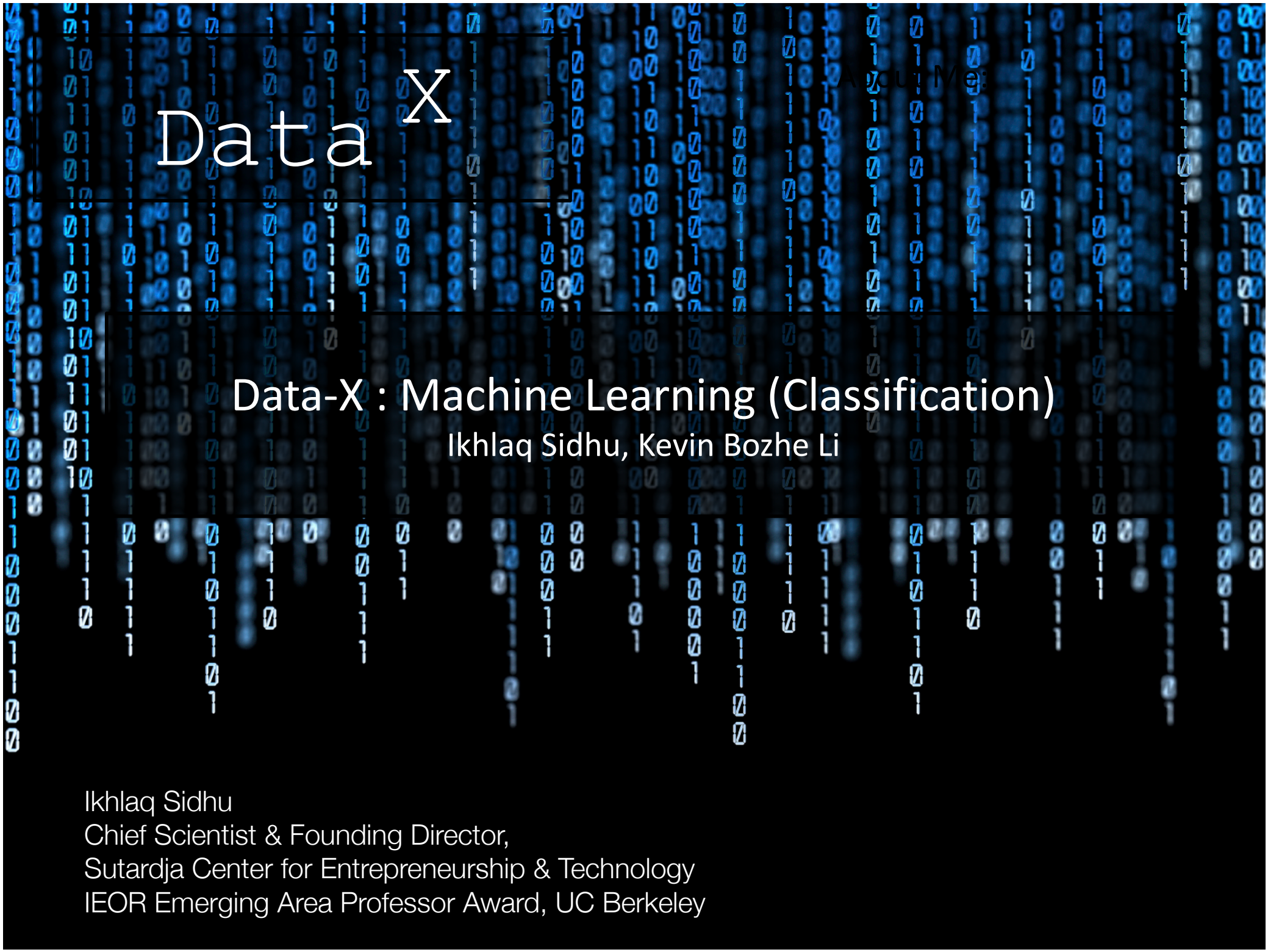


Data-X : Machine Learning (Classification)

Ikhlaq Sidhu, Kevin Bozhe Li

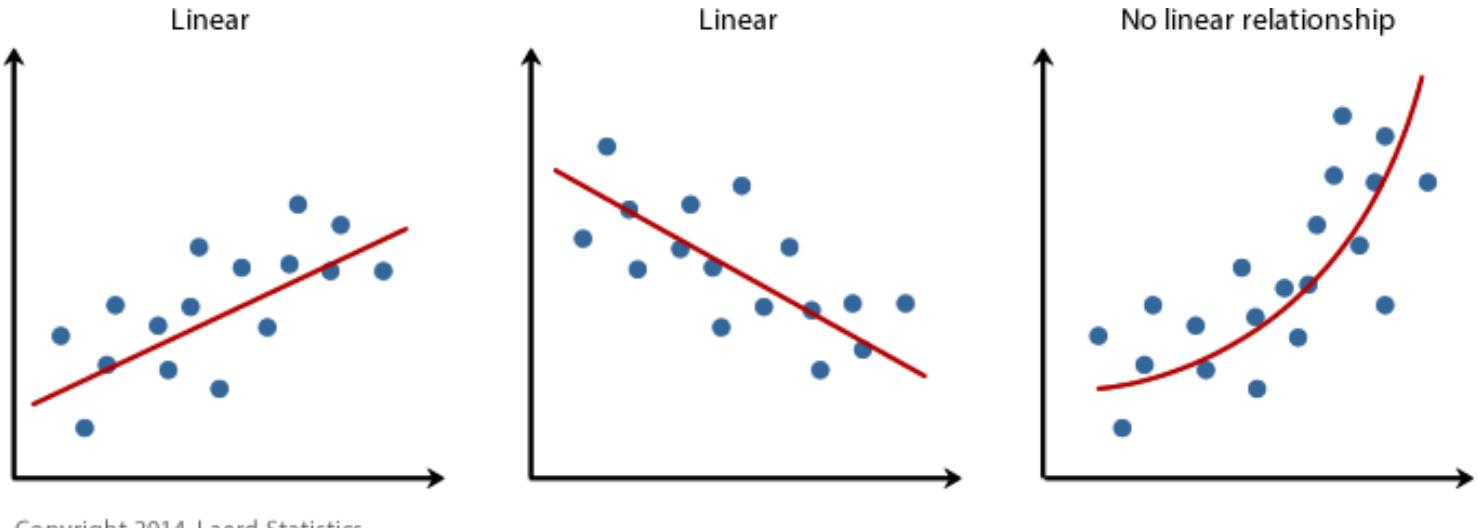
Ikhlaq Sidhu
Chief Scientist & Founding Director,
Sutardja Center for Entrepreneurship & Technology
IEOR Emerging Area Professor Award, UC Berkeley

