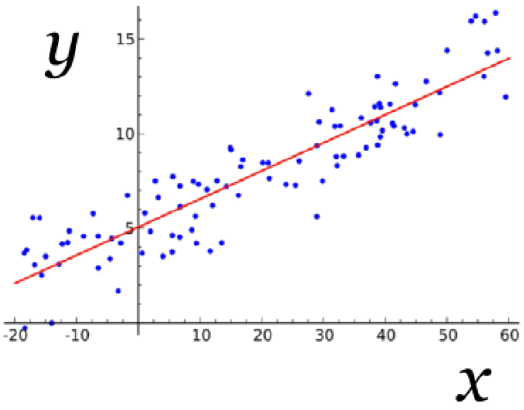
Classification

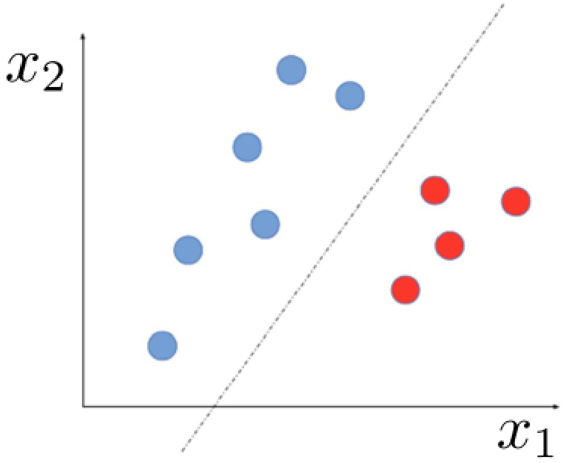
Machine Learning and Computational Statistics (DSC6135)

1. Introduction to Classification

y is a numeric variable
e.g. sales in $\$\$$



y is a binary variable
(red or blue)



Example: Binary 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. Decission Tree Regression
    3. K-Nearest Neighbors

Binary Prediction Step

Goal: Predict label (0 or 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: $y_i \in \{0, 1\}$

