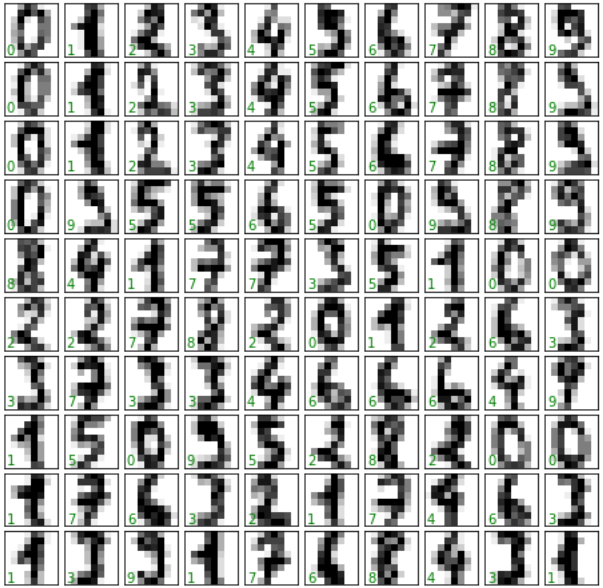
- Assuming a test set of 1000 instances, what would this dummy classifier's accuracy be?
- Answer:

$$\text{Accuracy}_{\text{DUMMY}} = 999 / 1000 = 99.9\%$$

New dataset! load_digits

- 1,797 examples
- 8x8 handwritten digits





Let's try learning a classifier on this digits dataset

- Predict 1 or not 1
- Actual sophisticated Logistic Regression classifier: 90.89%

Create a binary classification problem: is digit 1 vs NOT digit 1

- A "dummy" classifier that always guessed "is 1" would be correct

  182/1797 = 10.13% of the time

- What is the most frequent of the two classes?

Create a binary classification problem: is digit 1 vs NOT digit 1

- A "dummy" classifier that always guessed "is 1" would be correct

  182/1797 = 10.13% of the time

- What is the most frequent of the two classes? NOT 1.
- A classifier that always guessed "NOT 1" would be correct 1 – 0.1013 = 89.87% of the time

Let's try learning a classifier on this digits dataset

Actual sophisticated Logistic Regression classifier:
90.89%

Dummy classifier: 89.87%

The "dummy" isn't looking so dumb right about now
(though we never would put it into production…)

Dummy classifiers completely ignore the input data!

- Dummy classifiers serve as a sanity check on your classifier's performance.
- They provide a null metric (e.g. null accuracy) baseline.
- Dummy classifiers should not be used for real problems.

Dummy classifiers completely ignore the input data!

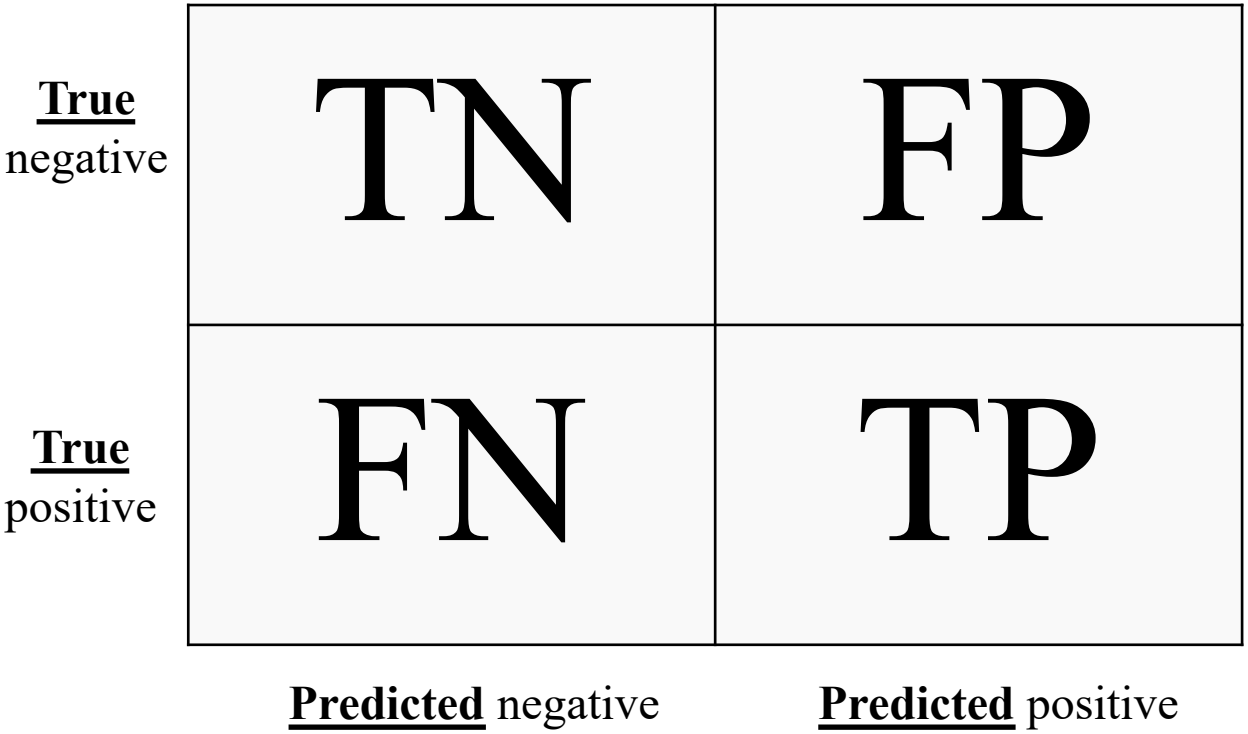
- Some commonly-used settings for the `strategy` parameter for DummyClassifier in scikit-learn:
  - *most_frequent : predicts the most frequent label in the training set.*
  - *stratified : random predictions based on training set class distribution.*
  - *uniform : generates predictions uniformly at random.*
  - *constant : always predicts a constant label provided by the user.*

Dummy regressors

`strategy` **parameter options:**

- *mean* : predicts the mean of the training targets.
- *median* : predicts the median of the training targets.
- *quantile* : predicts a user-provided quantile of the training targets.
- *constant* : predicts a constant user-provided value.

Binary prediction outcomes



Label 1 = positive class (class of interest)

Label 0 = negative class (everything else)

TP = true positive
FP = false positive (Type I error)
TN = true negative
FN = false negative (Type II error)

Confusion matrix for binary prediction task

	Predicted negative	Predicted positive
True negative	TN = 356	FP = 51
True positive	FN = 38	TP = 5

N = 450

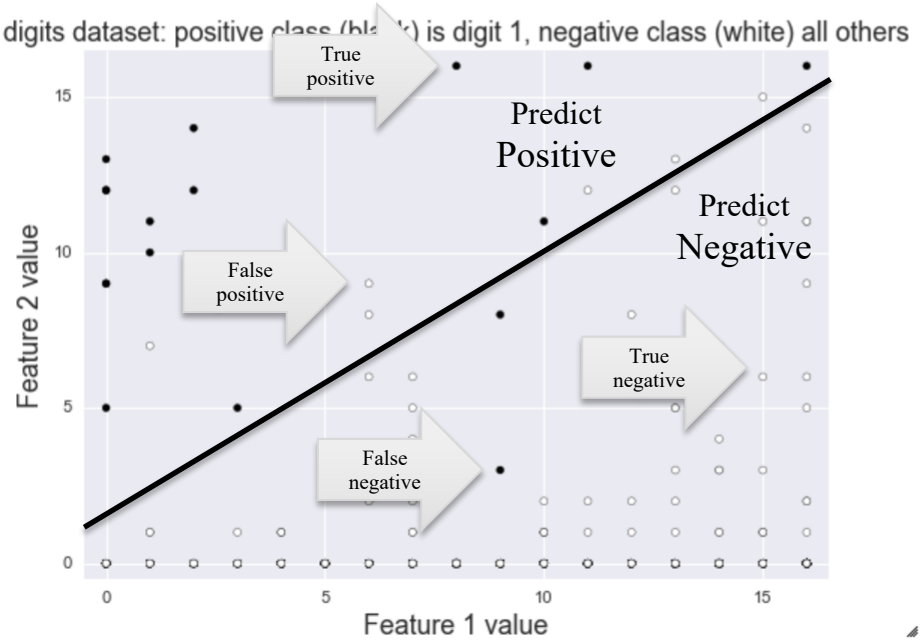
- Every test instance is in exactly one box (integer counts).
- Breaks down classifier results by error type.
- Thus, provides more information than simple accuracy.
- Helps you choose an evaluation metric that matches project goals.
- Not a single number like accuracy.. but there are many possible metrics that can be derived from the confusion matrix.

Confusion Matrix for Binary Prediction Task

	Predicted negative	Predicted positive	
True negative	TN = 400	FP = 7	
True positive	FN = 17	TP = 26	
			$N = 450$

Always look at the confusion matrix for your classifier.

Visualization of Different Error Types



TN = 429	FP = 6
FN = 2	TP = 13

Accuracy: for what fraction of all instances is the classifier's prediction correct (for either positive or negative class)?

True negative	TN = 400	FP = 7	
True positive	FN = 17	TP = 26	
	Predicted negative	Predicted positive	$N = 450$

$$\text{Accuracy} = \frac{TN+TP}{TN+TP+FN+FP}$$

$$= \frac{400+26}{400+26+17+7}$$

$$= 0.95$$

Classification error (1 – Accuracy): for what fraction of all instances is the classifier's prediction incorrect?

True negative	TN = 400	FP = 7	
True positive	FN = 17	TP = 26	
	Predicted negative	Predicted positive	$N = 450$

$$\text{ClassificationError} = \frac{FP + FN}{TN + TP + FN + FP}$$

$$= \frac{7+17}{400+26+17+7}$$

$$= 0.060$$

Recall, or True Positive Rate (TPR): what fraction of all positive instances does the classifier correctly identify as positive?

	Predicted negative	Predicted positive	
True negative	TN = 400	FP = 7	
True positive	FN = 17	TP = 26	
			$N = 450$

$$\text{Recall} = \frac{TP}{TP+FN}$$

$$= \frac{26}{26+17}$$

$$= 0.60$$

Recall is also known as:

- True Positive Rate (TPR)
- Sensitivity
- Probability of detection

Precision: what fraction of positive predictions are correct?

True negative	TN = 400	FP = 7	
True positive	FN = 17	TP = 26	
	Predicted negative	Predicted positive	$N = 450$

$$\text{Precision} = \frac{TP}{TP+FP}$$

$$= \frac{26}{26+7}$$

$$= 0.79$$

False positive rate (FPR): what fraction of all negative instances does the classifier incorrectly identify as positive?

	Predicted negative	Predicted positive	
True negative	TN = 400	FP = 7	
True positive	FN = 17	TP = 26	
			$N = 450$

$$FPR = \frac{FP}{TN+FP}$$

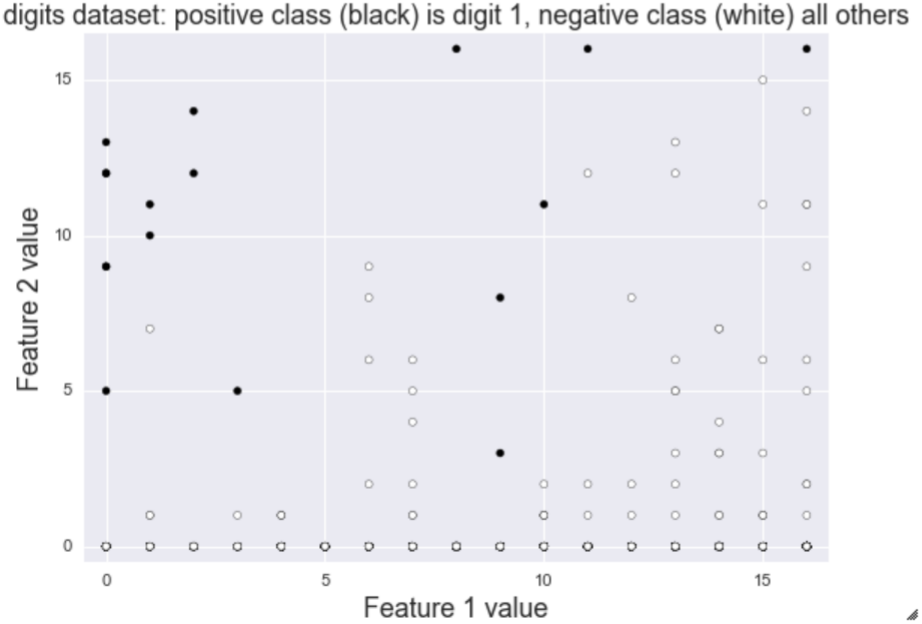
$$= \frac{7}{400+7}$$

$$= 0.02$$

False Positive Rate is also known as:

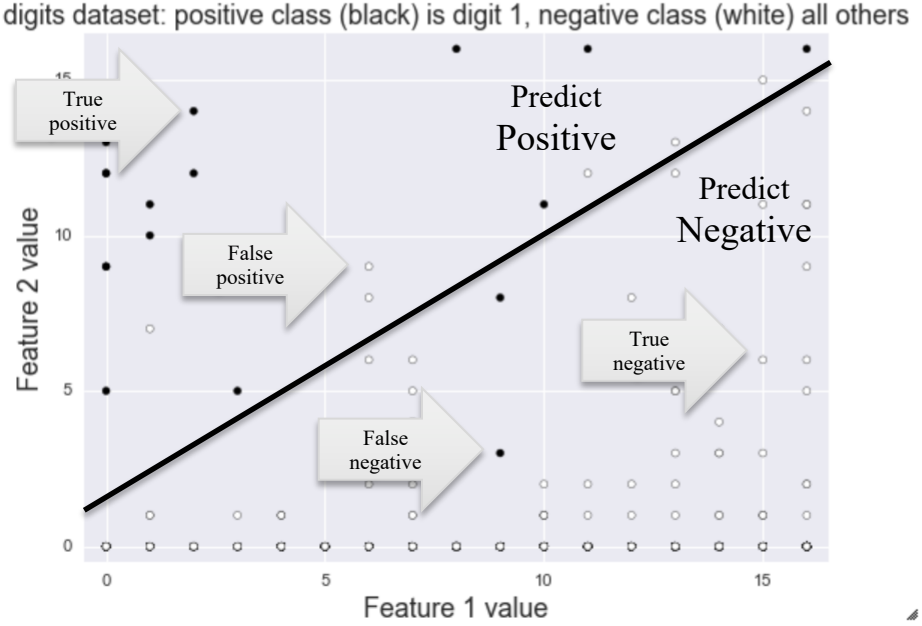
- Specificity

A Graphical Illustration of Precision & Recall



TN =	FP =
FN =	TP =

The Precision-Recall Tradeoff

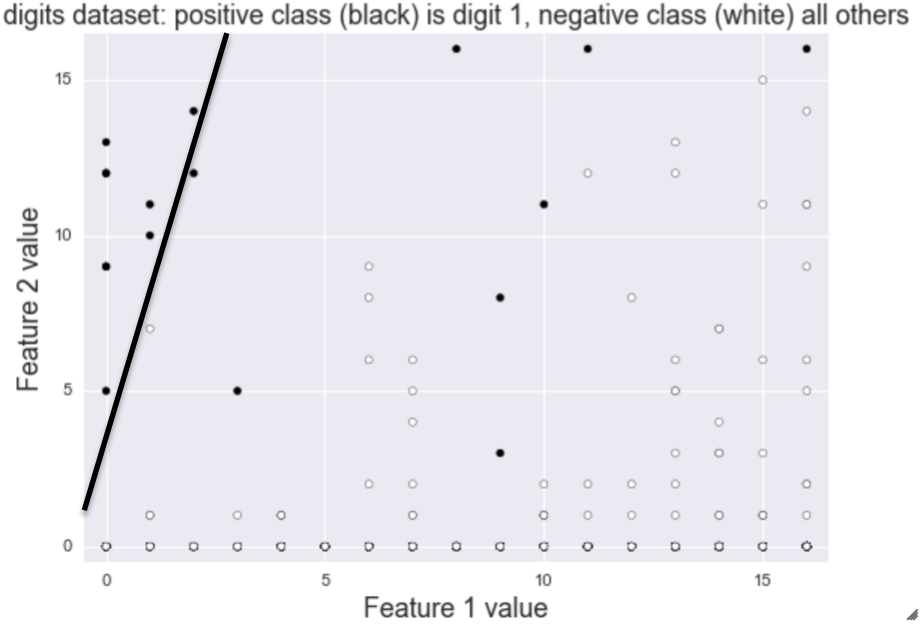


TN = 429	FP = 6
FN = 2	TP = 13

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{13}{19} = 0.68$$

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{13}{15} = 0.87$$

High Precision, Lower Recall

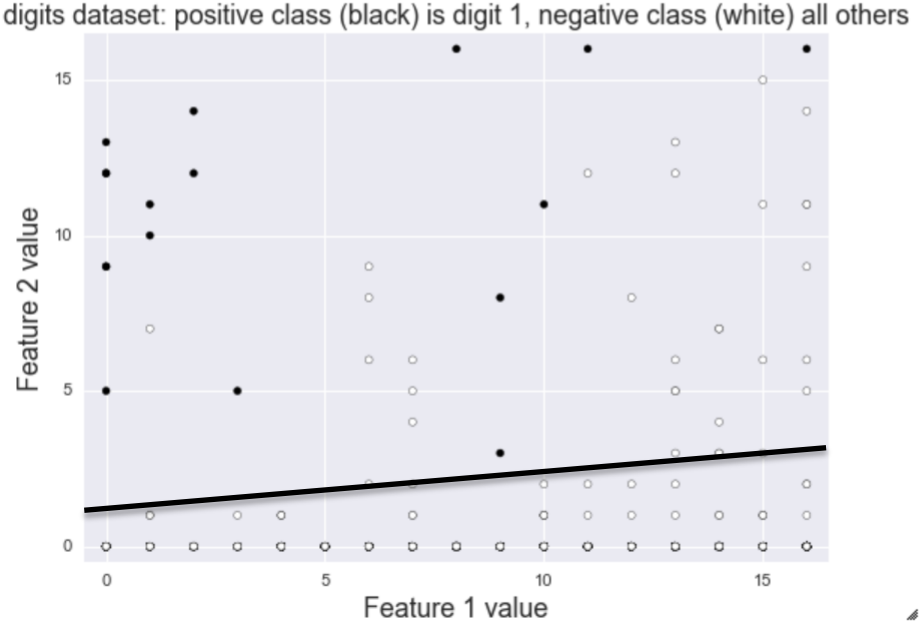


TN = 435	FP = 0
FN = 8	TP = 7

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{7}{7} = 1.00$$

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{7}{15} = 0.47$$

Low Precision, High Recall



TN = 408	FP = 27
FN = 0	TP = 15

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{15}{42} = 0.36$$

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{15}{15} = 1.00$$

There is often a tradeoff between precision and recall

- Recall-oriented machine learning tasks?

- Precision-oriented machine learning tasks?

There is often a tradeoff between precision and recall

- **Recall-oriented machine learning tasks:**
  - *Search and information extraction in legal discovery*
  - *Tumor detection*
  - *Often paired with a human expert to filter out false positives*
- **Precision-oriented machine learning tasks:**
  - *Search engine ranking, query suggestion*
  - *Document classification*
  - *Many customer-facing tasks (users remember failures!)*

F1-score: combining precision & recall into a single number

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = \frac{2 \cdot TP}{2 \cdot TP + FN + FP}$$

F-score: generalizes F1-score for combining precision & recall into a single number

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = \frac{2 \cdot TP}{2 \cdot TP + FN + FP}$$

$$F_\beta = (1 + \beta^2) \cdot \frac{Precision \cdot Recall}{(\beta^2 \cdot Precision) + Recall} = \frac{(1 + \beta^2) \cdot TP}{(1 + \beta^2) \cdot TP + \beta \cdot FN + FP}$$

β allows adjustment of the metric to control the emphasis on recall vs precision:

- **Precision-oriented users: $\beta = 0.5$ (false positives hurt performance more than false negatives)**
- **Recall-oriented users: $\beta = 2$ (false negatives hurt performance more than false positives)**

Think Pair Share

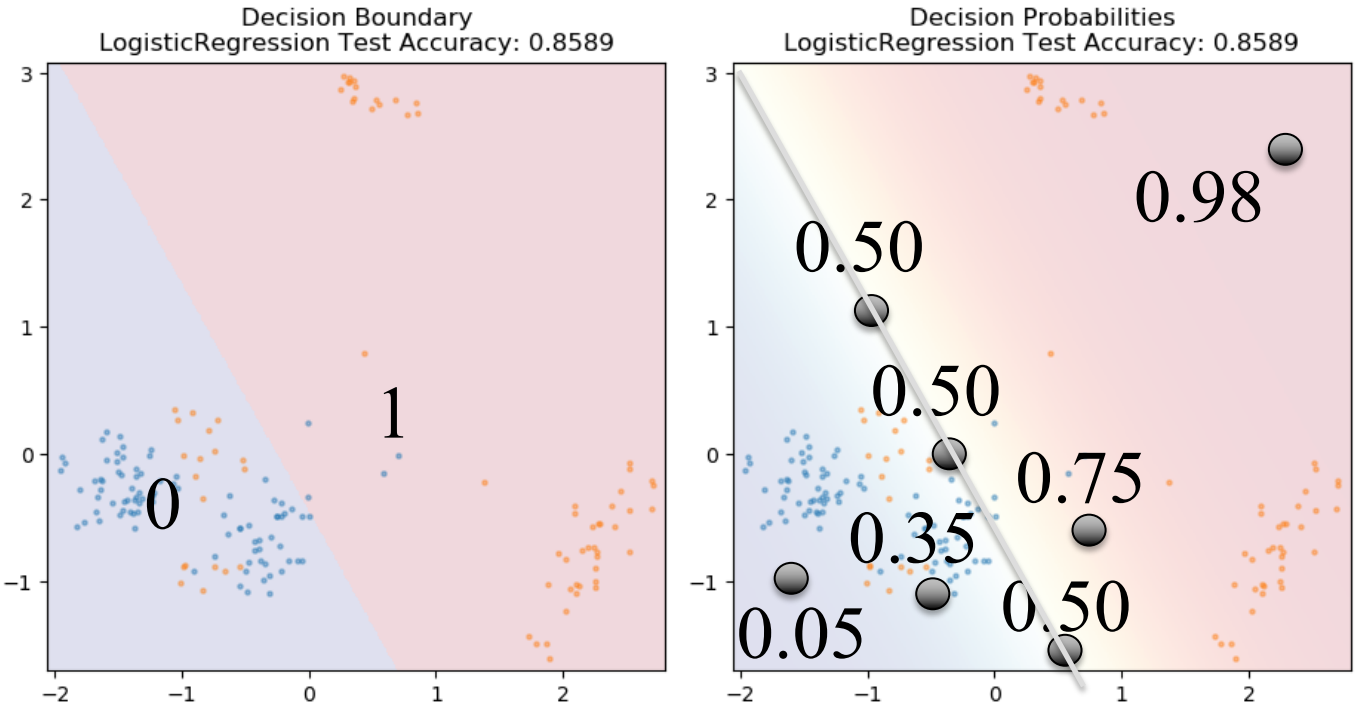
- These results have:
- A) The same precision and same recall.
- B) The same precision but different recall.
- C) Different precision but the same recall.
- D) Different precision and recall.

	Positive Label	Negative Label
Positive	250	45
Negative	35	60

	Positive Label	Negative Label
Positive	250	45
Negative	35	660

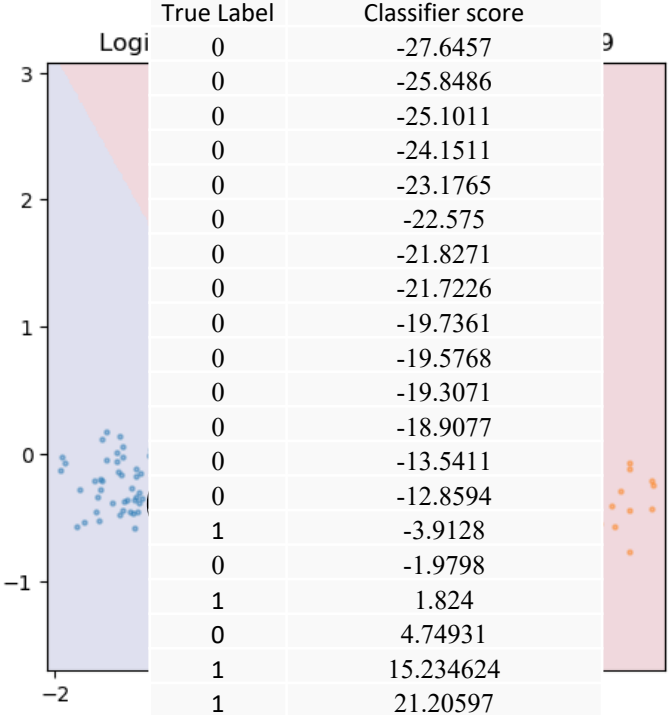
Precision-Recall and ROC curves

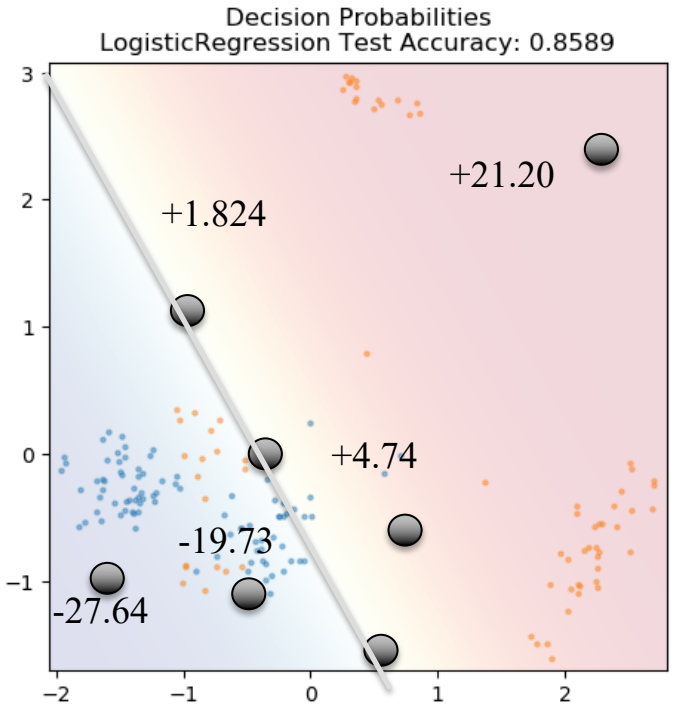
Hard vs Soft Classification



Imagine sorting all test points by their classifier score

True Label	Classifier score
0	-27.6457
0	-25.8486
0	-25.1011
0	-24.1511
0	-23.1765
0	-22.575
0	-21.8271
0	-21.7226
0	-19.7361
0	-19.5768
0	-19.3071
0	-18.9077
0	-13.5411
0	-12.8594
1	-3.9128
0	-1.9798
1	1.824
0	4.74931
1	15.234624
1	21.20597





Decision Functions (decision_function)

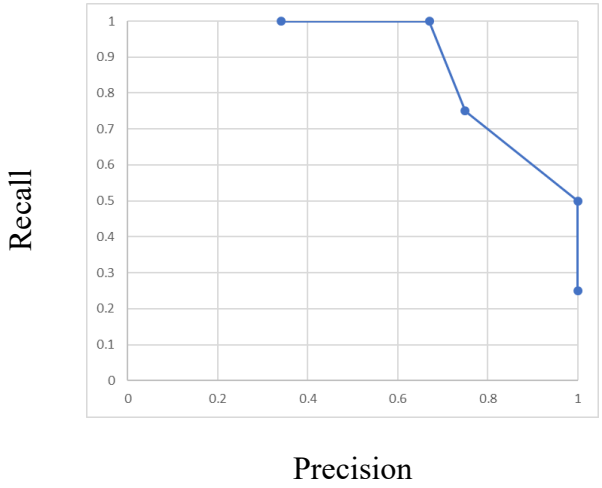
- Each classifier score value per test point indicates how confidently the classifier predicts the positive class (large-magnitude positive values) or the negative class (large-magnitude negative values).
- Choosing a fixed decision threshold gives a classification rule.
- By sweeping the decision threshold through the entire range of possible score values, we get a series of classification outcomes that form a curve.

True Label	Classifier score
0	-27.6457
0	-25.8486
0	-25.1011
0	-24.1511
0	-23.1765
0	-22.575
0	-21.8271
0	-21.7226
0	-19.7361
0	-19.5768
0	-19.3071
0	-18.9077
0	-13.5411
0	-12.8594
1	-3.9128
0	-1.9798
1	1.824
0	4.74931
1	15.234624
1	21.20597

Varying the Decision Threshold

True Label	Classifier score
0	-27.6457
0	-25.8486
0	-25.1011
0	-24.1511
0	-23.1765
0	-22.575
0	-21.8271
0	-21.7226
0	-19.7361
0	-19.5768
0	-19.3071
0	-18.9077
0	-13.5411
0	-12.8594
1	-3.9128
0	-1.9798
1	1.824
0	4.74931
1	15.234624
1	21.20597

Classifier score threshold	Precision	Recall
-20	4/12=0.34	4/4=1.00
-10	4/6=0.67	4/4=1.00
0	3/4=0.75	3/4=0.75
10	2/2=1.0	2/4=0.50
20	1/1=1.0	1/4 = 0.25

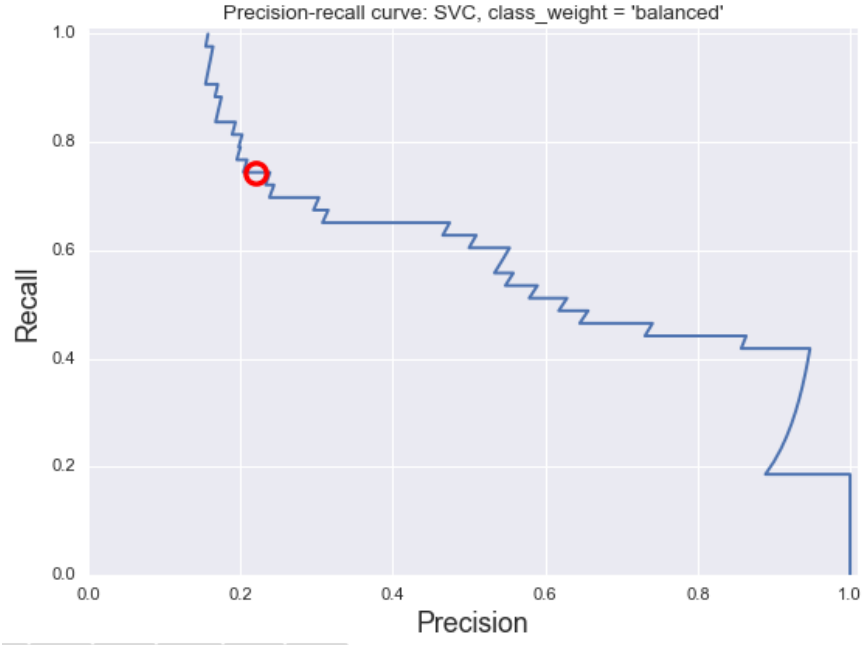


Precision-Recall Curves

X-axis: Precision
Y-axis: Recall

Top right corner:

- The "ideal" point
- Precision = 1.0
- Recall = 1.0

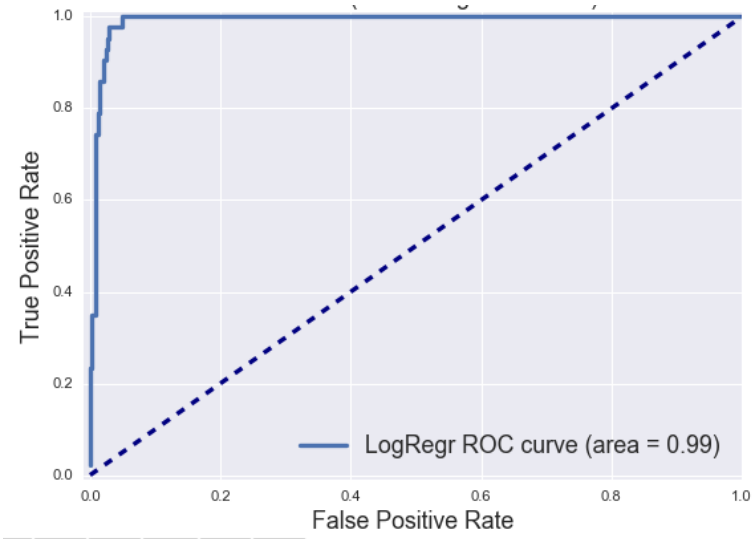


ROC Curves

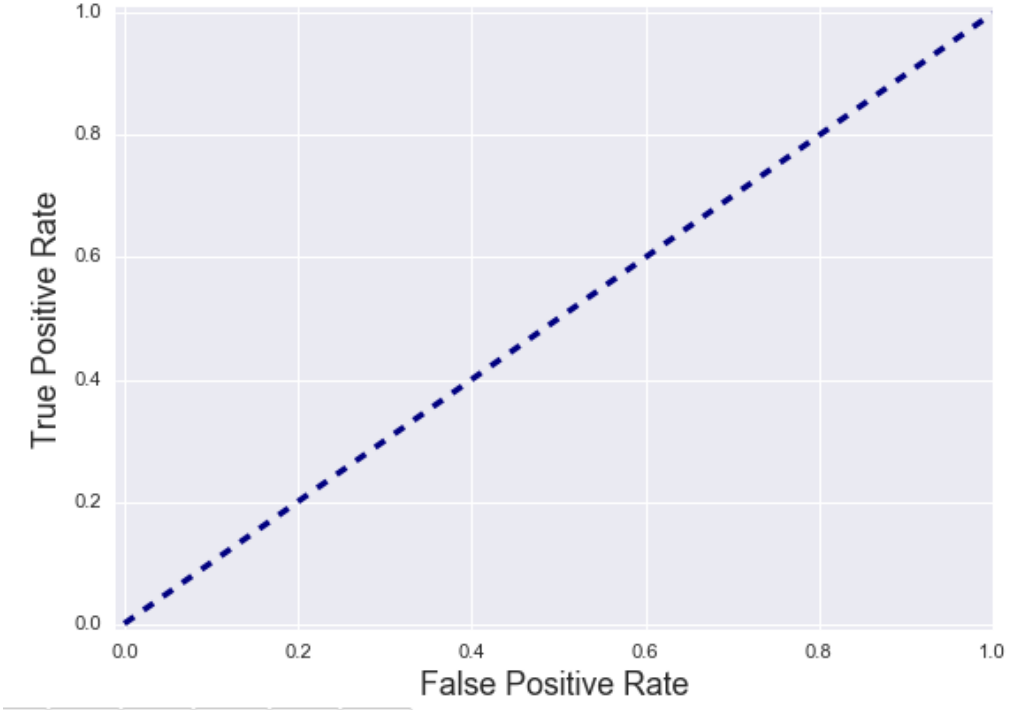
X-axis: False Positive Rate
Y-axis: True Positive Rate

Top left corner:

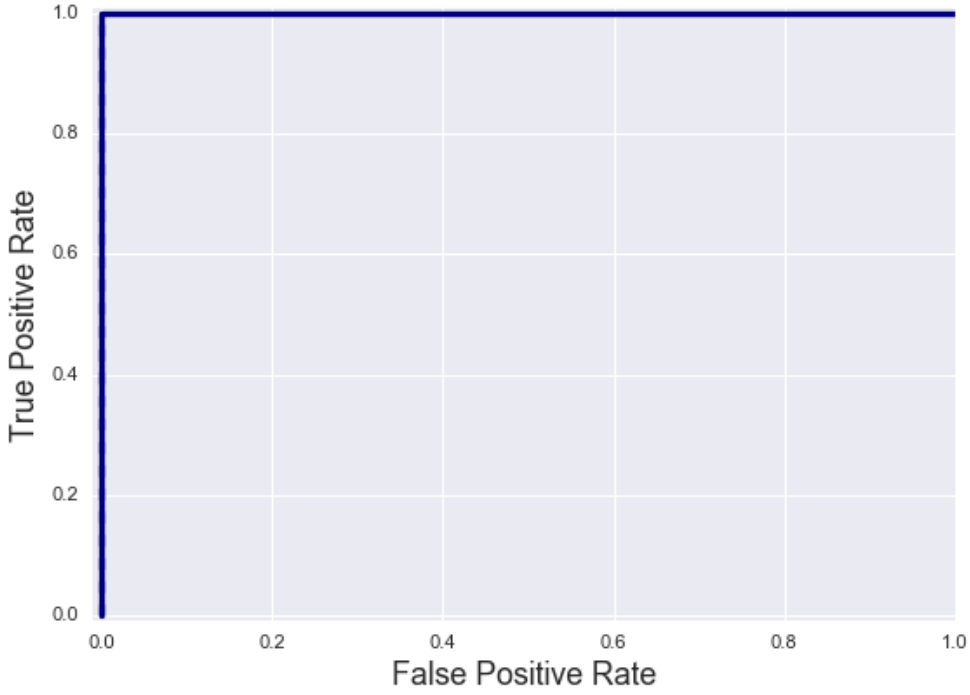
- The "ideal" point
- False positive rate of zero
- True positive rate of one



ROC curve examples: random guessing

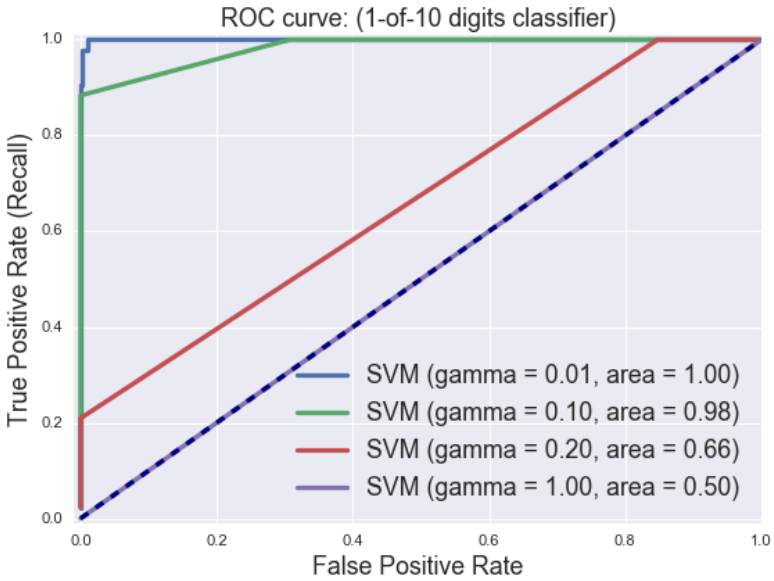


ROC curve examples: perfect classifier



Summarizing an ROC curve in one number: Area Under the Curve (AUC)

- AUC = 0 (worst) AUC = 1 (best)
- AUC can be interpreted as:
  1. *The total area under the ROC curve.*
  2. *The probability that the classifier will assign a higher score to a randomly chosen positive example than to a randomly chosen negative example.*
- Advantages:
  - *Gives a single number for easy comparison.*
  - *Does not require specifying a decision threshold.*
- Drawbacks:
  - *As with other single-number metrics, AUC loses information, e.g. about tradeoffs and the shape of the ROC curve.*
  - *This may be a factor to consider when e.g. wanting to compare the performance of classifiers with overlapping ROC curves.*



What evaluation measures are available?

```
from sklearn.metrics.scorer import SCORERS

print(sorted(list(SCORERS.keys())))
from sklearn.metrics.scorer import SCORERS

print(sorted(list(SCORERS.keys())))
['accuracy', 'adjusted_mutual_info_score', 'adjusted_rand_score', 'average_precision',
'completeness_score', 'explained_variance', 'f1', 'f1_macro', 'f1_micro', 'f1_samples',
'f1_weighted', 'fowlkes_mallows_score', 'homogeneity_score', 'log_loss',
'mean_absolute_error', 'mean_squared_error', 'median_absolute_error',
'mutual_info_score', 'neg_log_loss', 'neg_mean_absolute_error',
'neg_mean_squared_error', 'neg_mean_squared_log_error', 'neg_median_absolute_error',
'normalized_mutual_info_score', 'precision', 'precision_macro', 'precision_micro',
'precision_samples', 'precision_weighted', 'r2', 'recall', 'recall_macro',
'recall_micro', 'recall_samples', 'recall_weighted', 'roc_auc', 'v_measure_score']
```

Concluding Notes

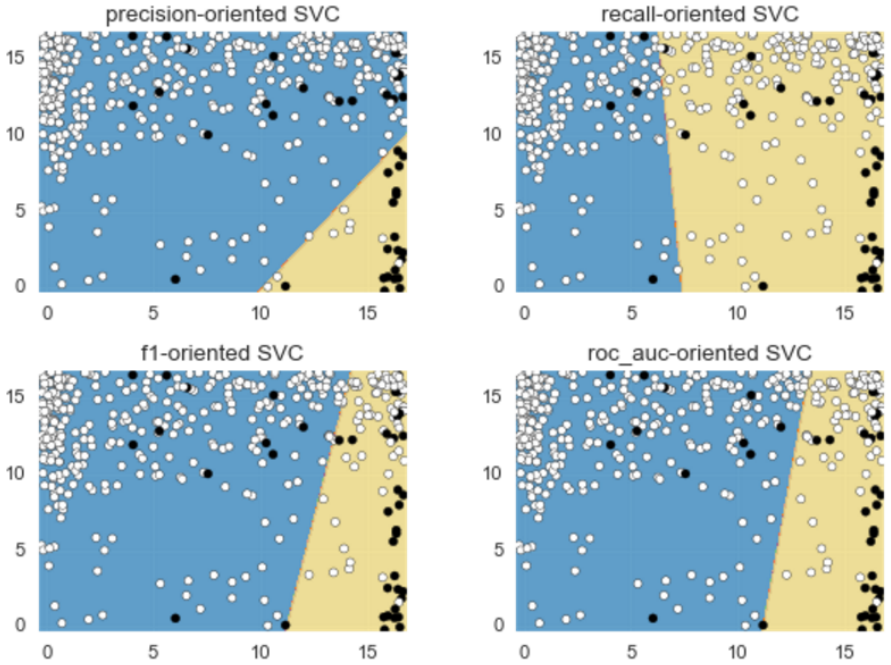
- **Accuracy is often not the right evaluation metric for many real-world machine learning tasks**
  - *False positives and false negatives may need to be treated very differently*
  - *Make sure you understand the needs of your application and choose an evaluation metric that matches your application, user, or business goals.*
- **Examples of additional evaluation methods include:**
  - *Learning curve: How much does accuracy (or other metric) change as a function of the amount of training data?*
  - *Sensitivity analysis: How much does accuracy (or other metric) change as a function of key learning parameter values?*

Optimizing Classifiers for Different Evaluation Metrics

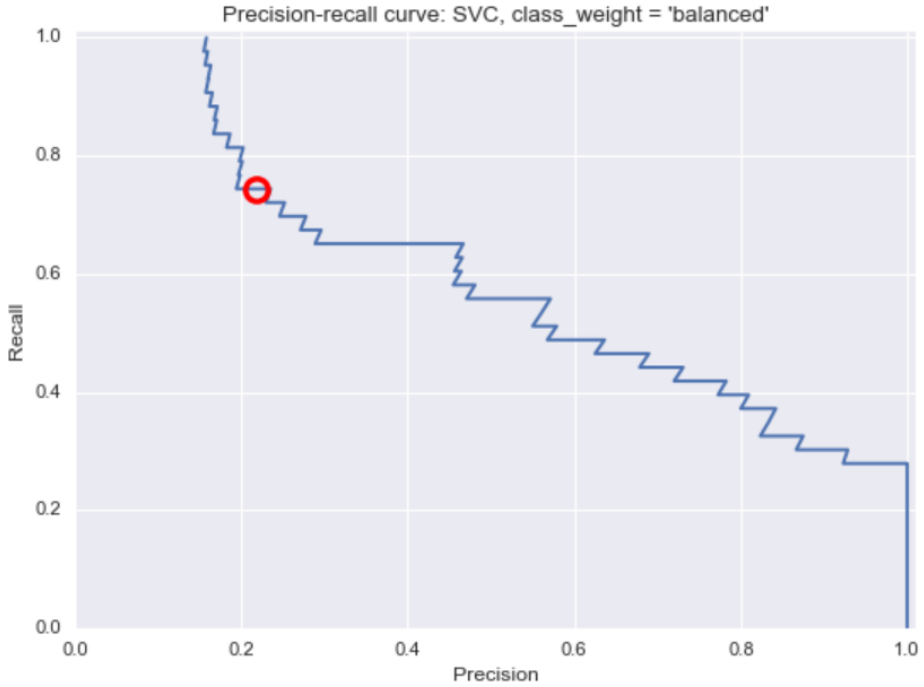
Certain functions (parameter search) can take an evaluation metric as a parameter

```
clf = Logistic(C=100.0).fit(X_twovar_train, y_train)
grid_values = {'class_weight':['balanced',
{1:2},{1:3},{1:4},{1:5},{1:10},{1:20},{1:50}]}
plt.figure(figsize=(10,7))
for i, eval_metric in enumerate(('precision','recall', 'f1','roc_auc')):
    grid_clf_custom = GridSearchCV(clf, param_grid=grid_values,
                                   scoring=eval_metric)
    grid_clf_custom.fit(X_twovar_train, y_train)
        ... etc
```

Example: Optimizing a Classifier Using Different Evaluation Metrics



Example: Precision-Recall Curve of Default Support Vector Classifier



Training, Validation, and Test Framework for Model Selection and Evaluation

- Using only cross-validation or a test set to do model selection may lead to more subtle overfitting / optimistic generalization estimates
  - *Why?*
- Instead, use three data splits:
  1. *Training set (model building)*
  2. *Validation set (model selection—tune parameters, but never fit models)*
  3. *Test set (final evaluation)*
- In practice:
  - *Create an initial training/test split*
  - *Do cross-validation on the training data for model/parameter selection*
  - *Save the held-out test set for final model evaluation*