Agenda

- Discussion of Classification, Logistic Regression, and how it works in SciKit Learn (45 min to 1 hour)
- Break Out Session with Sample Code (40 min)
- Homework for next week: Lighter to make more progress on project
- Project Discussion – Next steps, work with team:
  - User requirements: what you know and don't?
  - Technical needs: what you know and don't
  - Whiteboard Your Architecture and /or Data Model
  - Basic planning and prioritization

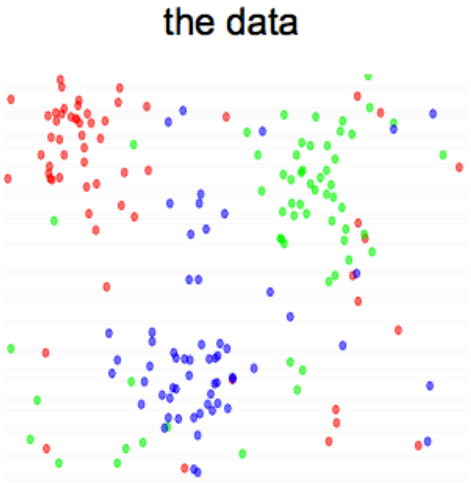
Prediction vs Classification

Prediction:
Linear or
Non-linear

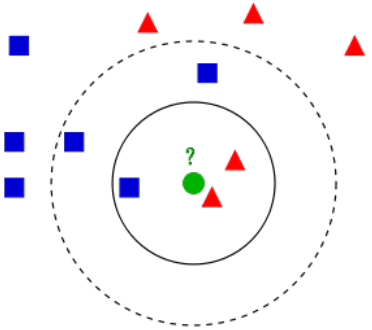


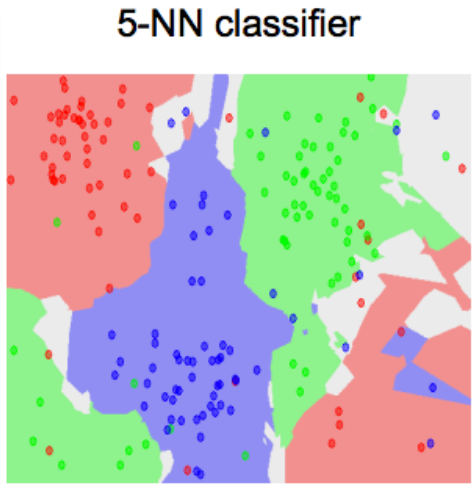
Classification:
Can also be
Linear or
Non-linear

Ex. KNN, K-means
Logitstic, SVM, ..



KNN Method: Find the k nearest images and have them vote on the label (i.e. take the mode)

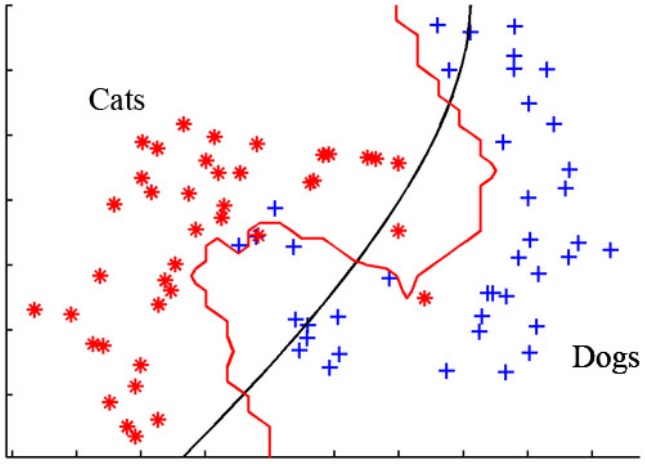


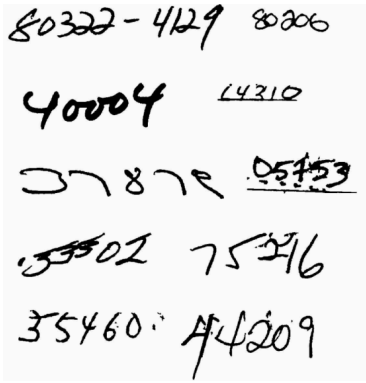


Classification

Examples

- Radar: WWII
- Spam Detection
- Image Classification: Cats VS Dogs
- Image Classification: Recognizing Digits





Classifiers: A Whirlwind Tour

- **Linear Classifiers**
  - The simplest **parametric** classifier
  - One must define a **loss function** (a way to measure the error) in order to optimize the weights of a linear classifier
  - **Regularization** is your friend!
  - Examples: *Support Vector Machine (SVM) & Logistic Classifier*

- **Nonlinear Classifiers**
  - Introduction to **Neural Networks** (an universal function approximator)

Reminder from Previous Class:
In the **ML framework**, there is no limit to the predictors or classifiers that can be used.

In Sample Data which we have
- d features
- N samples

Use for training

$$X \ (Nxd) \rightarrow Y \ (N \times 1)$$

Results in Y (which we know)

Sometimes measured
Y includes error or noise

We don't know:

P(X), the distribution of X

Function f:
f: X->Y

We use this to train:
(x1,y1), (x2,y2)…(xN,yN)

H = Hypothesis Set
All Possible Algorithms
Candidate Formulas

Algorithm:
Under test

Final Hypothesis

g: x->y

predictor
or
classifier

Error measure

- We try different functions g until
- *g(x) is close to f(x)*
- For any out of sample **x**, we can predict **Y** or classify it

Our Goal: To classify items.

xi → Model: f(x,W)

Actual Results:

$y_i = [\, y_{i,1}, y_{i,2}, .. y_{i,k} \,]$

$y_i = [\, +1 , -1, .. -1 \,]$

or

$y_i = [\, +1 , 0, .. 0 \,]$

We have this:
(X,Y): (x1,y1), (x2, y2) .. (xn,yn)

- xi is a vector (or even matrix) for each data element
- For a picture: $\mathbf{x_1 = [32 \times 32 \times 3]}$: array of numbers
- yi = (-1,+1)

y(i,1) = 1 if picture is dog, else -1*

Y(i,2) = 1 if picture is cat, else -1*

* Sometimes 0 vs 1 instead of -1 vs 1

Our Goal: To classify items.

We have this: (X,Y)

xi → Model: f(x,W)

Actual Results:

$y_i = [\, y_{i,1}, y_{i,2}, .. y_{i,k} \,]$

$y_i = [\, +1 , -1, .. -1 \,]$

And sometimes mapped to

$y_i = [\, +1 , 0, 1.. 0 \,]$

Machine Learning Steps to train a classifier model

1. Choose our model: $f(x_i, W)$ = our estimate of Y

   If f(x) is a Linear model: $f(i) = f(x_i) = Wx_i = w_1 x_{i,1} + w_2 x_{i,2} \ldots$
   W is list of parameters.
   Results in a different score for each sample point (picture) x_i

2. Define a loss function (L)

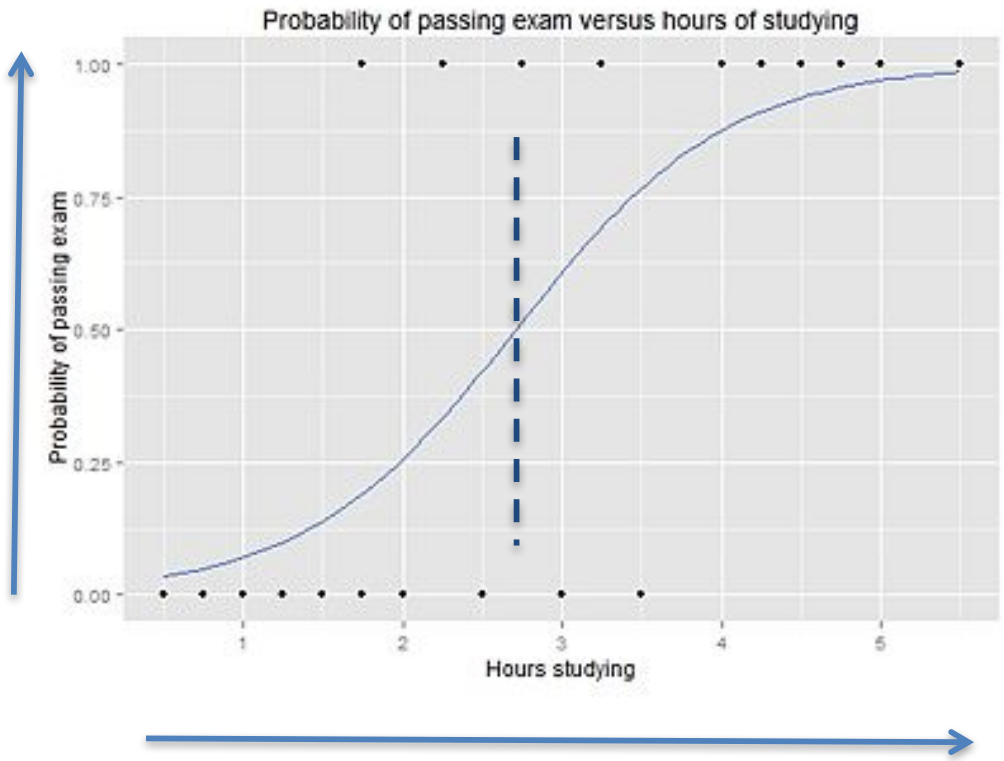
3. Optimize across the parameter space (W) to minimize the loss function to

Example Classification with Logistic Regression

Data: students study for an exam:
(x= hours studied, y = pass/not pass)

Hours	0.50	0.75	1.00	1.25	1.50	1.75	1.75	2.00	2.25	2.50	2.75	3.00	3.25	3.50	4.00	4.25	4.50	4.75	5.00	5.50
Pass	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1

y (or t):
0 = fail
1 = pass



x

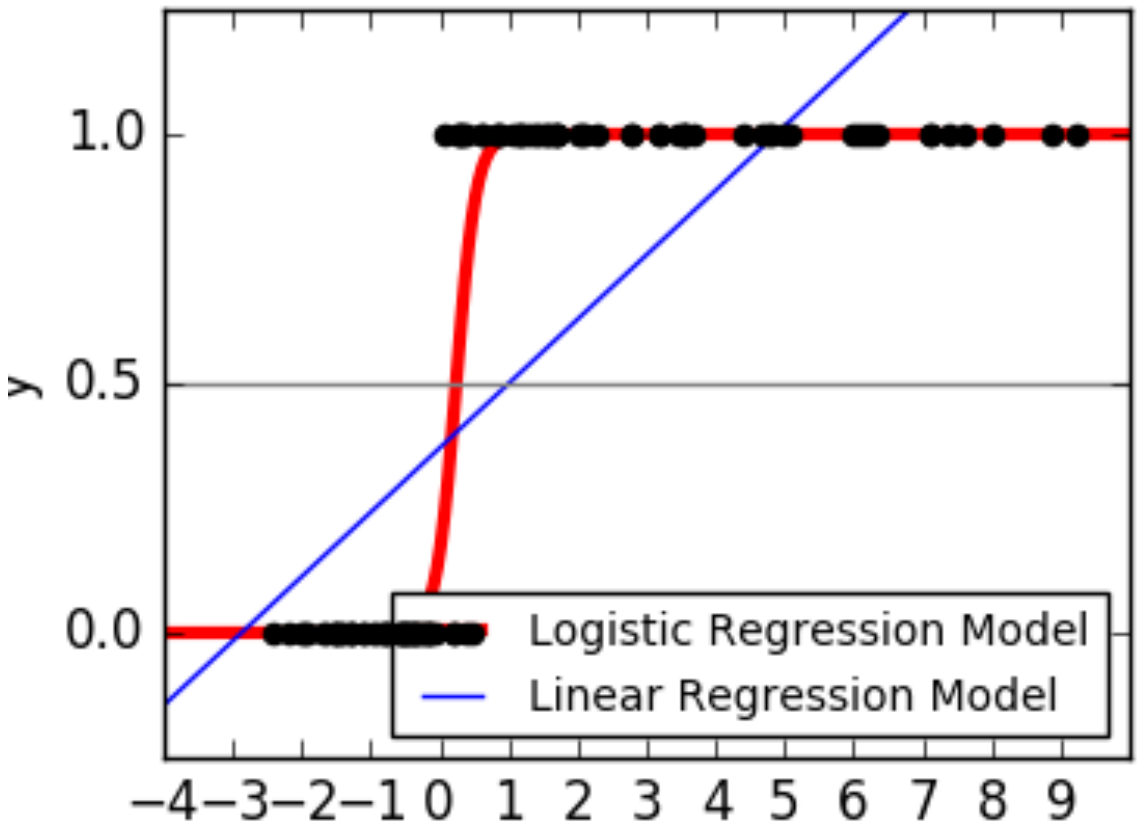
In this case output is binary (pass or not pass)

Problem:
Student studies x hours
Will the student pass

We use this curve to predict the probability that the student would pass given x hours of study

If Prob > 0.5, we classify student will pass the exam

Example: Logistic Regression
We use Logistic Regression because
a line is not a good estimator for binary results (classification)

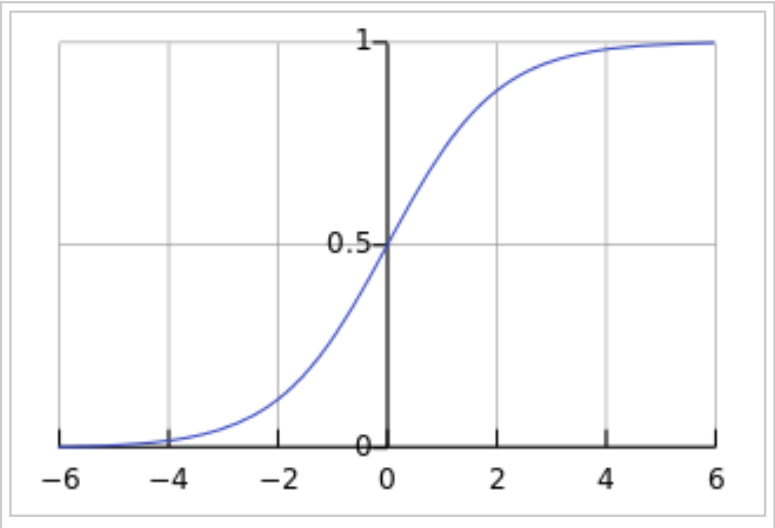


The logit function is a better fit for binary results

This function is a better fit for binary results:

$$f(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

Large t -> 1
Small t -> 0



This function results in (0,1)
for all real numbers (like a probability)

And if t has this form:

$t = w_0 + x_1 w_1$

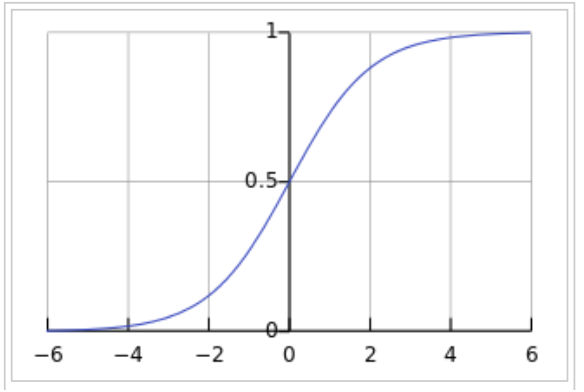
$$f(x_i, w0, w1) = \frac{1}{1 + e^{-(w0 + x1w1)}}$$

- Think of f(x,w) as probability y = 1 given any x
- Prob (y=1) = ½ when $e^{-(w0 + xi,1\ w1)} = 1$, ie $w_0 + x_1 w_1 = 0$
- Choose w0, w1,..wd to get best fit
- Still need a loss function and then solve for best W

A Loss Function Commonly Used for Logistic Regression

Cross Entropy $= -t\ln(f(\vec{x})) - (1-t)\ln(1-f(\vec{x}))$

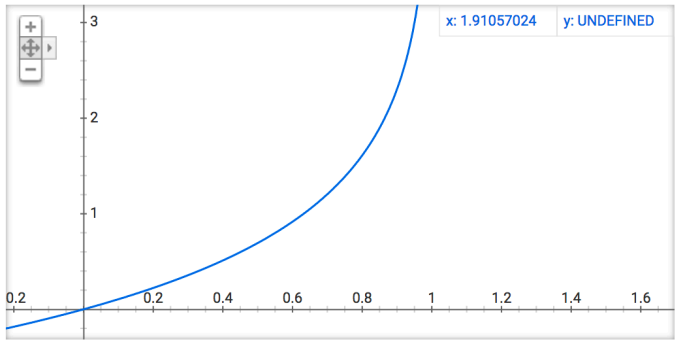
If y is -1 or +1, the we can map to t: (like actual value y)
t = (1+y)/2, so t ranges from 0 to 1



$$f(x_i, w0, w1) = \frac{1}{1 + e^{-(w0 + x1w1)}}$$

-ln(1-f(x)) when t = 0

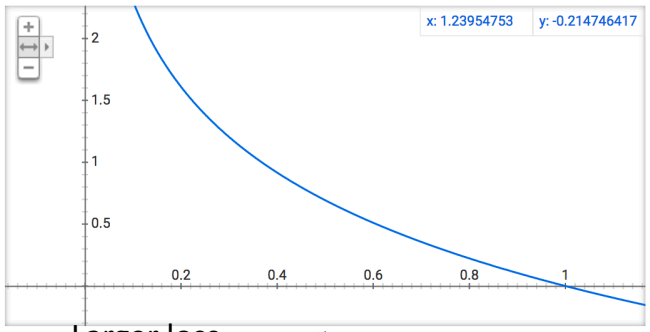
if t = 0: loss = -ln(1-f(x))
if f(x) is 1 -> loss = e,
if f(x) is 0 -> loss = 0



Larger loss

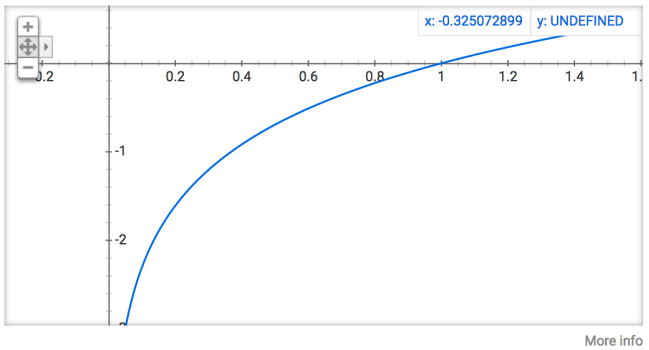
-ln(f(x)) when t = 1

If t = 1: loss = - ln(f(x))
if f(x) is near 0 ->
loss = large
if f(x) is near 1 ->
loss = approx. = 0



Larger loss

Graph for ln(x)

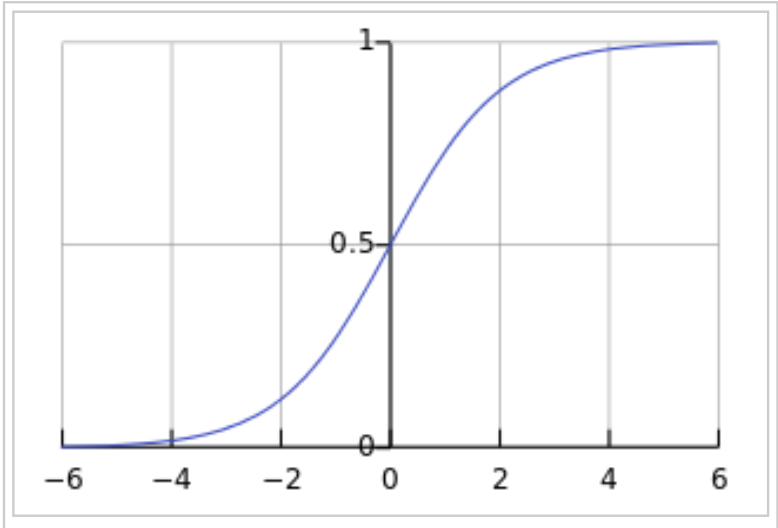


The logit function can be extended for multiple features (dimensions)

This function is a better fit for binary results:

$$f(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

Large t -> 1
Small t -> 0



This function results in (0,1)
for all real numbers (like a probability)

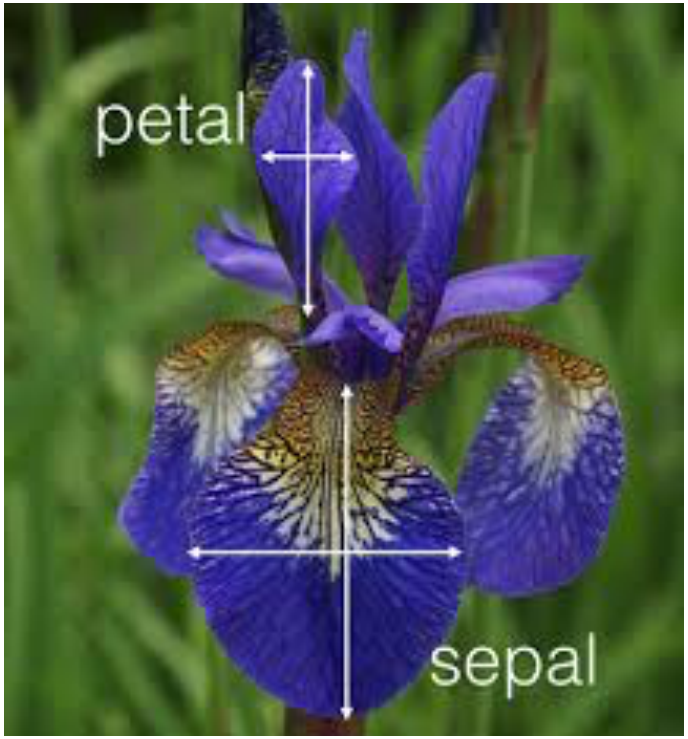
And if t has this form:

$t = w_0 + x_1 w_1$
or
$t = W x_i$ (in matrix form)
$= w_1 x_{i,1} + w_2 x_{i,2} \ldots$

- Easily extends to multiple features (x1, x2, ..)
- And multiple parameter weights

$$f(x_i, W) = \frac{1}{1 + e^{-(w0 + xi,1\ w1 + xi,2\ w2 \ldots)}}$$

Example Code Sample with Logistic Regression Classifier



Input data

X: Attribute Information:
- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm

Y:
0 = 'setosa',
1 = 'versicolor',
2 = 'virginica’

```
print type (X)
print X[0:5]

<type 'numpy.ndarray'>
[[ 5.1  3.5  1.4  0.2]
 [ 4.9  3.   1.4  0.2]
 [ 4.7  3.2  1.3  0.2]
 [ 4.6  3.1  1.5  0.2]
 [ 5.   3.6  1.4  0.2]]


print Y[0:5]
[0 0 0 0 0]
```

Example Code Sample with Logistic Regression Classifier

```
import numpy as np
1 → from sklearn import linear_model, datasets

X = iris.data[:, 1:3]  # only the first two features.
Y = iris.target

# https://en.wikipedia.org/wiki/Logistic_regression
2 → logreg = linear_model.LogisticRegression(C=1e5)

# we create an instance of Neighbours Classifier and
fit the data.
3 → logreg.fit(X, Y)

# predict a category for every row in X
4 → Z = logreg.predict(X)
```

Class sklearn.linear_model.LogisticRegression

(penalty='l2',
dual=False,
tol=0.0001,
C=1.0,
fit_intercept=True,
intercept_scaling=1,
class_weight=None,
random_state=None,
solver='liblinear', max_iter=100,
multi_class='ovr', verbose=0,
warm_start=False,
n_jobs=1)

http://scikit-learn.org/stable/modules/generatedsklearn.linear_model.LogisticRegression.html

* Z[2] will be the predicted number for row X[2]

fit(*X*, *y*, *sample_weight=None*) [source]

Fit the model according to the given training data.

Parameters: **X** : {array-like, sparse matrix}, shape (n_samples, n_features)

Training vector, where n_samples is the number of samples and n_features is the number of features.

y : array-like, shape (n_samples,)

Target vector relative to X.

sample_weight : array-like, shape (n_samples,) optional

Array of weights that are assigned to individual samples. If not provided, then each sample is given unit weight.

New in version 0.17: sample_weight support to LogisticRegression.

Returns: **self** : object

Returns self.

Fit and predict from ScikitLearn

predict(*X*) [source]

Predict class labels for samples in X.

Parameters: **X** : {array-like, sparse matrix}, shape = [n_samples, n_features]

Samples.

Returns: **C** : array, shape = [n_samples]

Predicted class label per sample.

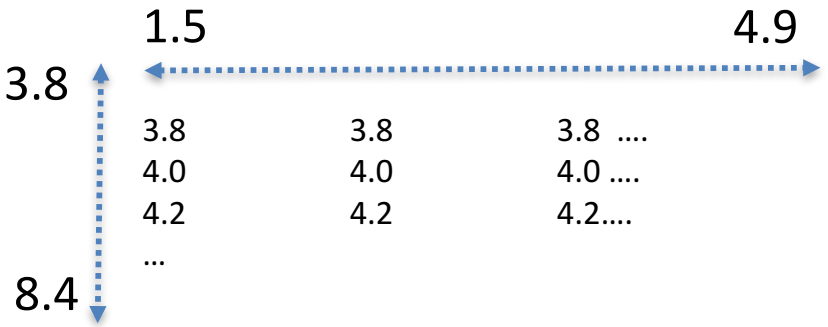
Code Samples with SciKit Learn

```
# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])
# numpy.ravel: Return a contiguous flattened array.
```

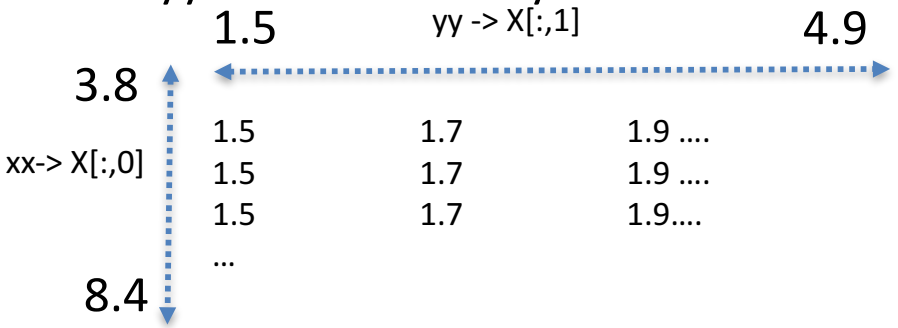
xx shape is (171, 231)
yy shape is (171, 231)