  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

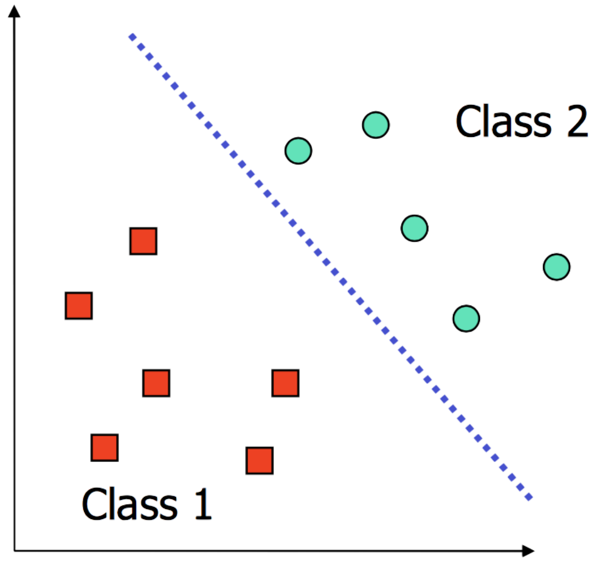
```
>>> # Given: pretrained regression object model
>>> # Given: 2D array of features x

>>> 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.



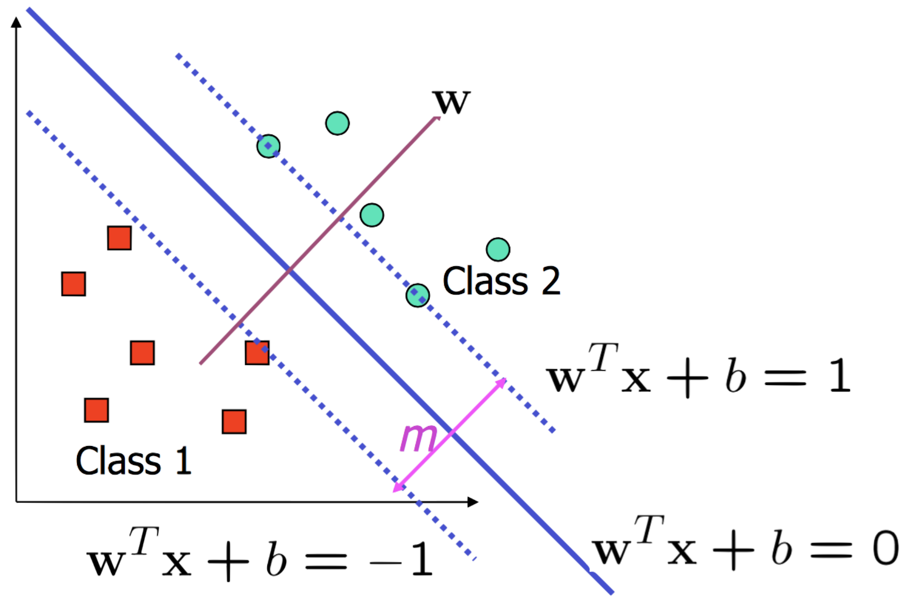
Logistic Regression: Linear Decision Boundary

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

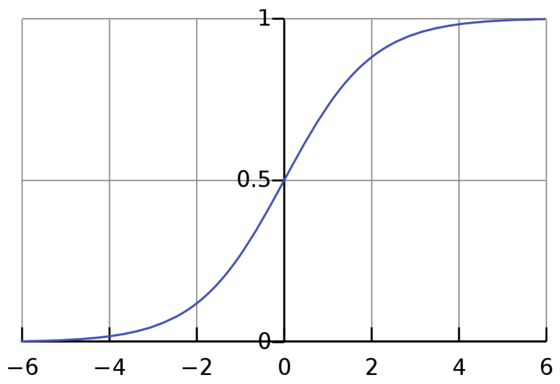
$$\mathbf{w}^\top \mathbf{x} = w_0 x_0 + w_1 x_1 + \ldots + w_D x_D = 0$$

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***:



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

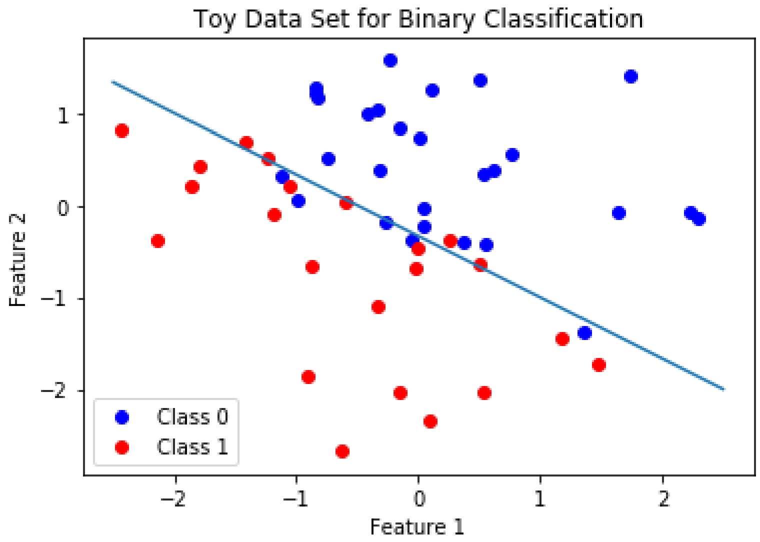
$$\text{Prob}[y = 1|\mathbf{x}] = \sigma(\mathbf{w}^\top \mathbf{x})$$

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 [6]:
```
x1 = np.random.randn(50, 1)  # some continuous features
x2 = np.random.randn(50, 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:

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

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

- 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:

