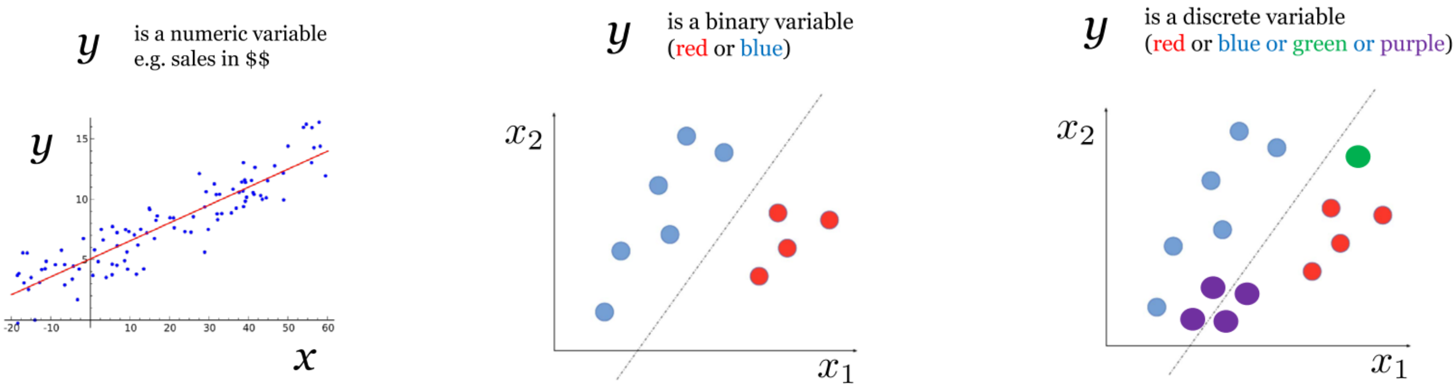
Classification

Machine Learning and Computational Statistics (DSC6135)

1. Introduction to Classification

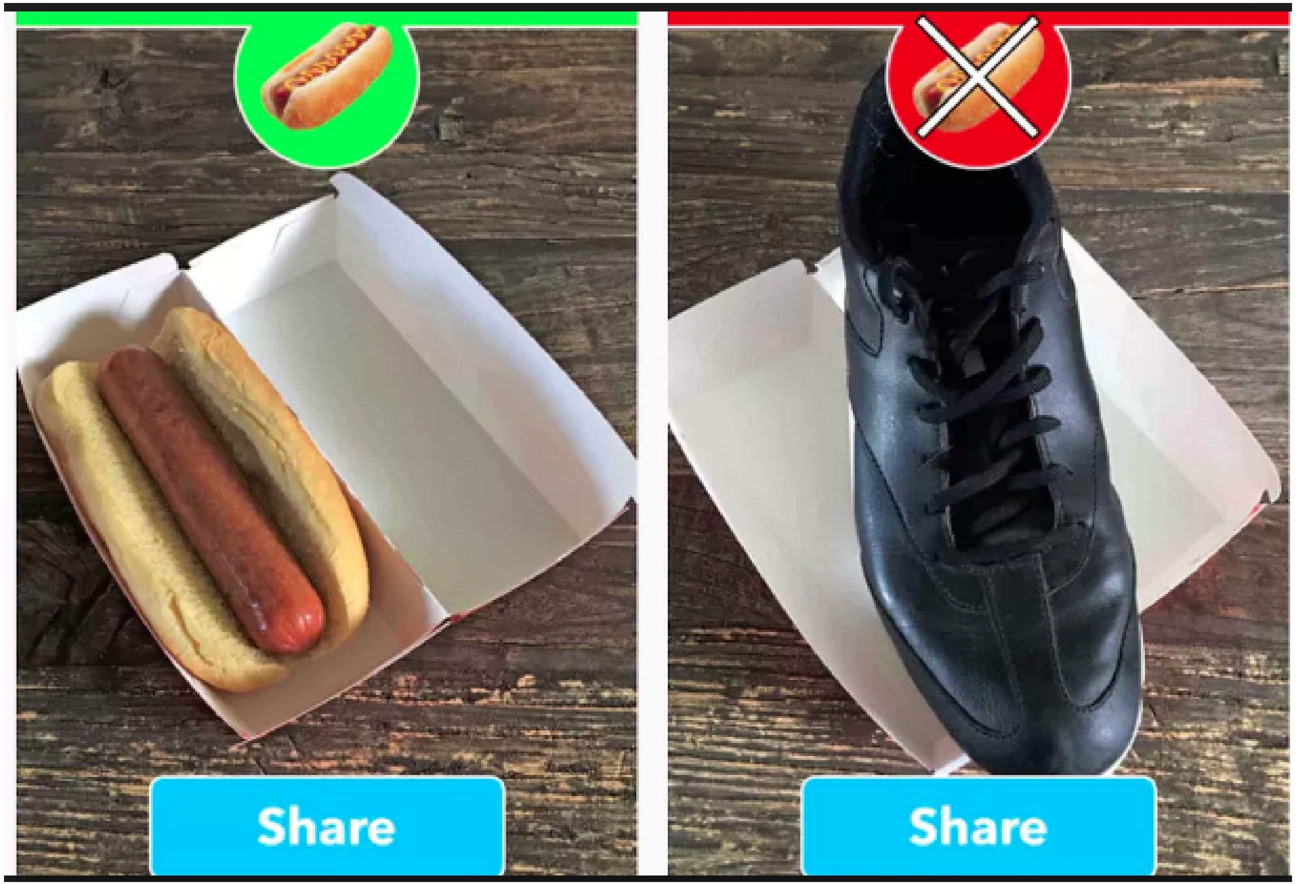


From left to right:

- linear regression
- binary classification
- multi-class classification

Example: Binary Classification

Example: Binary Classification



Example: Multi-class Classification

Predict words from keyboard trajectories

Many possible letters: *Multi-class* classification

Classification overview

- Steps for classification
    1. What is prediction? --> hard binary vs. probabilities
    2. What is training? --> we need a model
    3. How to evaluate --> we need performance metrics

- Possible methods for classification
    1. Logistic Regression
    2. K-Nearest Neighbors
    3. Decission Tree Regression
    4. Boosting
    5. Suppor Vector Classifiers (SVCs), Support Vector Machines (SVMs)

Binary Prediction Step

Goal: Predict label (0 or 1) given features x

- Input: $x_i \triangleq [x_{i1}, x_{i2}, \dots x_{if} \dots x_{iF}]$

"features"
"covariates"
"predictors"
"attributes"

Entries can be real-valued, or other numeric types (e.g. integer, binary)

- Output: $y_i \in \{0, 1\}$

Binary label (0 or 1)

"responses"
"labels"

```
>>> yhat_N = model.predict(x_NF)
>>> yhat_N[:5] # peek at predictions
[0, 0, 1, 0, 1]
```

Probability Prediction Step

Goal: Predict probability p(Y=1) given features x

- Input: $x_i \triangleq [x_{i1}, x_{i2}, \ldots x_{if} \ldots x_{iF}]$

  "features"
  "covariates"
  "predictors"
  "attributes"

  Entries can be real-valued, or other numeric types (e.g. integer, binary)

- Output: $\hat{p}_i$

  "probabilities"

  Probability between 0 and 1
  e.g. 0.001, 0.513, 0.987

Probability Prediction Step

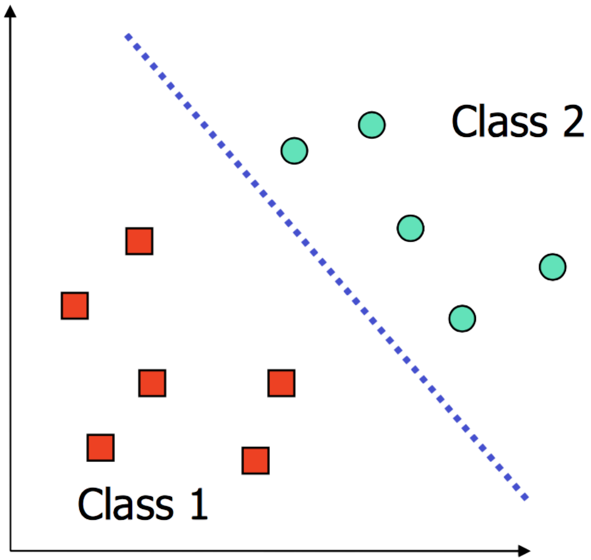
```
>>> # Given: pretrained regression object model
>>> # Given: 2D array of features x

>>> x_NF.shape
(N, F)

>>> yproba_N2 = model.predict_proba(x_NF)
>>> yproba_N2.shape
(N, 2)

>>> yproba_N2[:, 1]
[0.003, 0.358, 0.987, 0.111, 0.656]
```

Column index 1 gives probability of positive label $p(Y = 1)$

Logistic Regression: Linear Decision Boundary

We can try to model the **probability** of a data point being from a particular class by

1. which side of the decision boundary it's on
2. how far it is away from the boundary. Intuitively, the farther a data point is from the decision boundary, the more 'certain' we should be of it's classification.



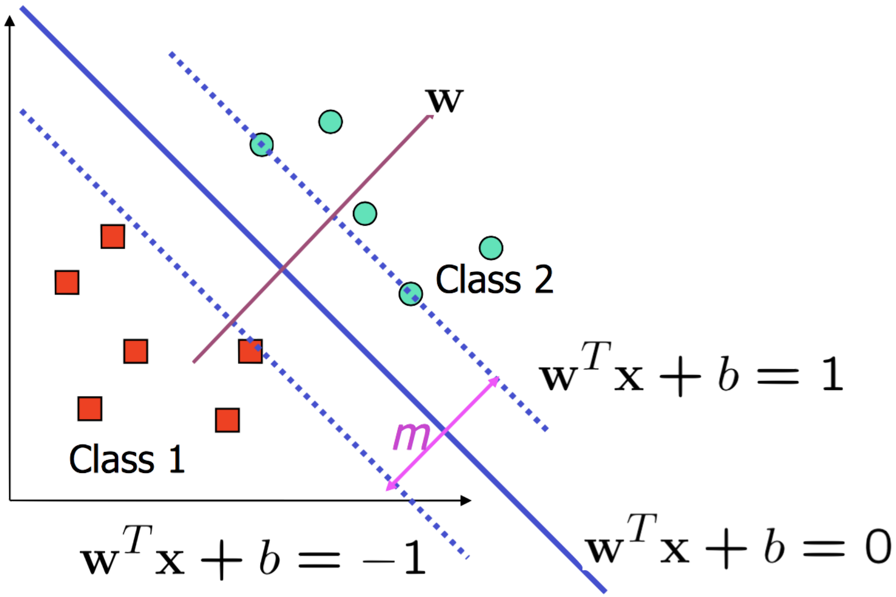
Logistic Regression: Linear Decision Boundary

When the decision boundary is linear, it is defined by the equation

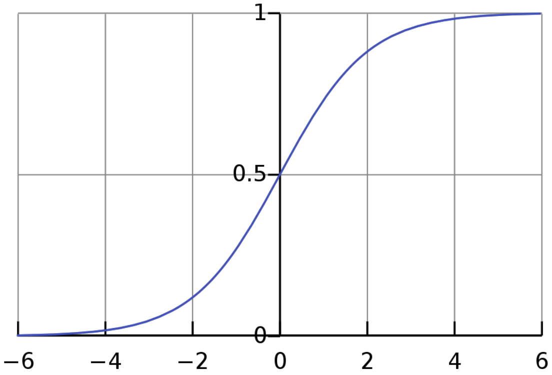
$$\mathbf{w}^\top \mathbf{x} = w_0 x_0 + w_1 x_1 + \ldots + w_D x_D = 0$$

where $x_0 = 1$.

The vector $\mathbf{w}$ allow us to gauge the 'distance' of a point from the decision boundary



To model the probability of labeling a point a certain class, we have to convert distance, $\mathbf{w}^\top \mathbf{x}$ (which is unbounded) into a number between 0 and 1, using the ***sigmoid function***:



$$\sigma(z) = \frac{1}{1 + e^{-z}},$$

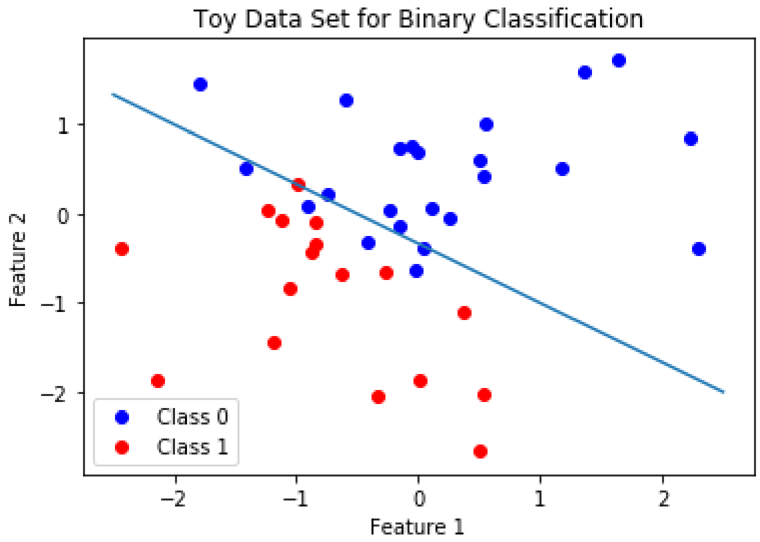
$$\text{Prob}[y = 1|\mathbf{x}] = \sigma(\mathbf{w}^\top \mathbf{x})$$

If $\text{Prob}[y = 1|\mathbf{x}] \geq 0.5$ we label $\mathbf{x}$ class 1, otherwise we label it class 0.

To *fit* our model, we need to learn the parameters of $\mathbf{w}$ that maximizes the likelihood of our training data.

How to generate data from logistic model?

In [4]:
```
x1 = np.random.randn(40, 1)  # some continuous features
x2 = np.random.randn(40, 1)
X = np.hstack((x1, x2))
z = 1 + 2*x1 + 3*x2  # linear combination with a bias
pr = 1/(1+np.exp(-z))  # pass through an inv-logit function
y = np.random.binomial(1, pr).reshape(-1) # bernoulli response variable
xspan = np.linspace(-2.5, 2.5, 100)
boundary = -(1+2*xspan) / 3.0

plot_logistic(X,y)
```



Logistic Loss

- We are given training data $X = (x_1, \ldots, x_N)$, $y = (y_1, \ldots, y_N)$, where $x_i \in \mathbb{R}^p$ and $y_i \in \{1, 0\}$.
- We assign a probability based on the logistic function to each data point for belonging to a specific class:

$$p = \text{Prob}[y = 1|x] = \frac{1}{1 + e^{-\mathbf{w}^T x}},$$

and $\text{Prob}[y = 0|x] = 1 - \text{Prob}[y = 1|x]$.

- We employ a Bernoulli random variable with probability mass function:

$$J(X, y, \mathbf{w}) = \prod_{i=1}^{N} \text{Prob}[y = y_i | x] = \prod_{i=1}^{N} p^{y_i}(1 - p)^{1-y_i}$$

- J is the maximum likelihood estimator we want to minimize, given the model.
- For optimization purposes, we maximize the log-likelihood:

$$\max_{\mathbf{w}} \log \left[\prod_{i=1}^{N} p^{y_i}(1 - p)^{1-y_i} \right]$$

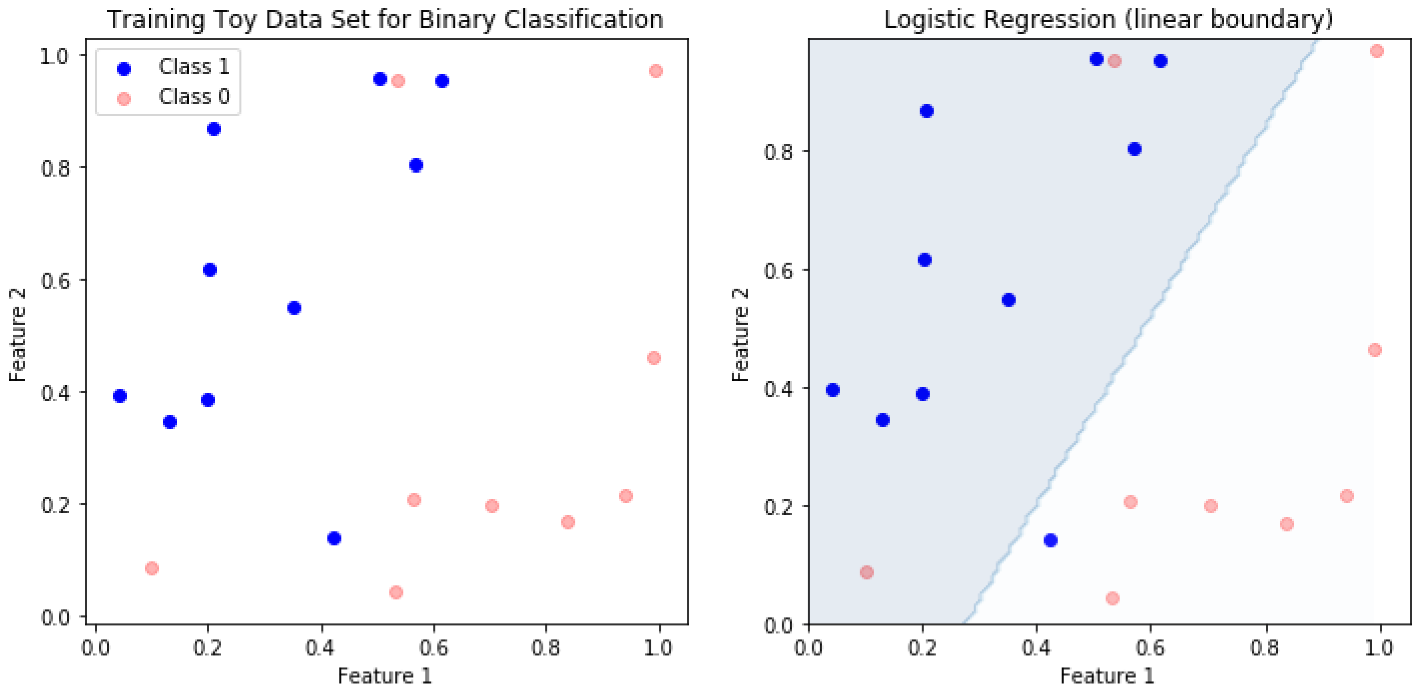
$$\max_{\mathbf{w}} \sum_{i=1}^{N} \left[y_i \log(1 + e^{-(\mathbf{w}^T x_i)}) + (1 - y_i) \log(1 + e^{(\mathbf{w}^T x_i)}) \right]$$

Example with sklearn:

In [6]:
```
# create a logistic regression model with linear boundary
logreg = linear_model.LogisticRegression(C=1., solver='lbfgs')
# fit our logistic regression model
logreg.fit(X_train, y_train)
```

Out[6]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='lbfgs',
          tol=0.0001, verbose=0, warm_start=False)
```

In [7]:
```
# plot data and decision boundary of logistic regression
plot_fig(X_train,y_train)

# plot decision boundary
ax[1] = plot_decision_boundary(X_train, y_train, logreg, 'Logistic Regression (l
inear boundary)', ax[1])

plt.tight_layout()
plt.show()
```



In [8]:

```
# evaluate model
scores_df = pd.DataFrame(data={'logistic regression': [logreg.score(X_train, y_t
rain), logreg.score(X_test, y_test)]},
                         index=['train score', 'test score'])
scores_df.head()
```

Out[8]:

	logistic regression
train score	0.842105
test score	0.759494

Questions: Why does the model do worse on testing data rather than training data?

Optimization of the logistic loss

- Summary of variables:
  - Feature vector:
$$\mathbf{x} = [1\ x_1\ x_2\ \ \dots\ \ \mathbf{x}_F]^T$$
  - Weight vector:
$$\mathbf{w} = [b\ w_1\ w_2\ \ \dots\ \ \mathbf{w}_F]^T$$
  - Score:
$$z_i = \mathbf{w}^T\mathbf{x}$$
  - Loss (minimization):
$$\sum_{i=1}^{N} -[y_i \log(1 + \exp^{-(\mathbf{w}^T x_i)}) + (1 - y_i) \log(1 + \exp^{(\mathbf{w}^T x_i)}]$$

- Gradient of the sigmoid:
$$\frac{\partial \sigma(z)}{\partial z} = \frac{\partial}{\partial z}(1+e^{-z})^{-1} = e^{-z}(1+e^{-z})^{-2} = \frac{1}{1+e^{-z}}\frac{e^{-z}}{1+e^{-z}} = \sigma(z)$$

- Gradient of the log sigmoids:

$$\frac{\partial \log \sigma(z_i)}{\partial \omega^T} = \frac{1}{\sigma(z_i)}\frac{\partial \sigma(z_i)}{\partial \omega^T} = \frac{1}{\sigma(z_i)}\frac{\partial \sigma(z_i)}{\partial z_i}\frac{\partial z_i}{\partial \omega^T} = (1-\sigma(z_i))x_i$$
$$\frac{\partial \log(1-\sigma(z_i))}{\partial \omega^T} = \frac{1}{1-\sigma(z_i)}\frac{\partial(1-\sigma(z_i))}{\partial \omega^T} = -\sigma(z_i)x_i$$

- Gradient of the logistic loss

$$\frac{\partial l_i(\omega)}{\partial \omega^T} = -y_i x_i(1-\sigma(z_i)) + (1-y_i)x_i\sigma(z_i) = x_i(\sigma(z_i)-y_i)$$

Gradient descent

- The gradient of the logistic regression does not have closed form solution!
- We will need to use iterative methods to solve the problem approximately.
- One such simple method, is gradient descent:

$$\min_{\boldsymbol{w},w_0} \; -\sum_i \log\, p(y_i \,|\boldsymbol{x}_i; \boldsymbol{w}, w_0) + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

Start with $\boldsymbol{w}^0 = 0, w_0^0 = 0$, step size s

for $t = 0, \ldots, (T-1)$

$\quad \boldsymbol{w}^{t+1} = \boldsymbol{w}^t - s\,\nabla J(\boldsymbol{w}^t, w_0^t) - \lambda \boldsymbol{w}^t$

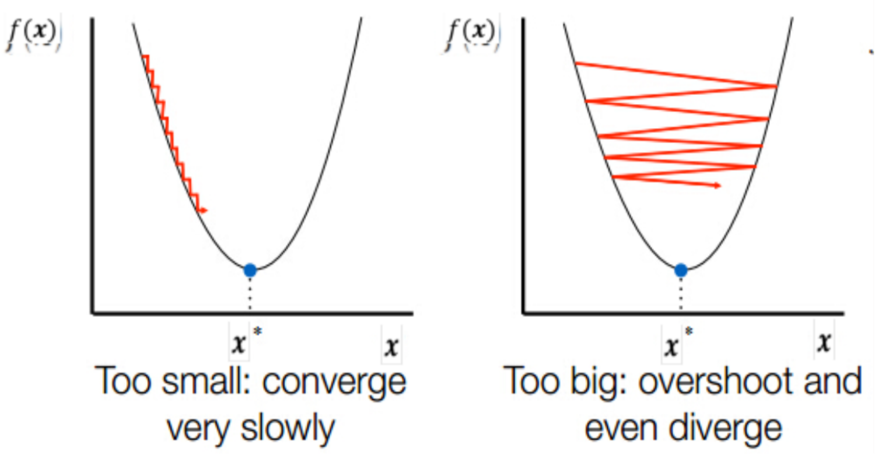
$\quad w_0^{t+1} = w_0^t - s\,\nabla J(\boldsymbol{w}^t, w_0^t)$

$\quad$ if $L(\boldsymbol{w}^{t+1}, w_0^{t+1}) - L(\boldsymbol{w}^t, w_0^t) < \delta$

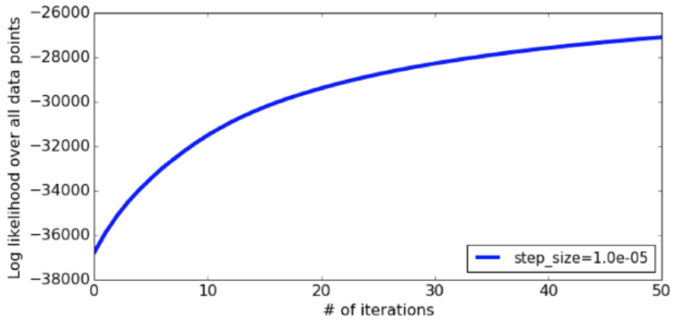
$\quad\quad$ break

return $\boldsymbol{w}^T, w_0^T$

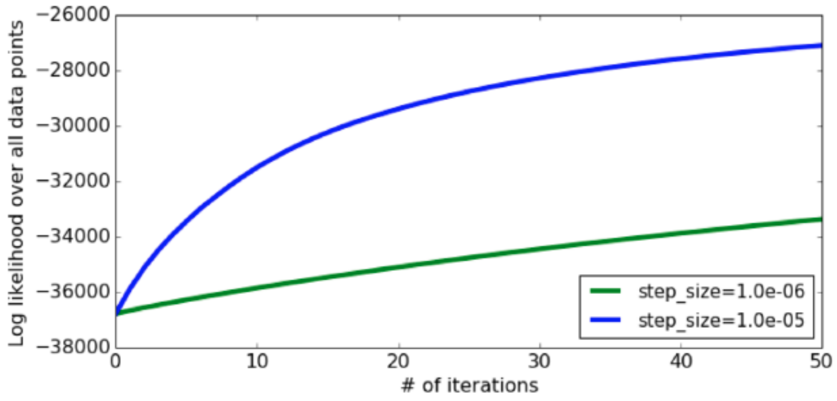
Intuition in 1D



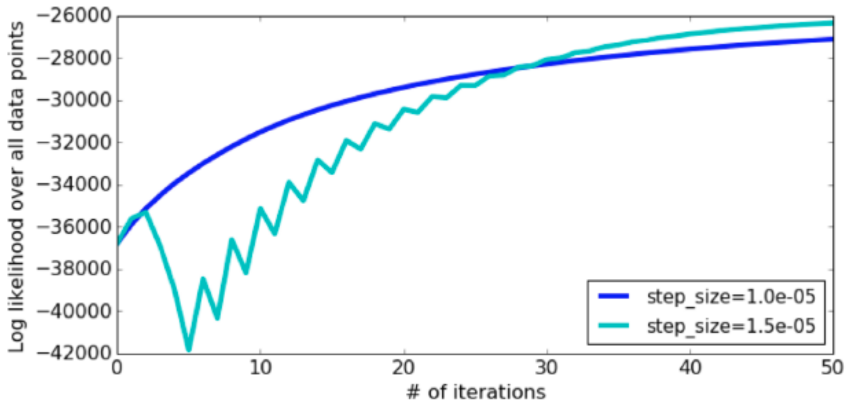
Step size tuning



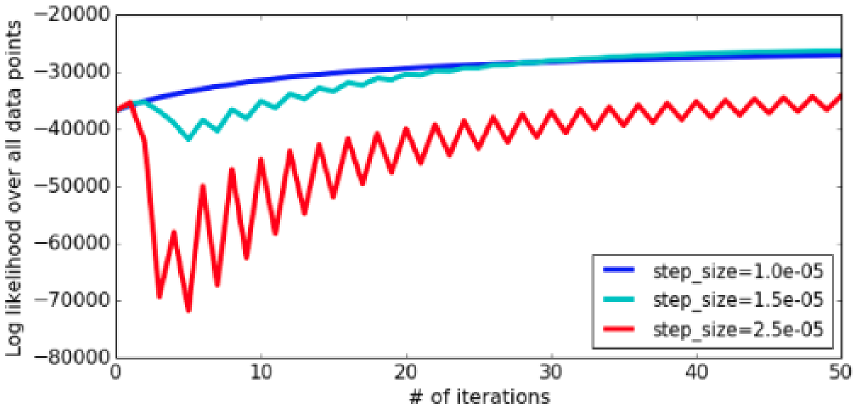
Step size tuning: too small



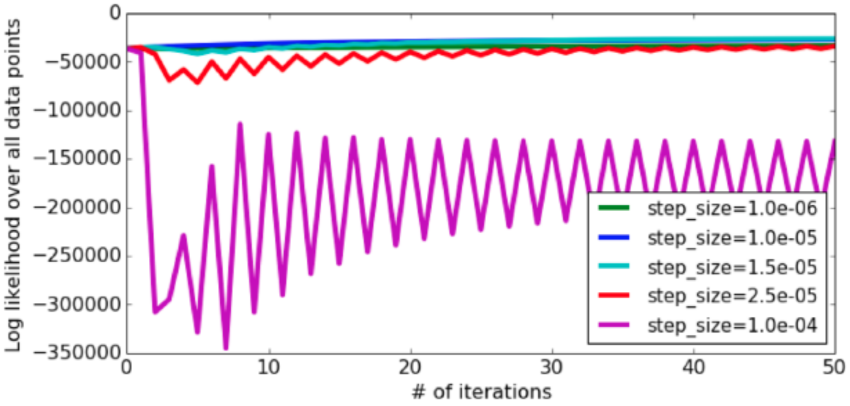
Step size tuning: large



Step size tuning: too large



Step size tuning: way tooo large

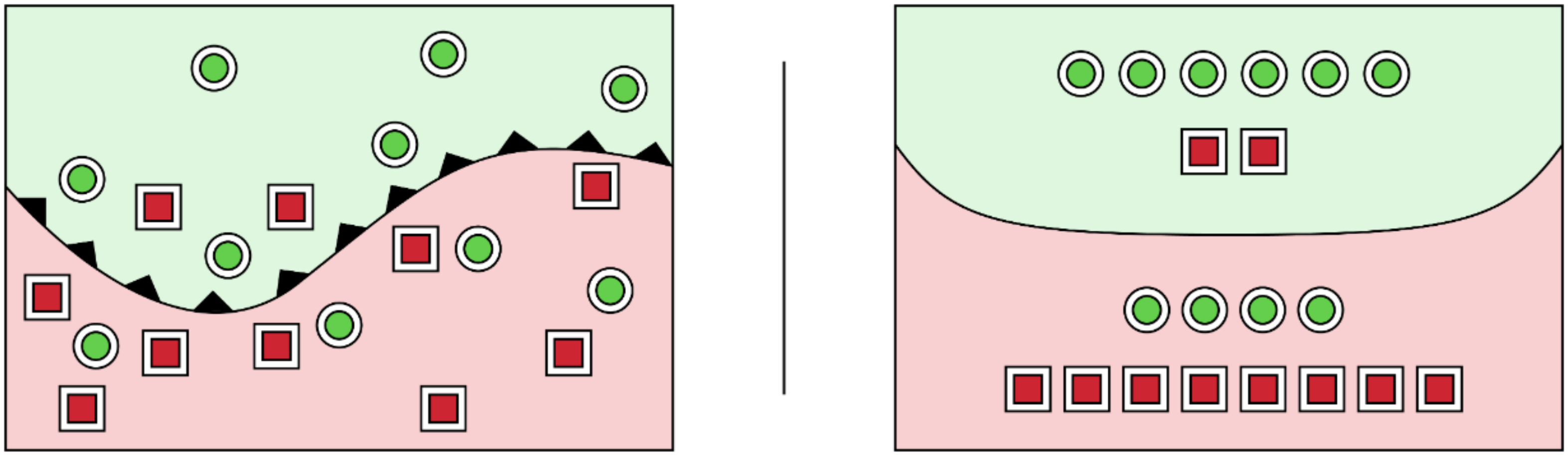


Logistic Regression: Non-Linear Decision Boundary

Go to external notebook.

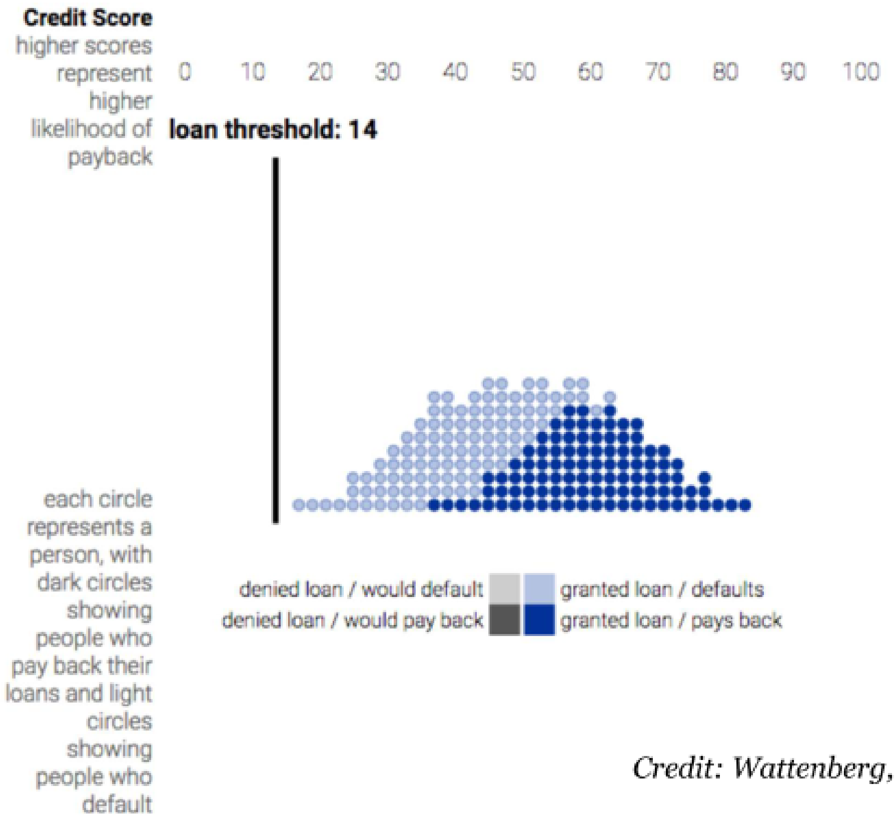
Question: What is a good/bad classifier?

2. Evaluation in Classification Task



What is a good/bad classifier? How can we measure this?

Thresholding to get Binary Decisions



Credit: Wattenberg, Viégas, Hardt

Evaluation metrics for classification

```
Many possibilities in practice.
```

1. Evaluate probabilities / scores directly: logistic loss, hinge loss, ...
2. Evaluate binary decisions at specific thresholds: accuracy, TPR, TNR, PPV, NPV, etc.
3. Evaluate across range of thresholds ROC curve, Precision-Recall curve,

Libraries: [https://scikit-learn.org/stable/modules/model_evaluation.html (https://scikit-learn.org/stable/modules/model_evaluation.html)](https://scikit-learn.org/stable/modules/model_evaluation.html)

1. `log_loss`, `hinge_loss`.
2. `metrics.precision_score`, `metrics.recall_score`
3. `metrics.average_precision_score`, `metrics.roc_auc_score`

Types of binary predictions

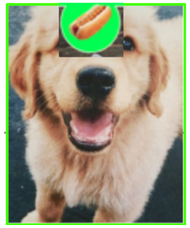
```
- TN: True negative
- FN: False negative
```

```
- FP: False positive
- TP: True positive
```

The following table is called the **confusion matrix**.

		classifier calls	
		“negative” C=0	“positive” C=1
true outcome	Y=0	TN	FP
	Y=1	FN	TP

Examples

Hot Dog	Not Hot Dog
	
	

Evaluating correct predictions:

- **Accuracy**: fraction of correct predictions

$$\frac{TP + TN}{TP + TN + FN + FP}$$

- **Potential problem**:

```
Suppose your dataset has 1 positive example and 99 negative examples.

What is the accuracy of the classifier that always predicts ”negative”?
```

Other evaluation metrics:

METRIC	FORMULA	IN WORDS **"Probability that …"** **Or** **"How often the …"**	EXPRESSION
True Positive Rate (TPR) sensitivity recall	$\frac{TP}{TP + FN}$	subject who is positive will be called positive	Pr(C = 1 | Y = 1)
True Negative Rate (TNR) specificity, 1-FPR	$\frac{TN}{FP + TN}$	subject who is negative will be called negative	Pr(C = 0 | Y = 0)
Positive Predictive Value (PPV) precision	$\frac{TP}{TP + FP}$	subject called positive will actually be positive	Pr(Y = 1 | C = 1)
Negative Predictive Value (NPV)	$\frac{TN}{TN + FN}$	subject called negative will actually be negative	Pr(Y = 0 | C = 0)

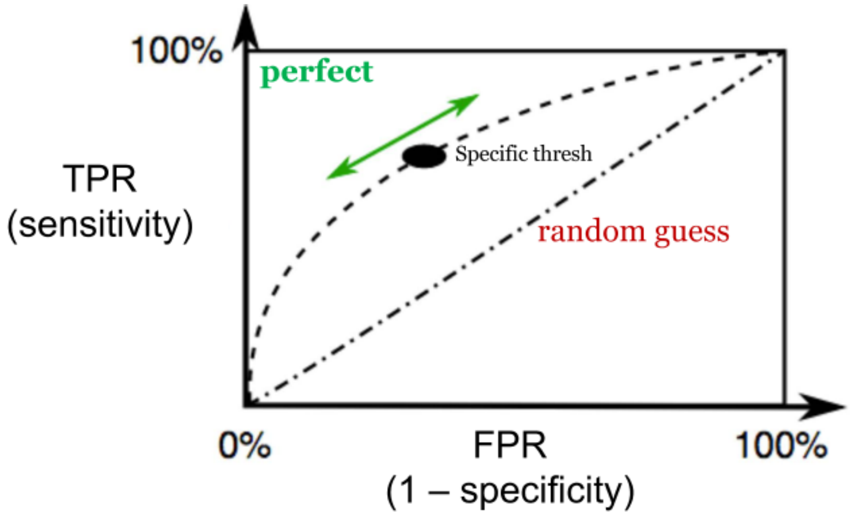
Example applications for evaluation choice

1. App to classify cats vs. dogs from images?

1. Classifier to find relevant tweets to list on website?

1. Detector for tumors based on medical image?

ROC Curve (across thresholds)



Area under ROC curve

(also called AUROC or AUC or "C-statistic")

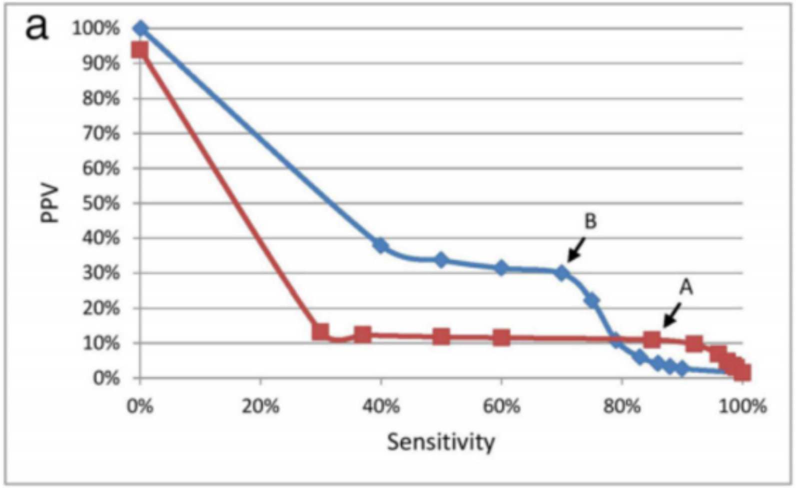
- Area varies from 0.0 to 1.0
- 0.5 is random guess
- 1.0 is perfect

- **Probabilistic meaning**:
  $\mathrm{AUROC} \doteq \Pr(\hat{y}_i > \hat{y}_j | y_i = 1, y_j = 0)$

For random pair of examples, one positive and one negative. What is the probability that the classifier will rank the positive one higher?

Precision-Recall Curve

precision

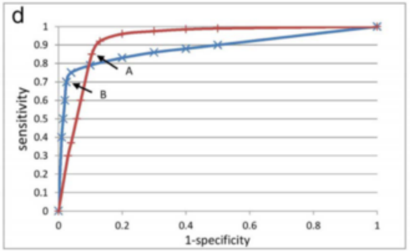


recall (= TPR)

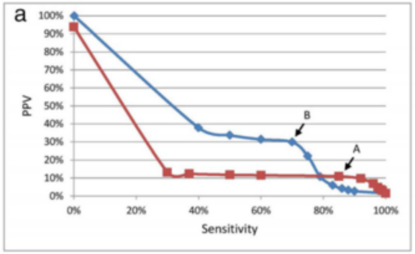
AUROC not always the best choice

Why the *C*-statistic is not informative to evaluate early warning scores and what metrics to use

Santiago Romero-Brufau[1,2*], Jeanne M. Huddleston[1,2,3], Gabriel J. Escobar[4] and Mark Liebow[5]



AUROC: red is better



Blue much better for alarm fatigue

Summary of Methods

Methods	Function class flexibility	Knobs to tune	Interpret?
Logistic Regression	Linear	L2/L1 penalty on weights	Inspect weights
Decision Tree Classifier	Axis-aligned, Piecewise constant	Max. depth, Min. leaf size, Goal criteria	Inspect tree
K Nearest Neighbors Classifier	Piecewise constant	Number of Neighbors, distance metric, how neighbors vote	Inspect neighbors

In [ ]:

```

```