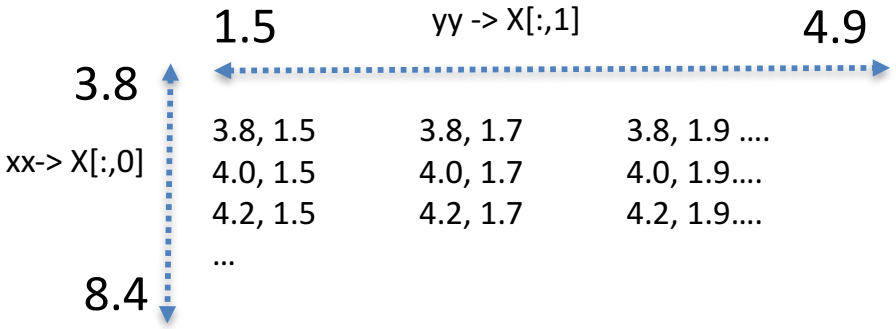
np.c returns shape (39501, 2)
[[3.8 1.5]
[3.82 1.5]
[3.84 1.5] …]
Z shape is (39501,)

xx is a matrix of all the first values



yy is matric of only the second values





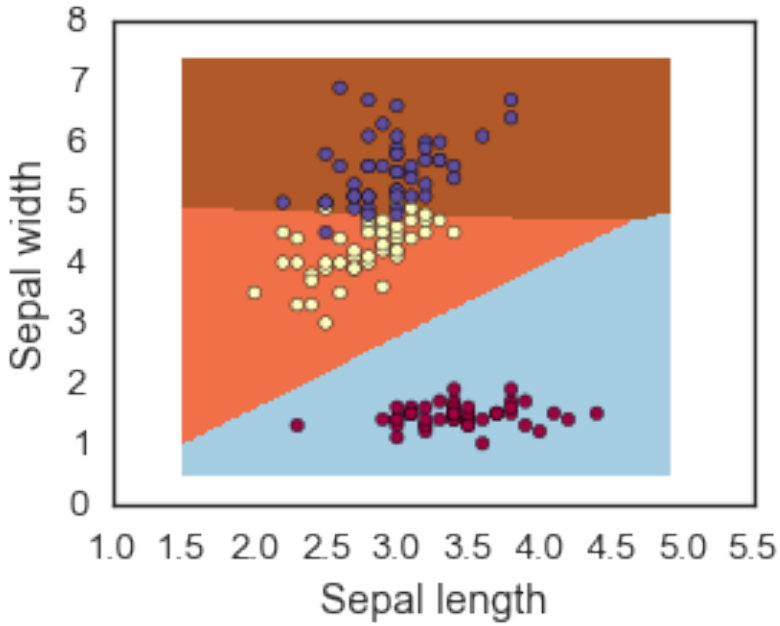
Plotting the Results

```
# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='k',
cmap=get_cmap("Spectral"))
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

#plt.xlim(xx.min(), xx.max())
#plt.ylim(yy.min(), yy.max())
#plt.xticks(())
#plt.yticks(())

plt.show()
```



Loss Functions

xi = [32 x 32 x 3]:
array of numbers

1. If f(x) is our selected model

If f(x) is a Linear:

$$\text{Score}(i) = f(x_i, W) = Wx_i$$

Actual Results:

$y_i = [\, y_{i,1}, y_{i,2}, .. y_{i,k} \,]$

$y_i = [\, +1 , -1, .. -1 \,]$

Score

Score $f(x_i, W)$ typically ranges from (-1,+1)

2. Loss Function = price of inaccuracy of score:

Loss

- example: $L_i(f(xi), W), y) = (1 - yf(\vec{x}))^2$ (square loss)
- See that y f(x) approx =
  - +1 for matches: y=1, f(x)=1 or y=-1, f(x) = -1)
  - -1 for does not match: y=1, f(x)=-1, y=-1, f(x)=1

$L_i(f(xi, W), yi)$ = must be selected (there are many options)
Loss L = E[Li] = average all i sample data during training or testing

Other Famous Loss Functions

- Logistic Loss $= \frac{1}{\ln 2} \ln(1 + e^{-yf(\vec{x})})$

y f(x) = 1 for match: proportional to exp(1+e^{-1})
y f(x) = -1 for no match, proportional to exp (1+e^{1})

- Cross Entropy $= -t \ln(f(\vec{x})) - (1 - t) \ln(1 - f(\vec{x}))$

t = (1+y)/2, so t ranges from 0 to 1

if t = 0: loss = -ln(1-f(x)) = e, when f(x) is 1

If t = 1: loss = - ln(f(x) = large when score f(x) is near 0
approx. = 0, when f(x) = 1

Note: Used for Logistic Regression

After we have a loss function, we need to choose W (parameters) to min the loss

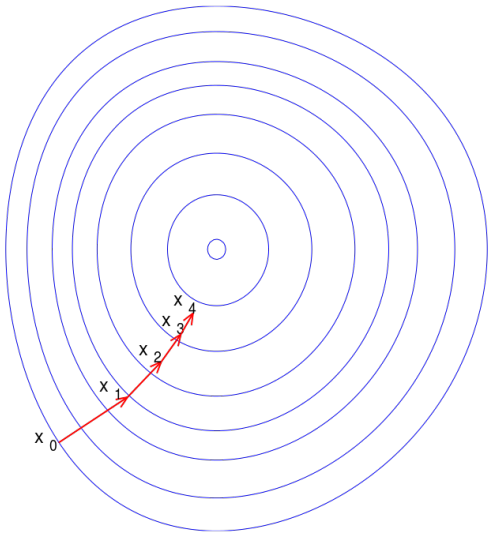
L(X, W, Y) = Loss is a function of f(x,W) and Y

ML algorithms try a random W

Expected Loss is calculated for small changes in W

$W_{i+1} = W_i$ – [(step size gamma) x - gradient of L(W)]

Gradient Descent



Gradient descent is a first-order iterative optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point.

End of Section

Regularization

Why: To avoid over-fitting

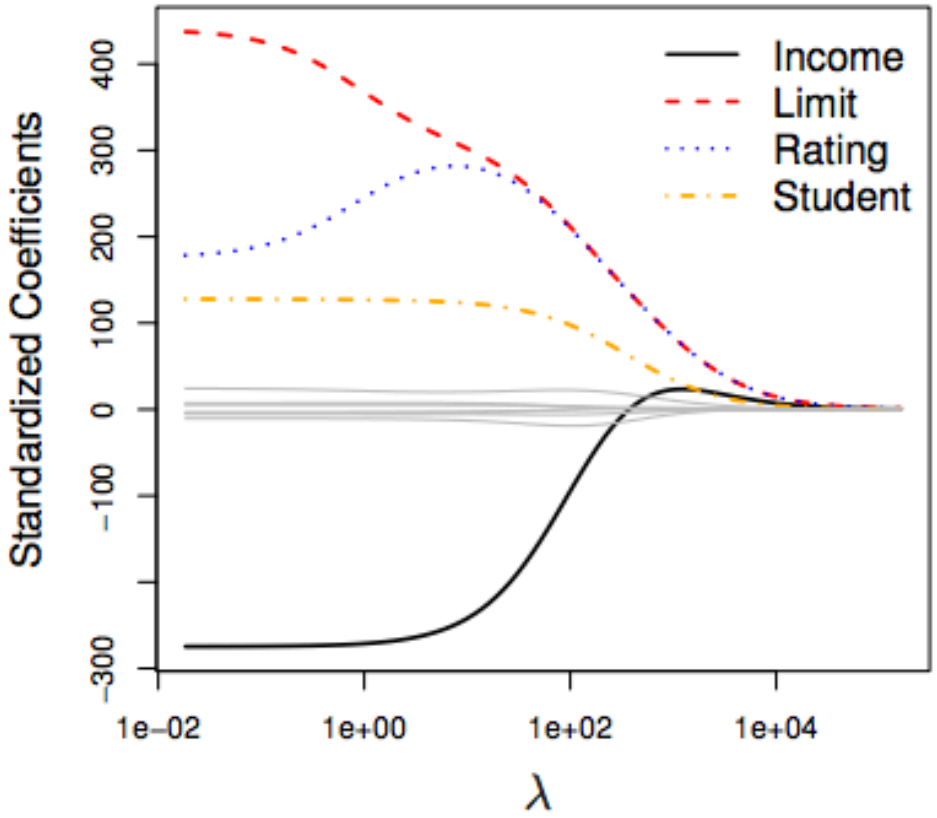
How: You penalize your loss function by adding a multiple of an L1 (LASSO) or an L2 (Ridge) norm of your weights vector w

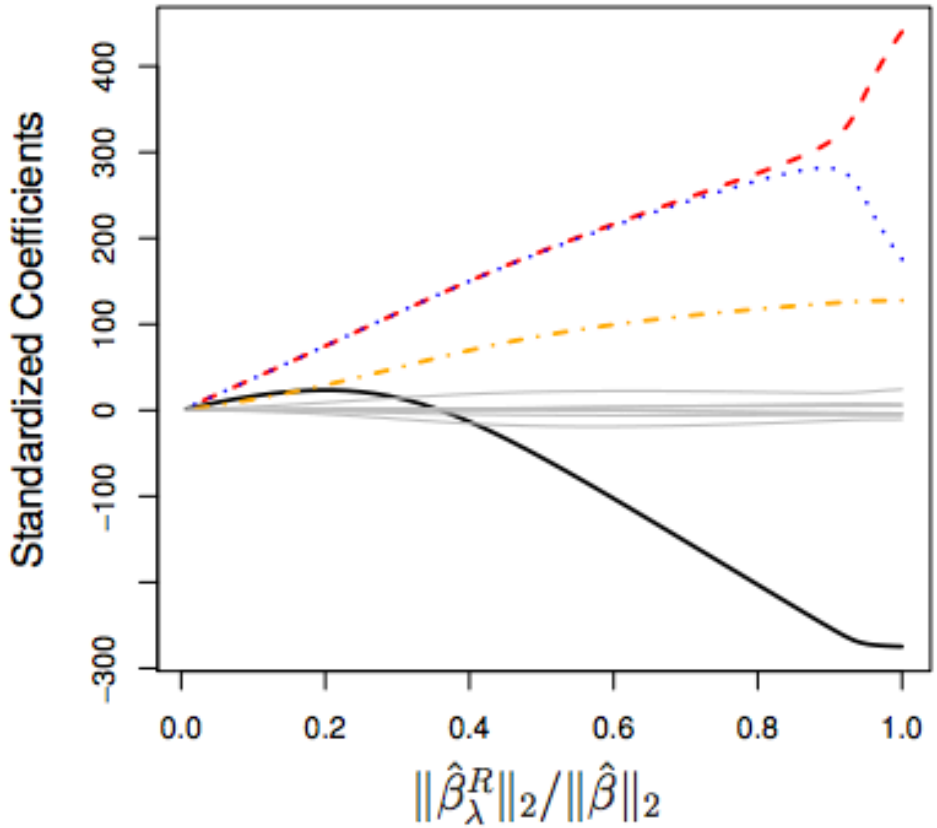
Your new loss function = $L(X,Y) + \lambda N(w)$

Tuning the regularization term λ: Cross-validation:

- divide your training data,
- train your model for a fixed value of λ and test it on the remaining subsets
- repeat this procedure while varying λ.
  Then you select the best λ that minimizes your loss function.

Shrinkage Methods II: An example





References

- The material presented in this lecture references lecture material draws on the materials the following courses:
- UC Berkeley – CS 294-129 (Designing, Visualizing, and Understanding Deep Neural Networks): https://bcourses.berkeley.edu/courses/1453965/pages/cs294-129-designing-visualizing-and-understanding-deep-neural-networks
- Stanford – CS231n (Convolutional Neural Networks for Visual Recognition): http://cs231n.stanford.edu/