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

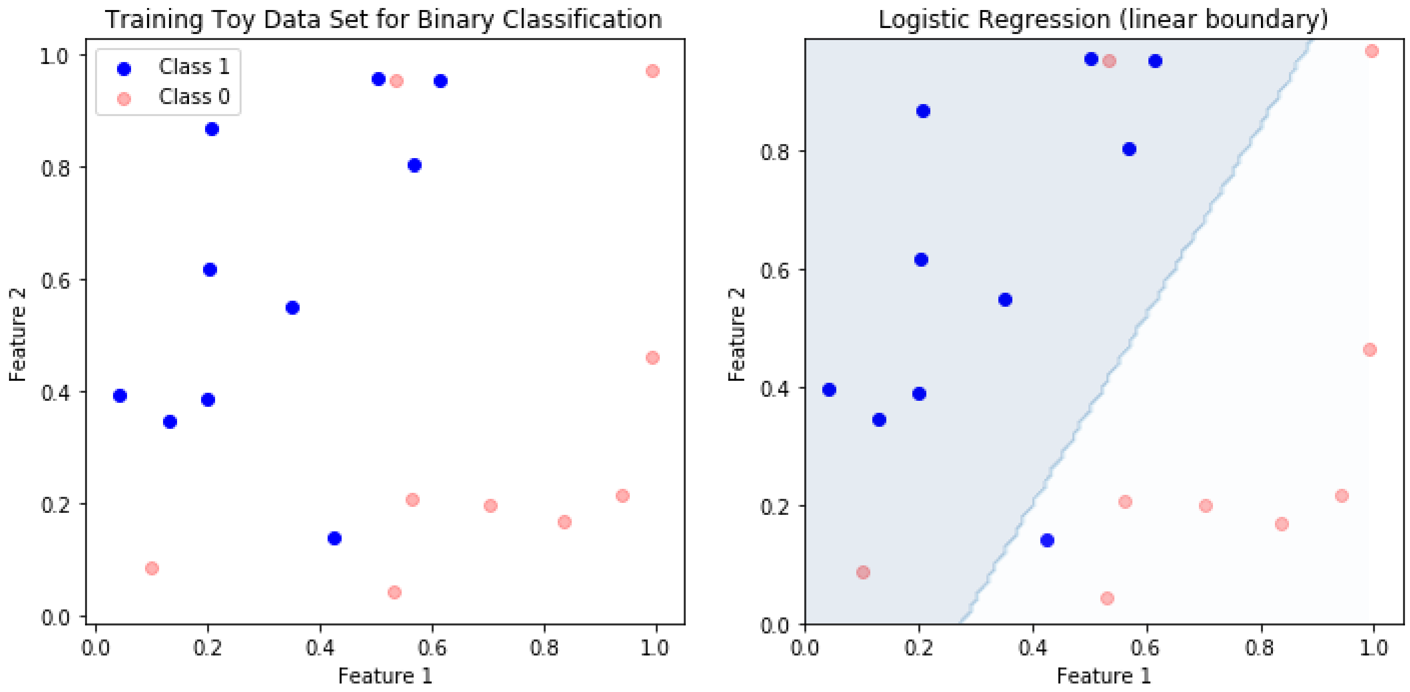
$$\max_{\mathbf{w}} \sum_{i=1}^{N} \left[y_i \log(1 + e^{-(\mathbf{w}^T x_i)}) + (1 - y_i) \log(1 + e^{(\mathbf{w}^T x_i)} \right]$$

Example with sklearn:

In [8]:
```
# create a logistic regression model with linear boundary
logreg = linear_model.LogisticRegression(C=1.0, solver='lbfgs')
# fit our logistic regression model
logreg.fit(X_train, y_train)
```

Out[8]:
```
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 [10]:
```
# plot data and decision boundary of logistic regression
plot_fig(X_train,y_train)
```



In [11]:

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

Out[11]:

	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:
$$\mathbf{x} = [1\ x_1\ x_2\ \ \dots\ \ x_F]^T$$
  - Weight vector:
$$\mathbf{w} = [w_0\ w_1\ w_2\ \ \dots\ \ w_F]^T$$
  - 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:

$$\frac{\partial \sigma(z)}{\partial z} = \frac{\partial}{\partial z}(1+e^{-z})^{-1} = e^{-z}(1+e^{-z})^{-2} = \frac{1}{1+e^{-z}}\frac{e^{-z}}{1+e^{-z}}$$
$$= \sigma(z)(1-\sigma(z))$$

- 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

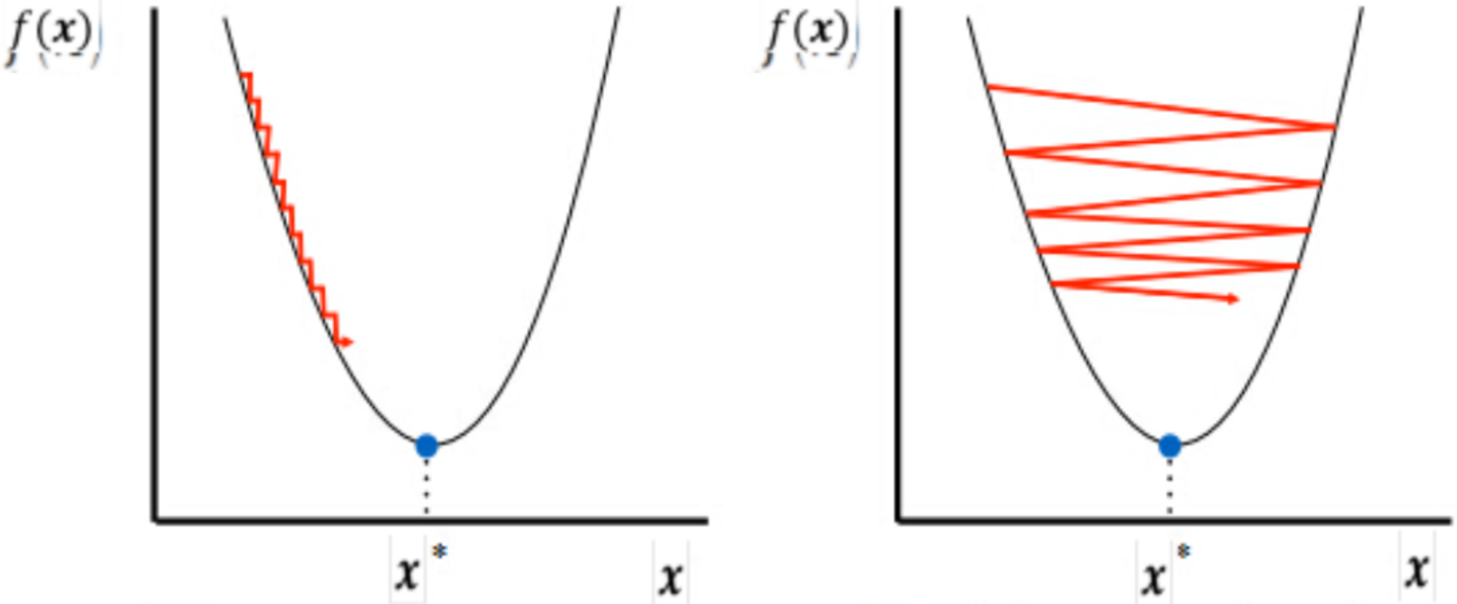
$$\frac{\partial l_i(\omega)}{\partial \omega^T} = -y_i x_i(1-\sigma(z_i)) + (1-y_i)x_i\sigma(z_i) = x_i(\sigma(z_i)-y_i)$$

- Careful explanation: [stackexchange (https://stats.stackexchange.com/questions/68391/hessian-of-logistic-function)](https://stats.stackexchange.com/questions/68391/hessian-of-logistic-function)

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.

Intuition in 1D



Too small: converge very slowly

Too big: overshoot and even diverge

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

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

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

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

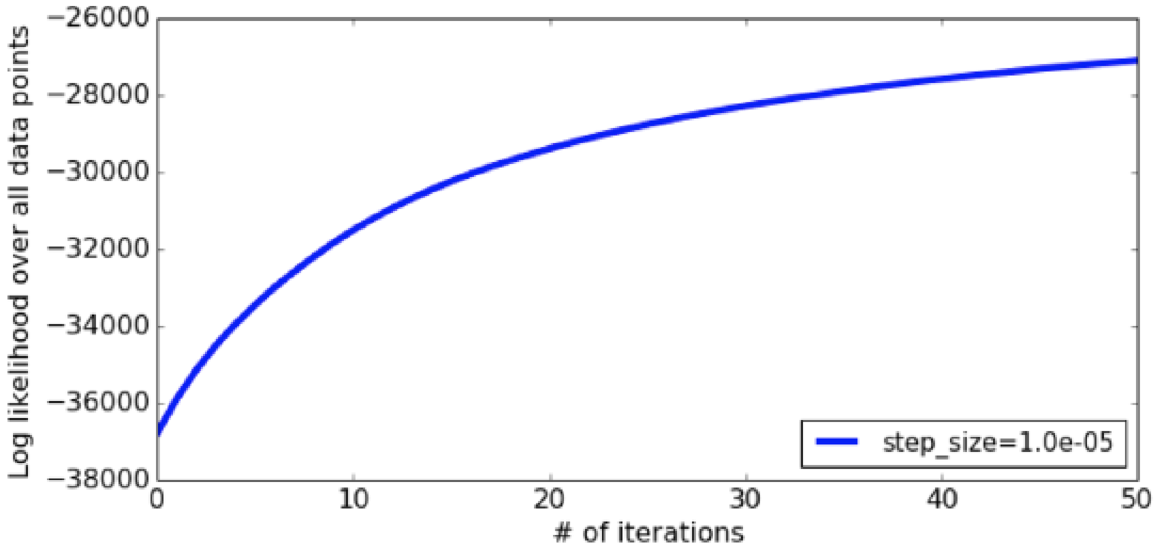
$\quad w_0^{t+1} = w_0^t - \mathrm{s} \, \nabla J(\boldsymbol{w}^t, w_0^t)$

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

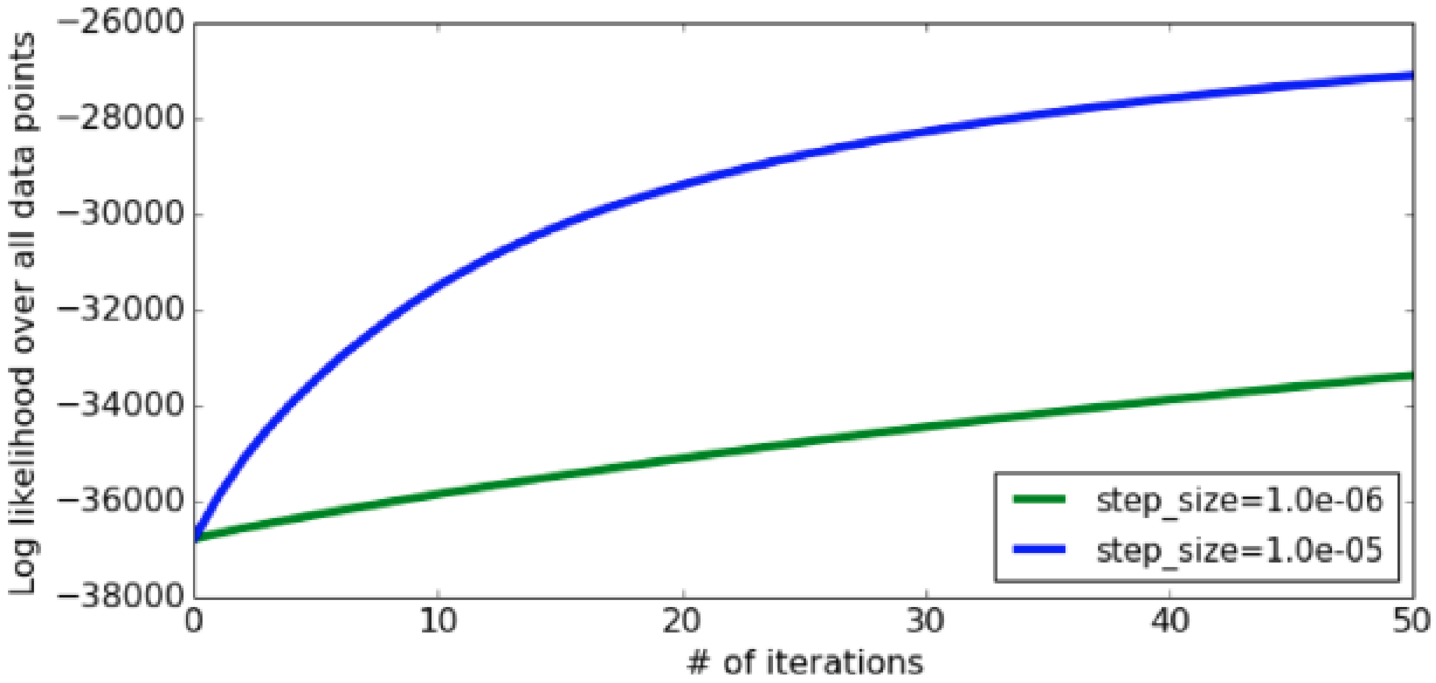
$\quad\quad$ break

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

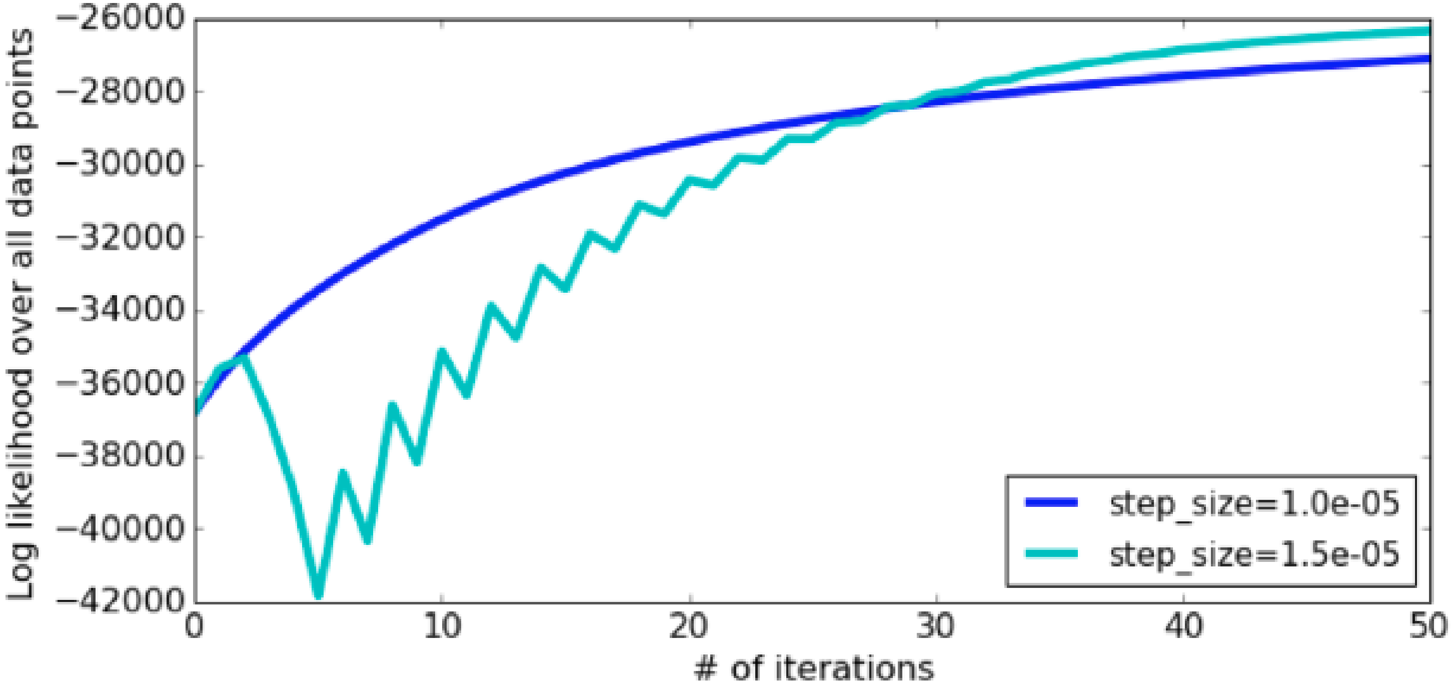
Step size tuning



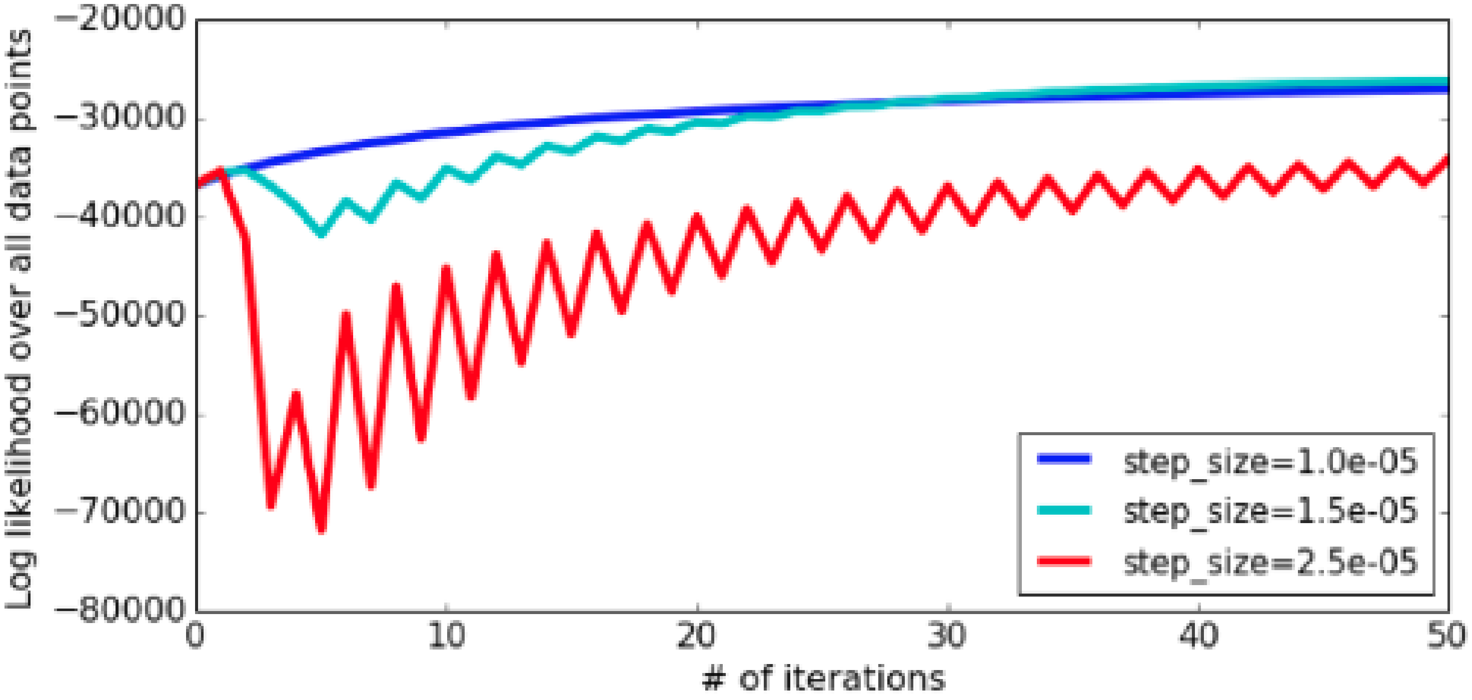
Step size tuning: too small



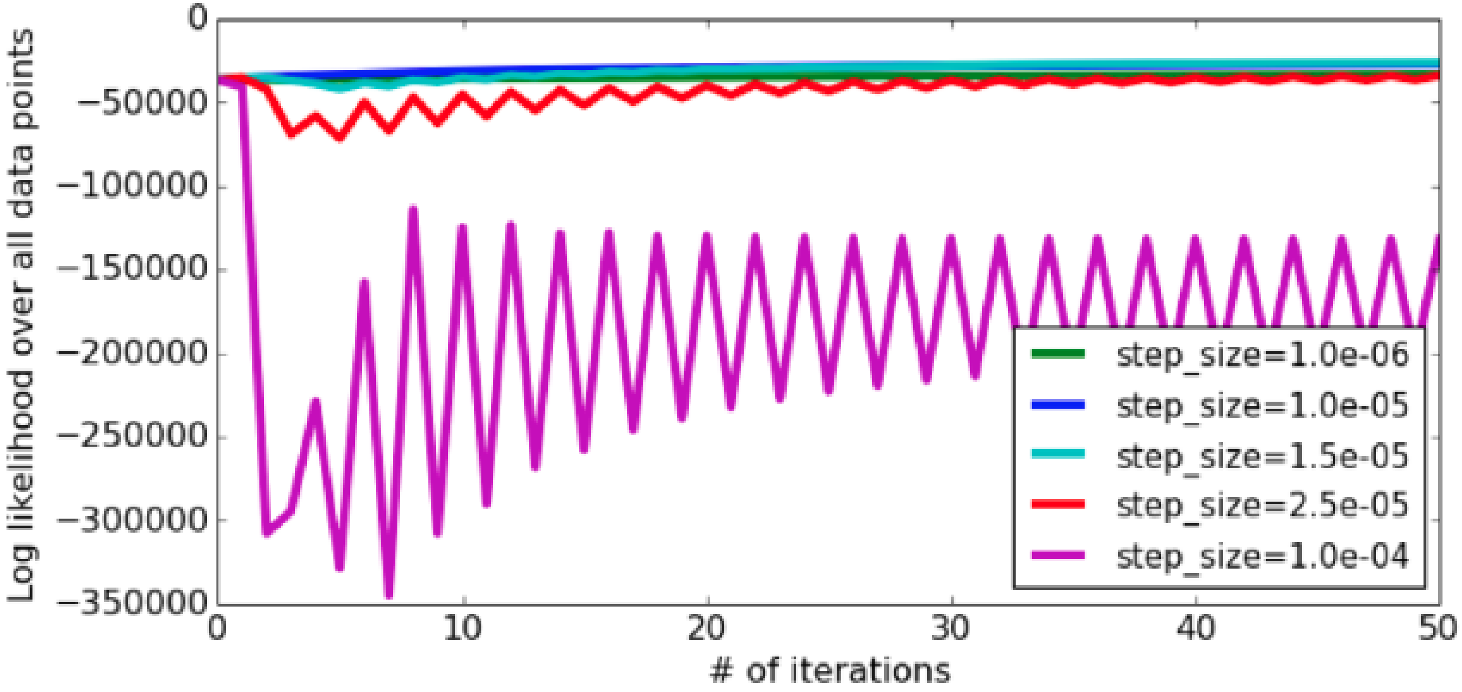
Step size tuning: large



Step size tuning: too large



Step size tuning: way tooo large



Logistic Regression: Non-Linear Decision Boundary

Go to external notebook.

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 [ ]: