# What is Machine Learning?

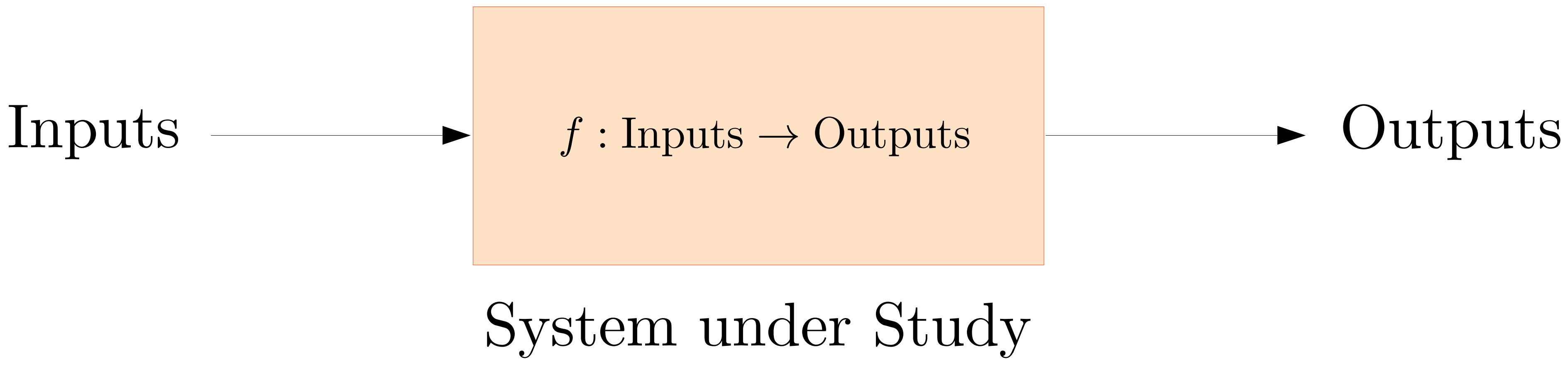
Sanjay Arora
AI Center of Excellence
Red Hat

Ulrich Drepper
AI Center of Excellence
Red Hat



Red Hat

# Why and how does something work?




**Question**: what are the rules governing the system i.e. what is f?

# The Scientific Method

Most successful approach till now:

- Observe system with different inputs and measure outputs
- <u>Guess rules or guess f</u>
- Make predictions from guessed rules
- Compare predictions to outputs
- If predictions “match” outputs, you **might** have the correct rules
- If predictions don’t match outputs, you are definitely wrong

# The Scientific Method

- A scientific theory can never be **proven** to be correct. There’s no proof the sun will rise tomorrow – just an overwhelming possibility.

- Comparison of the predictions and outputs is more subtle than it appears. It depends on the degree of accuracy required. Rules might predict outputs within 10% but not within 1% and that might be sufficient to explain the core elements of the system under study.

- Simplicity is key. The best (in terms of predictive power) scientific theories tend to be simple.

# How to guess the rules?

Many ways – see Einstein, Dirac, Feynman’s work for some examples

One way:
Collect data and scan for patterns. How Kepler found the laws for planetary motion

# What is Machine Learning?

- Collect data and find patterns to discover rules.
- Let computers scan through several guesses/hypotheses and find the best ones.
- Need to invent algorithms that can scan multiple hypotheses efficiently (in time, memory, amount of data required).

# Terminology: Model

Our guess of the rules

# Terminology: Learning

Process of scanning data to find the best rules

# Terminology: Supervised Learning

Learning when data has labels (feedback) i.e. we have both the inputs and the corresponding outputs

# Terminology: Unsupervised Learning

Learning when data doesn’t have labels (no feedback)

Very common in real-life AND much harder than supervised learning

# Terminology: Reinforcement Learning

Learning when feedback infrequent in time

Closely related to control theory

Models dynamics of an agent interacting with
a complex dynamic environment

# Some Lingo

**Data**:

Table where each column is a feature and each row is the result of one experiment.

# Some Lingo

**Features**:

Numbers/categories collected that can potentially affect the final result. Usually, the input data has one column for each feature.

# Some Lingo

**Results/Labels/Targets**:

Quantity to predict: Can be a continuous number or a discrete category (0/1/2 etc.).

# Some Lingo

**Training data**:

A randomly selected percentage (say 70%) of the rows of the data.
Rest of data ignored for model-building

# Some Lingo

**Test data**:

Part of data not included in training data (say 30%). The model built using training data is applied to test data to see how well it works on an independent set.

# Some Lingo

**Cleaning data**:

The procedure involving filling missing data, doing sanity checks on ranges and values of various features. Tends to be an ad-hoc process.

# Some Lingo

**Supervised Learning:**

Building a model where for each input, an output value is known.

Examples: Predict email is spam or non-spam based on its contents.
Predict height of a person from weight.

# Some Lingo

Supervised Learning - **Regression:**

A supervised learning model where the output value being predicted is a real number.

Examples: Predict height of a person from weight.

# Some Lingo

Supervised Learning - **Classification:**

A supervised learning model where the output value being predicted is a category.

Examples: Predict email is spam or non-spam based on its contents.

# Some Lingo

**Unsupervised Learning**:

Build a model where nothing is being predicted. Techniques used to find patterns within data.

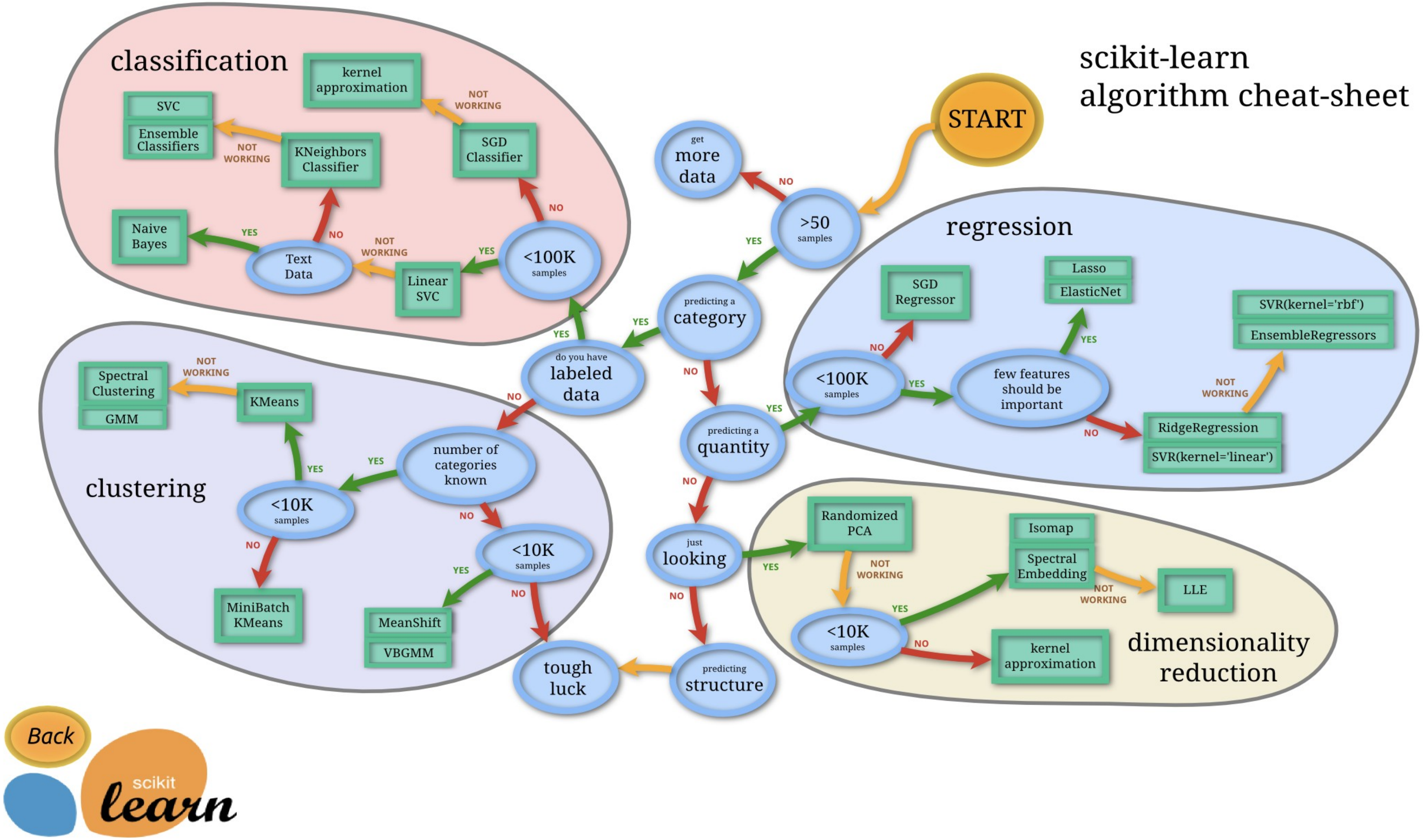
Example: clustering/grouping of data based on their pairwise distances.

# Survey of Techniques: Regression

# Problem Statement

$$\text{Given features} : x_1, x_2, \ldots, x_n$$

$$\text{predict real number y}$$




**Don’t take this too literally!**

# Linear Regression

$$y = w_0 + w_1 x_1 + \ldots + w_n x_n$$

Linear in weights, **NOT** features

For example, maybe $x_1 = x$ and $x_n = x^n, n > 1$
Definitely not linear in features x, but linear in weights $w_i$



Red Hat

# Linear Regression: Assumptions

$$\text{Data:} \qquad y = w_0 + w_1 x_1 + \ldots + w_n x_n + \underbrace{\mathcal{N}(0, \sigma^2)}_{\text{Gaussian noise}}$$

$$\textbf{Linear relationship between} \;\; \mathrm{y} \;\; \text{and} \;\; \mathrm{x}_m, \forall m$$
$$\text{where all other } x_n \text{ are fixed}$$

$\mathcal{N}(0, \sigma^2)$ : **Gaussian** noise with **0 mean** and **constant variance**

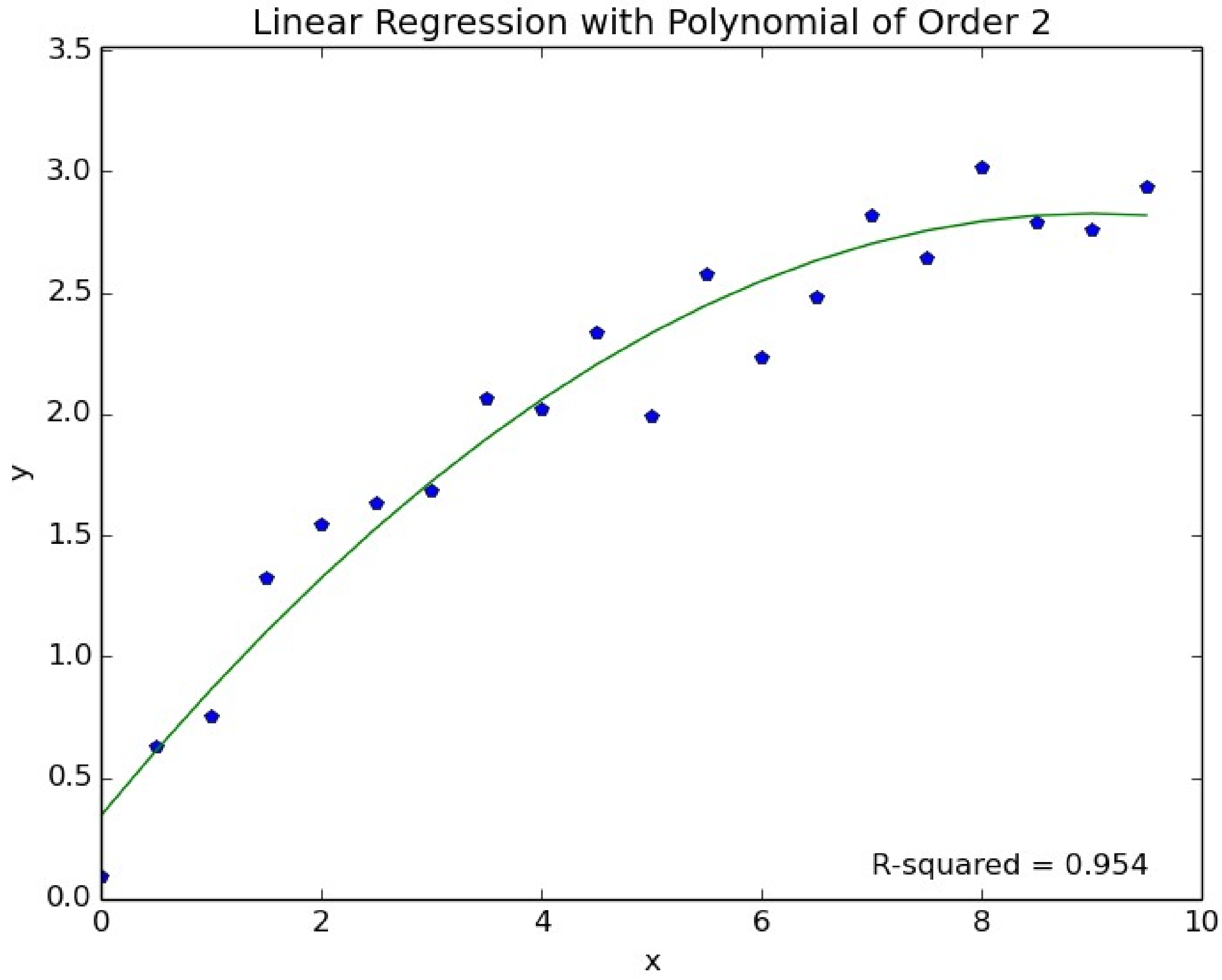
Might consist of many noise terms

They should be **independent**, **Gaussian** with **constant variance**



Red Hat

# Example




$$y = w_0 + w_1 x + w_2 x^2 = 0.344 + 0.551x - 0.031x^2$$

# Linear Regression: Preprocessing

Generally, features $x_m$ can have different ranges they take values in

Good to normalized each feature to have mean $= 0$ and standard deviation $= 1$

Mean across all examples

$$x_m \rightarrow \frac{x_m - \mu(x_m)}{\sigma(x_m)}$$

Standard dev. across all examples

Mean $= 0$ helps with convergence of algorithms too

If features in similar ranges, can compare weights $w_m$ to rank features according to influence on output

# Linear Regression: Python Code

```
from sklearn import linear_model

model = linear_model.LinearRegression() #create instance of linear regression model
model.fit(x_values, y_values) #train model

#DONE!!!

model.predict(new_x_values) #predict result on new inputs

model.coef_ #see weights

model.score(x_values, y_values) #we'll see this soon. Compute R2
```

# Linear Regression: Python Code

```
model = linear_model.Ridge(alpha = 0.5) #"Ridge" regression with one free parameter

model = linear_model.Lasso(alpha = 0.5) #"Lasso" regression with one free parameter
```

Use these instead of linear model used previously

Prevent overfitting (coming up soon)

Also, it's very enlightening to look at source code if interested

# Linear Regression: Performance

$$\text{Residual} \equiv \Sigma_{i=1}^{N}(y_i - f(x_i))^2$$

Result of
example i

Model prediction for
example i

$$\text{Variance} \equiv \Sigma_{i=1}^{N}(y_i - \bar{y})^2$$

$$\text{Unexplained Variance Ratio} = \frac{\text{Residual}}{\text{Variance}}$$

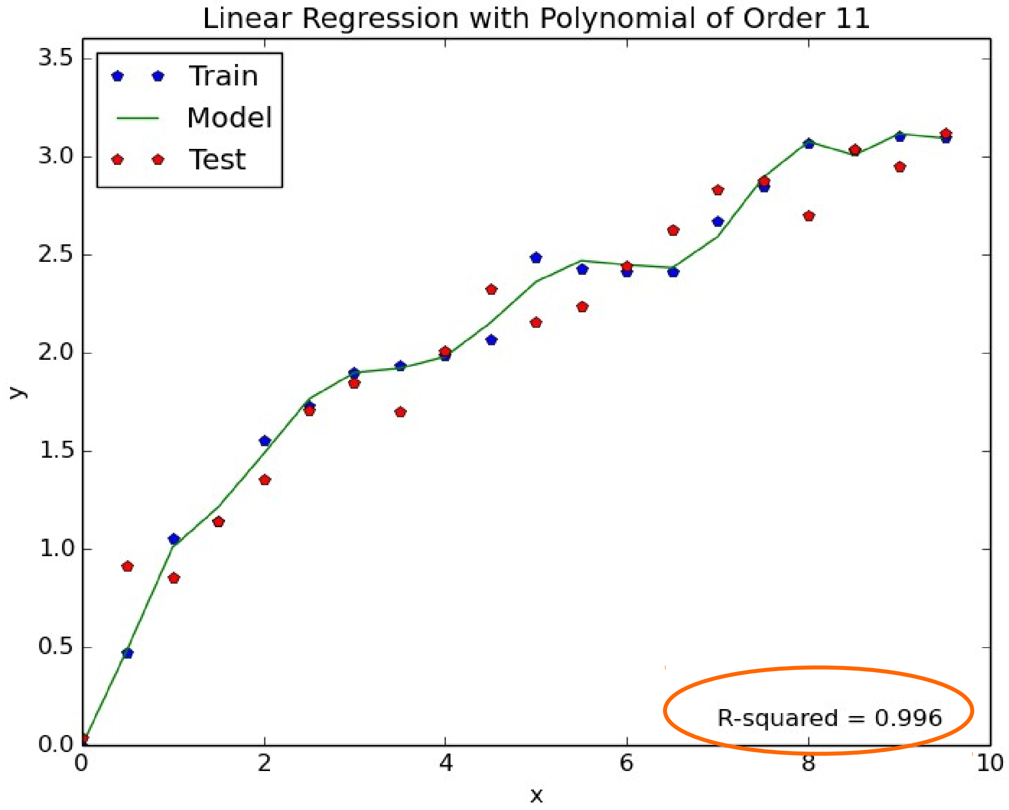
$$\boxed{R^2 \equiv 1 - \frac{\text{Residual}}{\text{Variance}}}$$

Close to 1 = Good(*)

# Linear Regression: Performance

$$R^2 \equiv 1 - \frac{\text{Residual}}{\text{Variance}}$$

**Close to 1 but might be overfitting**




Compare $R^2$ between train and test sets!!!!!



Red Hat

# Linear Regression: Performance

Look at distribution of residuals: $y_i - f(x_i)$

Should be Gaussian with constant variance

# K-Nearest Neighbors

Find K ”closest” examples

Average their results

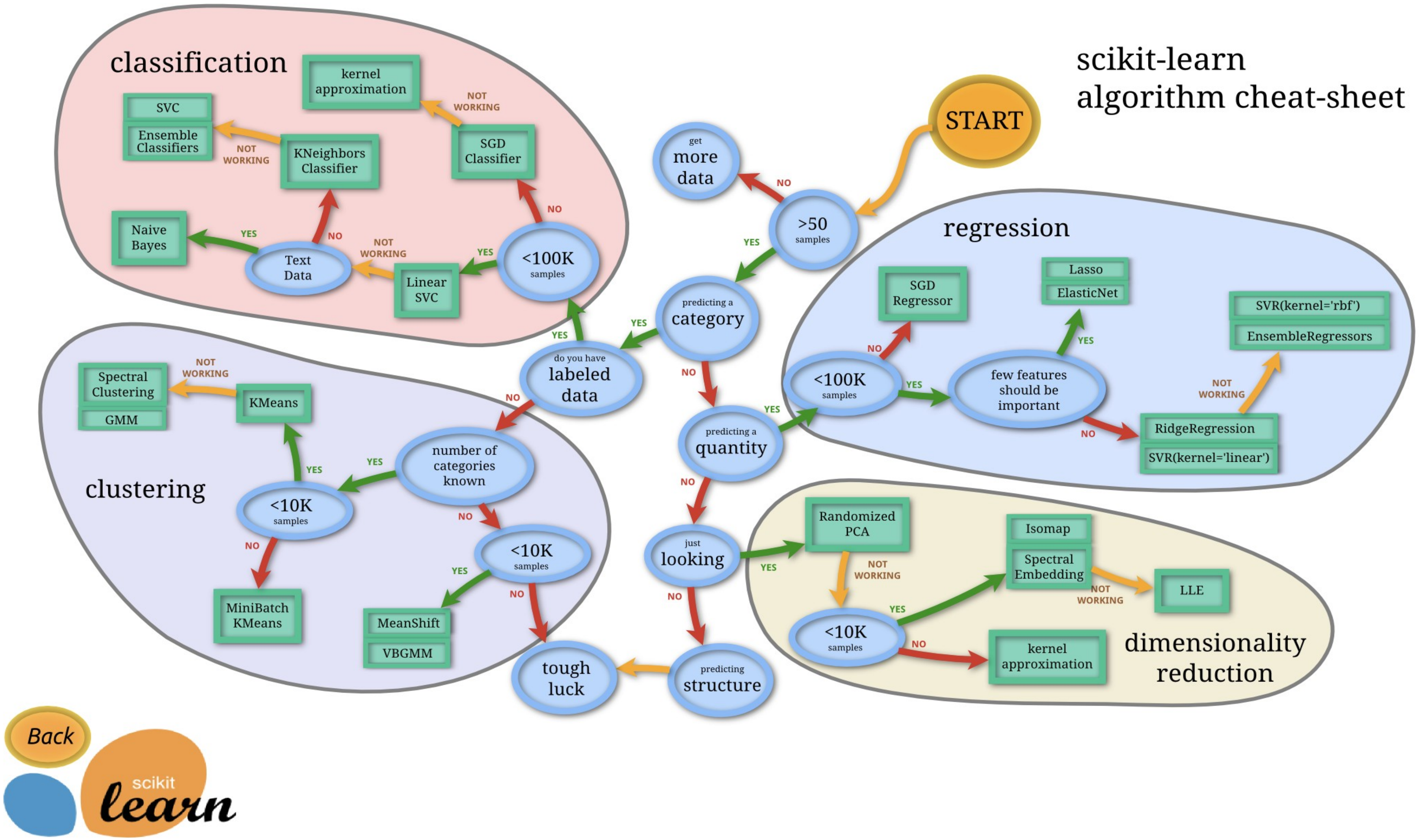
Really! That’s it

# Survey of Techniques: Classification

# Problem Statement

$$\text{Given features} : x_1, x_2, \ldots, x_n$$

$$\text{predict class or label, y} = 0 \text{ or } 1$$




Logistic “Regression”    Decision Trees    K-Nearest Neighbors    Neural Networks

# Logistic "Regression"

How about:

$$(x_1, x_2, \ldots, x_n)$$

$$\downarrow$$

$$w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

Linear combination:

$$\downarrow ?$$

$$[0, 1] \longrightarrow$$ Interpret as probability of label = 1

# Logistic/Sigmoid Function

Want to map real number to [0,1]

One possible solution: $f(x) = \frac{1}{1+e^{-x}}$

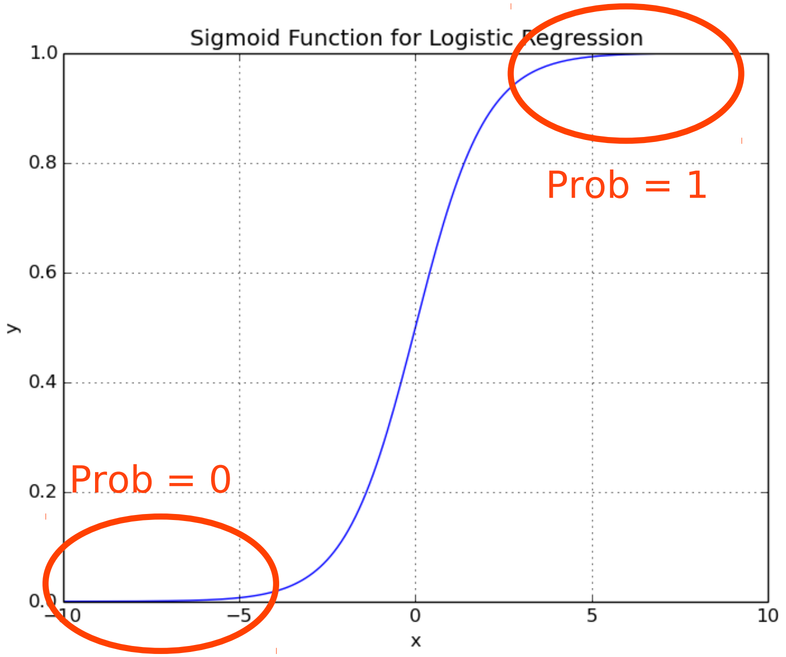
$$x \to \infty, f(x) \to \frac{1}{1+0} = 1$$

$$x \to -\infty, f(x) \to \frac{1}{1+\infty} = 0$$

$$x = 0, f(x) = 0.5$$

# Logistic/Sigmoid Function

$$x \to -\infty, f(x) \to \frac{1}{1+\infty} = 0$$




$$x \to \infty, f(x) \to \frac{1}{1+0} = 1$$

Probability

Want to map real number to [0,1]

$$\text{One possible solution: } f(x) = \frac{1}{1+e^{-x}}$$

$x = 0, f(x) = 0.5$ → "Boundary" or "Threshold"

$$(x_1, x_2, \ldots, x_n)$$

$$w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

Linear combination:

$$f(w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n) = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n)}}$$

$< 0.5$

$= 0.5$

$> 0.5$

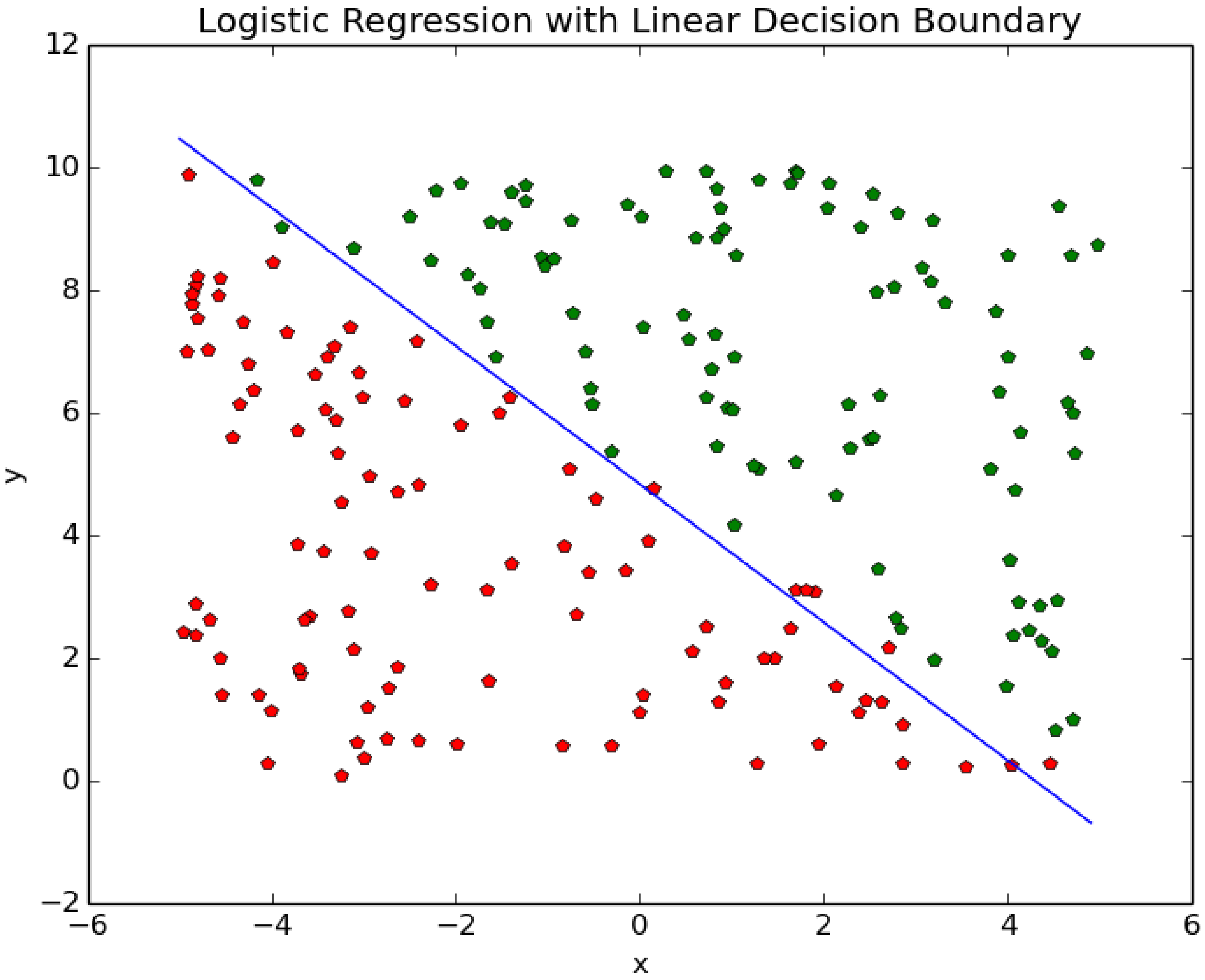
Label 0

Boundary separating classes:

Label 1

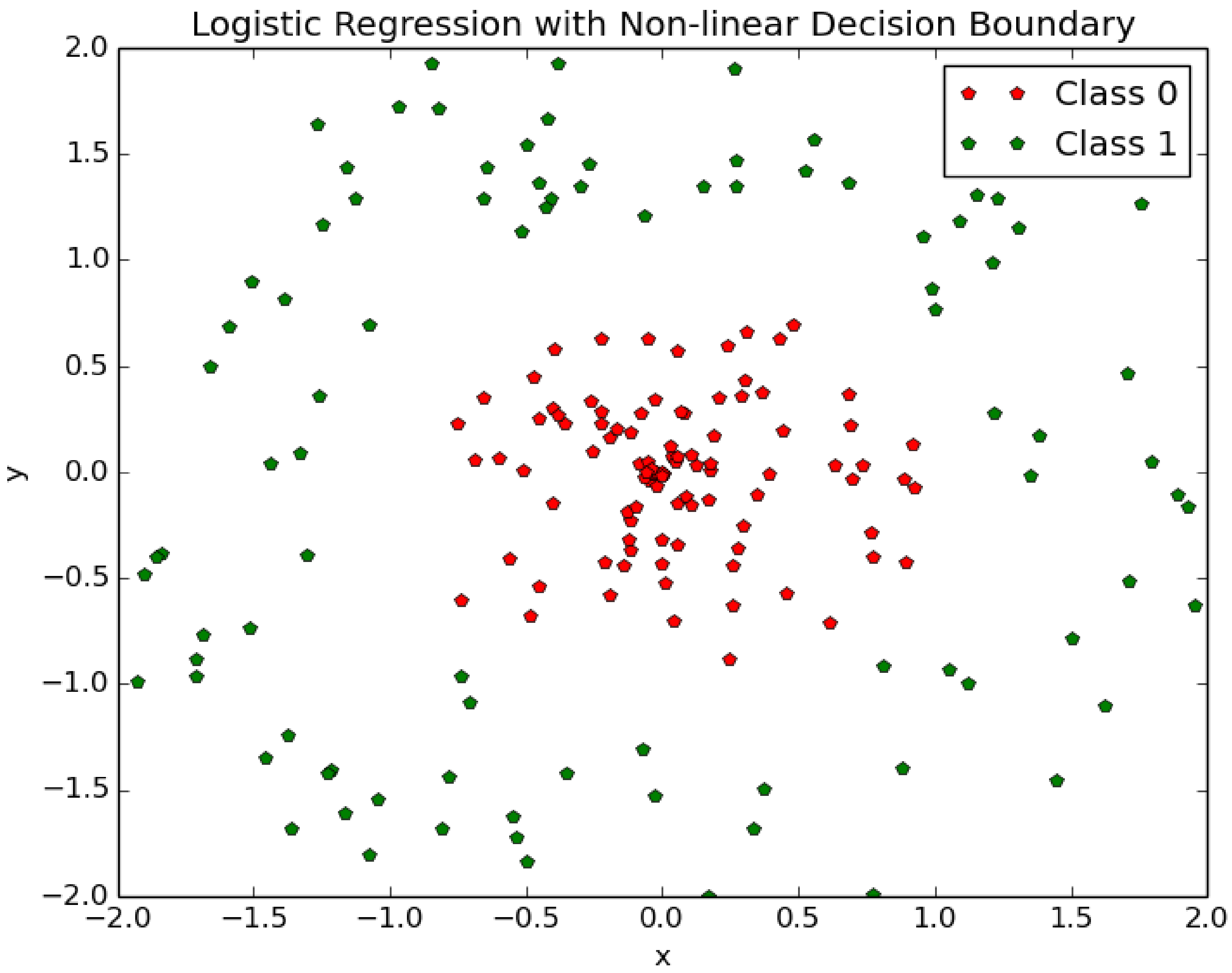
$$w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n = 0$$

# Example




$$\text{Decision Boundary}: -4.86 + 1.05x + 1.00y = 0$$

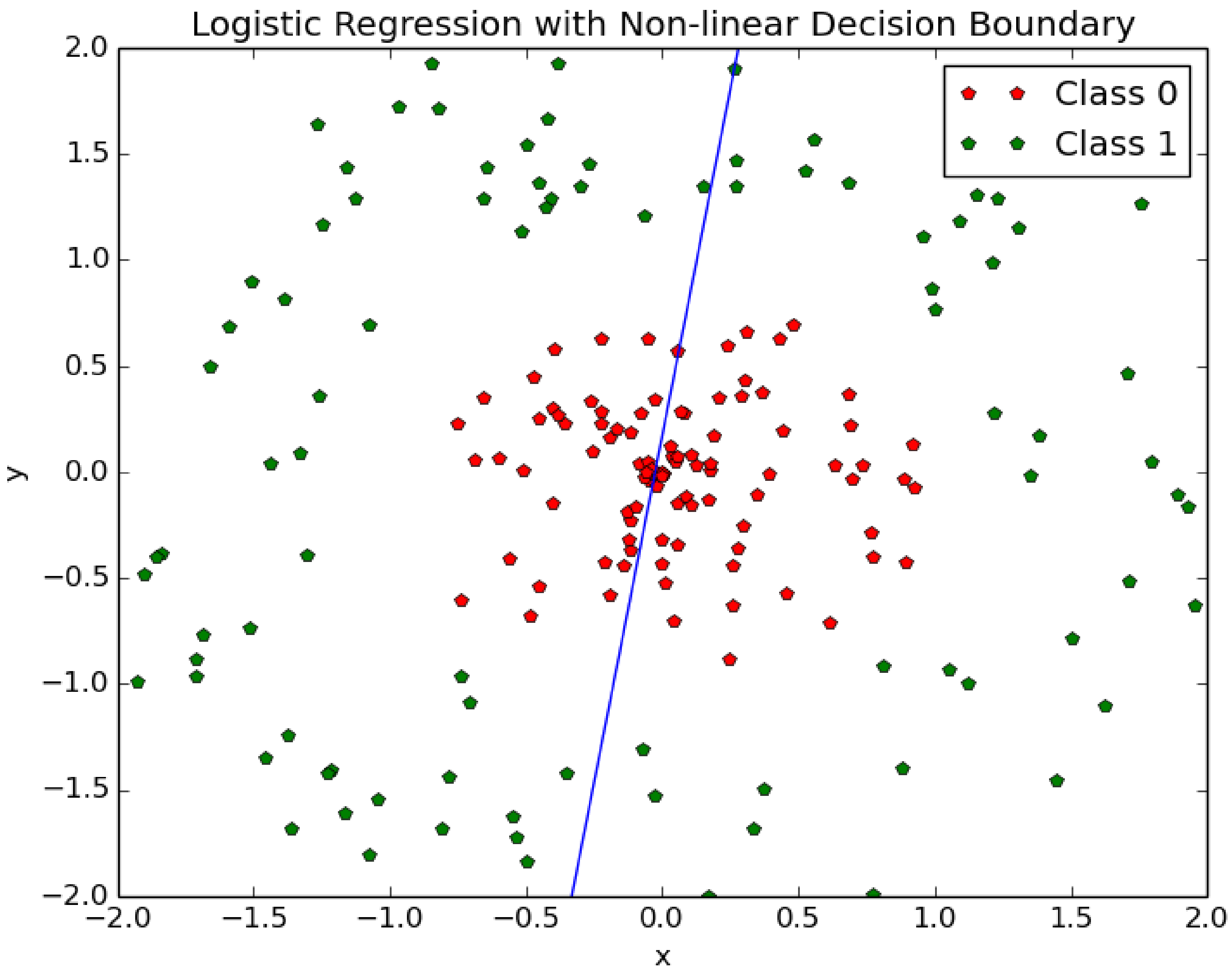
# Example




Features are: $(x, y)$

**Accuracy = 64%**

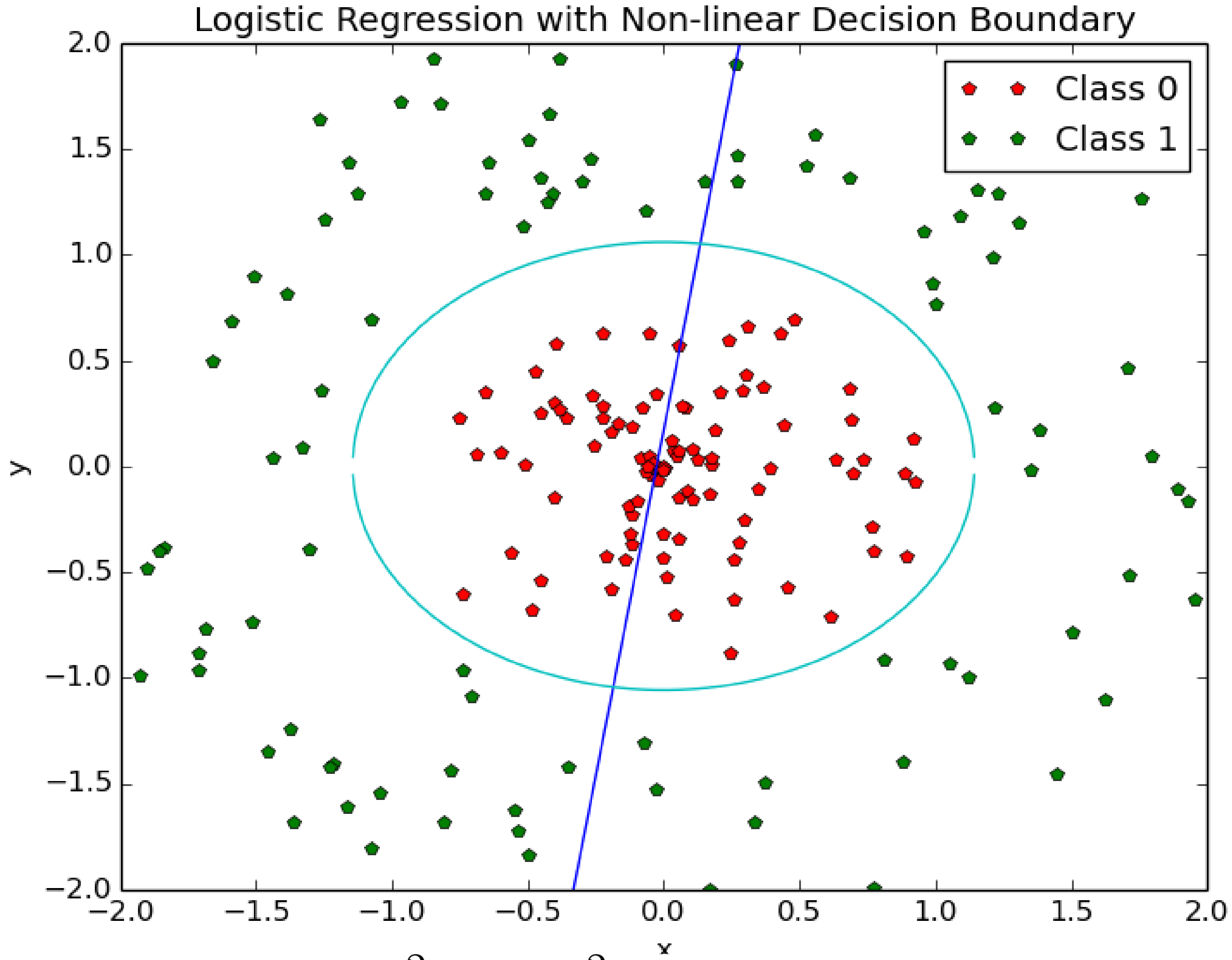
# Example




Bad!!!

$$\text{Decision Boundary: } w_0 + w_1 x + w_2 y = 0 \implies \text{Straight Line}$$

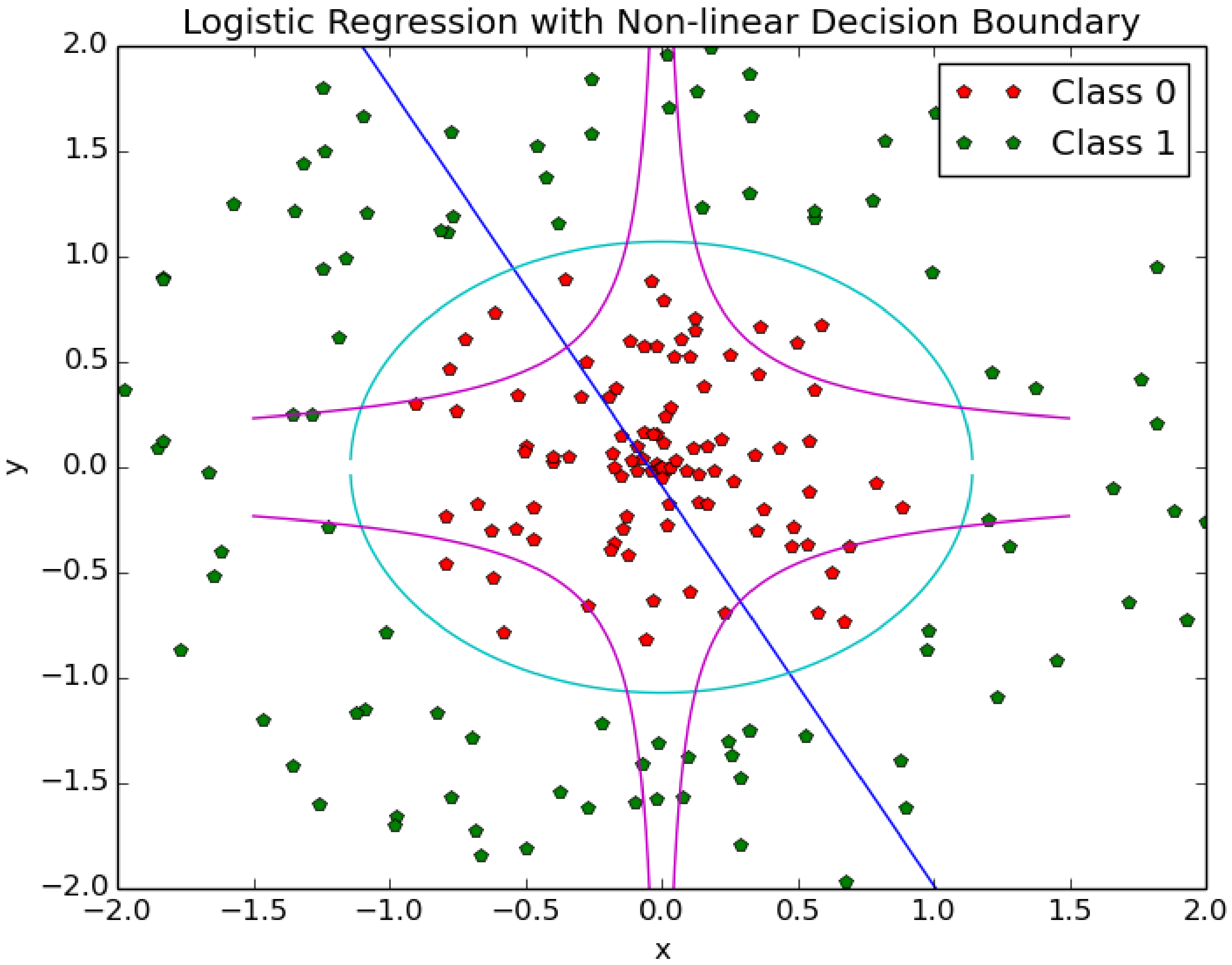
# Features are: $(x^2, y^2)$ NOT $(x, y)$




**Accuracy = 100%**

Decision Boundary: $w_0 + w_1 x^2 + w_2 y^2 = 0 \implies$ Ellipse (in this case at least)

Features are: $(\log|x|, \log|y|)$ NOT $(x, y)$




**Accuracy = 86%**

Decision Boundary: $a\log|x| + b\log|y| + c = 0 \implies y = \pm\frac{e^{-\frac{c}{b}}}{|x|^{\frac{a}{b}}}$



Red Hat

# Logistic Regression: Python Code

```
from sklearn import linear_model

model = linear_model.LogisticRegression()
model.fit(features, labels) #train the model

model.predict(test_features) #make predictions (0/1)
model.predict_proba(test_features) #predict probabilities
```

# Metrics for evaluation: Classification

These metrics apply to **any** classification model's output

| $C_i$ | $pred = 0$ | $pred = 1$ |
|---|---|---|
| $label = 0$ | $0 \rightarrow 0$ | $0 \rightarrow 1$ |
| $label = 1$ | $1 \rightarrow 0$ | $1 \rightarrow 1$ |

$$i \rightarrow j = \text{label i} \rightarrow \text{pred j}$$

| $C_i$ | $pred = 0$ | $pred = 1$ |
|---|---|---|
| $label = 0$ | True Negatives | False Positives |
| $label = 1$ | False Negatives | True Positives |

# Metrics for evaluation: Classification

Good

Bad

| $C_i$ | $pred = 0$ | $pred = 1$ |
|---|---|---|
| $label = 0$ | $0 \to 0$ | $0 \to 1$ |
| $label = 1$ | $1 \to 0$ | $1 \to 1$ |

$$i \to j = \text{label i} \to \text{pred j}$$

| $C_i$ | $pred = 0$ | $pred = 1$ |
|---|---|---|
| $label = 0$ | True Negatives | False Positives |
| $label = 1$ | False Negatives | True Positives |

# Metrics for evaluation: Classification

$$\text{Accuracy} = \frac{\#\ \text{correct}}{\#\ \text{total}} = \frac{(0 \to 0) + (1 \to 1)}{(0 \to 0) + (1 \to 1) + (0 \to 1) + (1 \to 0)}$$

What if 98% of examples are labeled 0 and 2% labeled 1?

Build "dumb" model that predicts 0 for every examples

Accuracy = 98% (Get everything labeled 0 correct)

Bad measure if population of labeled classes very uneven

# Metrics for evaluation: Classification

Track metrics:

$$\text{Accuracy} = \frac{(0 \to 0) + (1 \to 1)}{(0 \to 0) + (1 \to 1) + (0 \to 1) + (1 \to 0)}$$

**Ideal Case: 100%**

$$\text{Specificity} = \frac{(0 \to 0)}{(0 \to 0) + (0 \to 1)}$$

**Ideal Case: 100%**

$$\text{Sensitivity/Recall} = \frac{(1 \to 1)}{(1 \to 1) + (1 \to 0)}$$

**Ideal Case: 100%**

$$\text{Precision} = \frac{(1 \to 1)}{(0 \to 1) + (1 \to 1)}$$

**Ideal Case: 100%**

$$\text{type I error} = \frac{(0 \to 1)}{(0 \to 1) + (0 \to 0)}$$

**Ideal Case: 0%**

$$\text{type II error} = \frac{(1 \to 0)}{(1 \to 0) + (1 \to 1)}$$

**Ideal Case: 0%**

# Precision and Recall

$$\text{Sensitivity/Recall} = \frac{(1 \to 1)}{(1 \to 1) + (1 \to 0)}$$

% of things that are labeled 1 were predicted to be 1

e.g. *in a labeled dataset, if there are 200 fraudulent credit card transactions, what % are predicted to be fraudulent by model*

$$\text{Precision} = \frac{(1 \to 1)}{(0 \to 1) + (1 \to 1)}$$

Out of things that the model predicts should be 1, what % are actually 1s

e.g. *if a model predicts 100 credit card transactions are fraud, what % are actually frauds – relevant to the end-user*

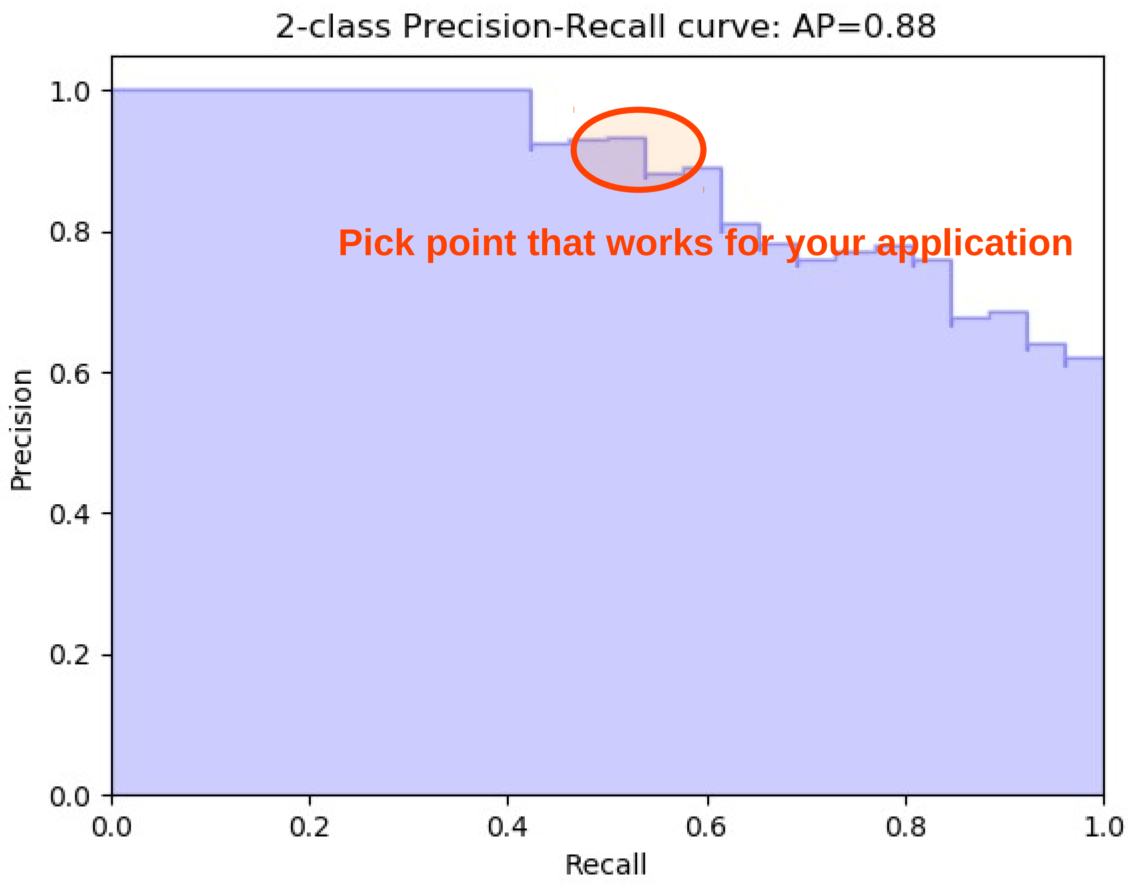
# One more note

Often models don’t predict 0 or 1

Prediction is a **probability** of being 1

A cutoff has to be chosen such that probability > CUTOFF means the prediction and probability <= CUTOFF means the prediction is 0

Precision and recall can be computed for a range of CUTOFFs to give a precision-recall curve




scikit-learn.org

# Decision Tree

Series of if-else statements on features that distinguish between two classes

# Decision Tree

Series of if-else statements on features
that distinguish between two classes

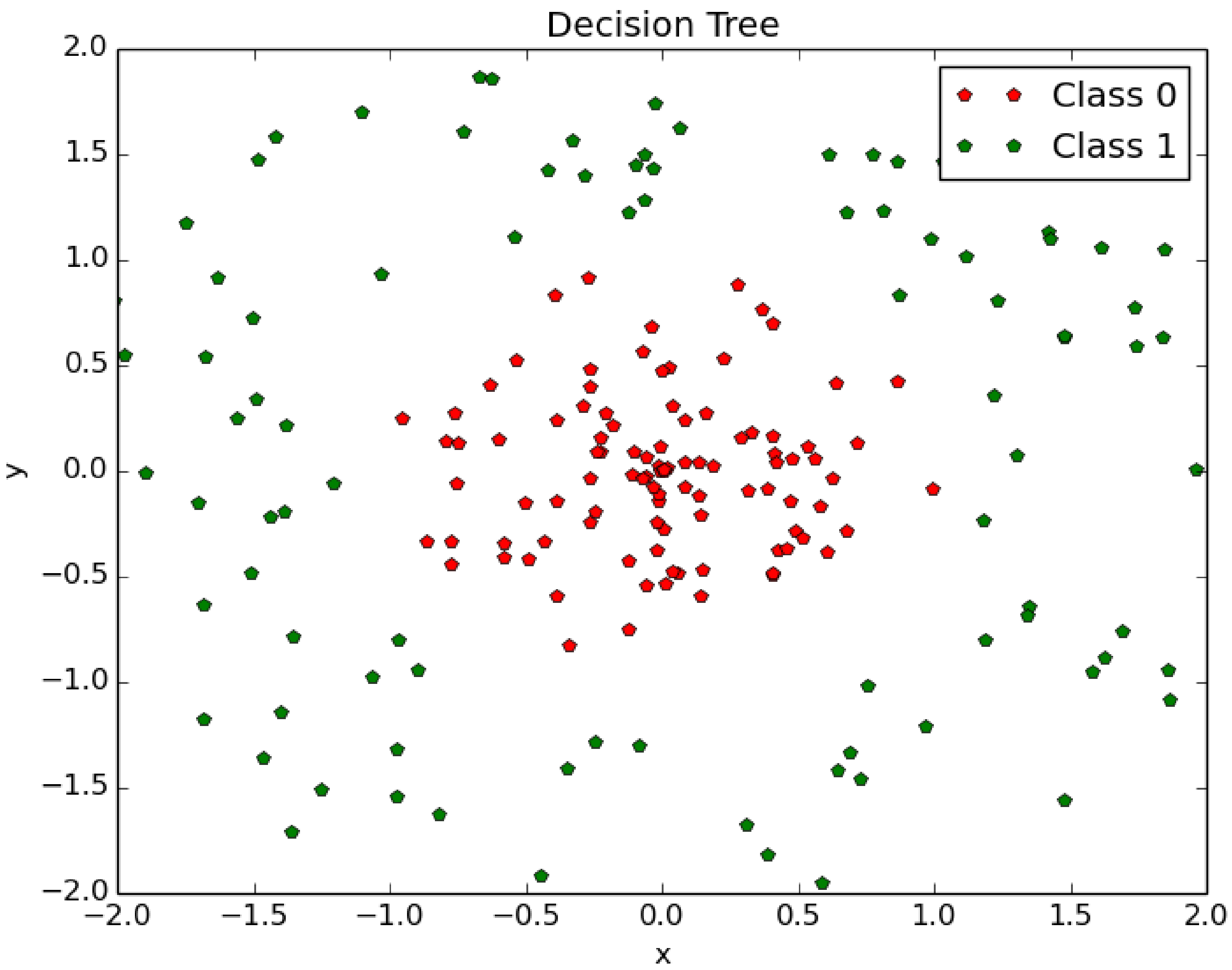
Some we code up all the time!

# Decision Tree

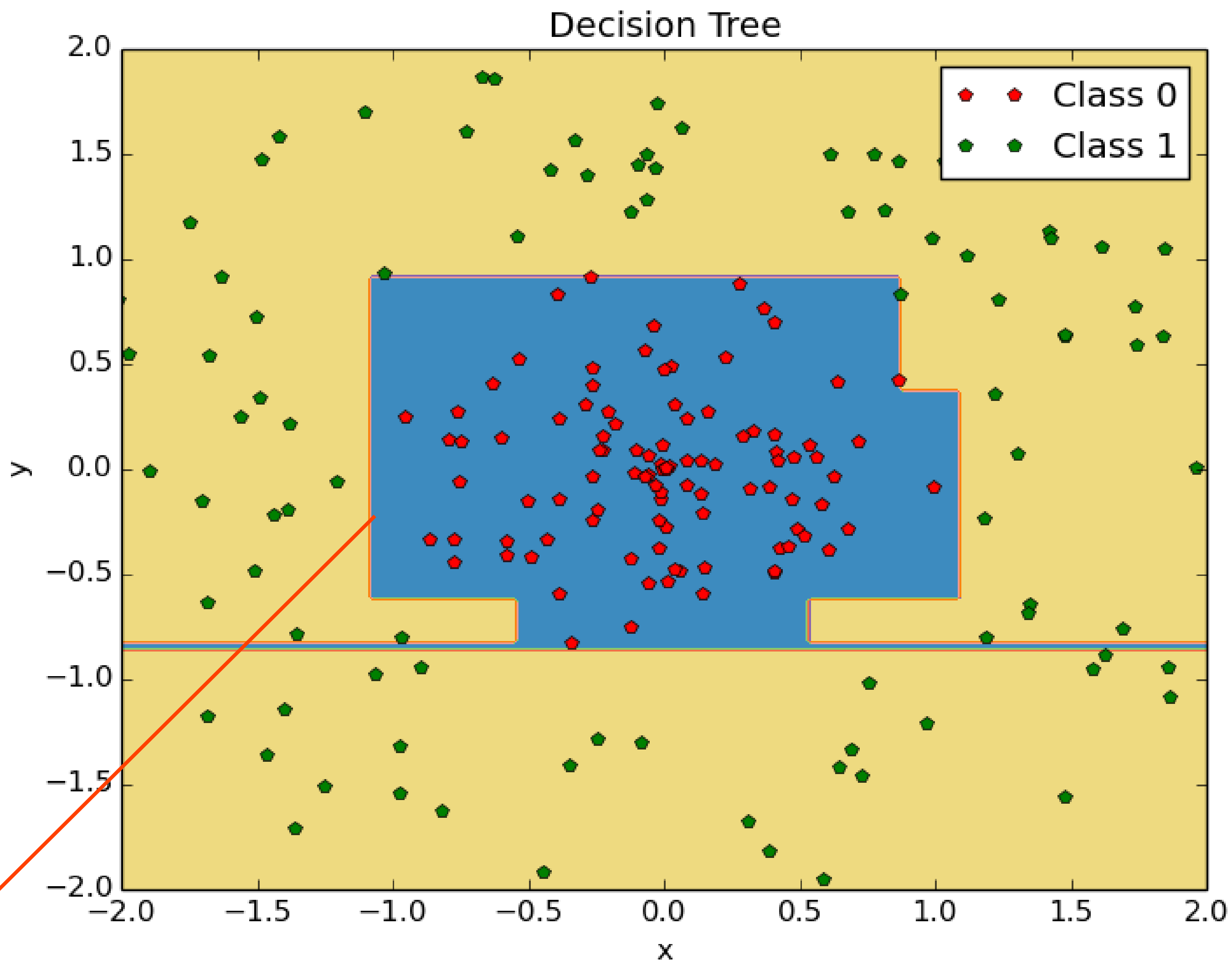
Series of if-else statements on features
that distinguish between two classes

Something we code up all the time!

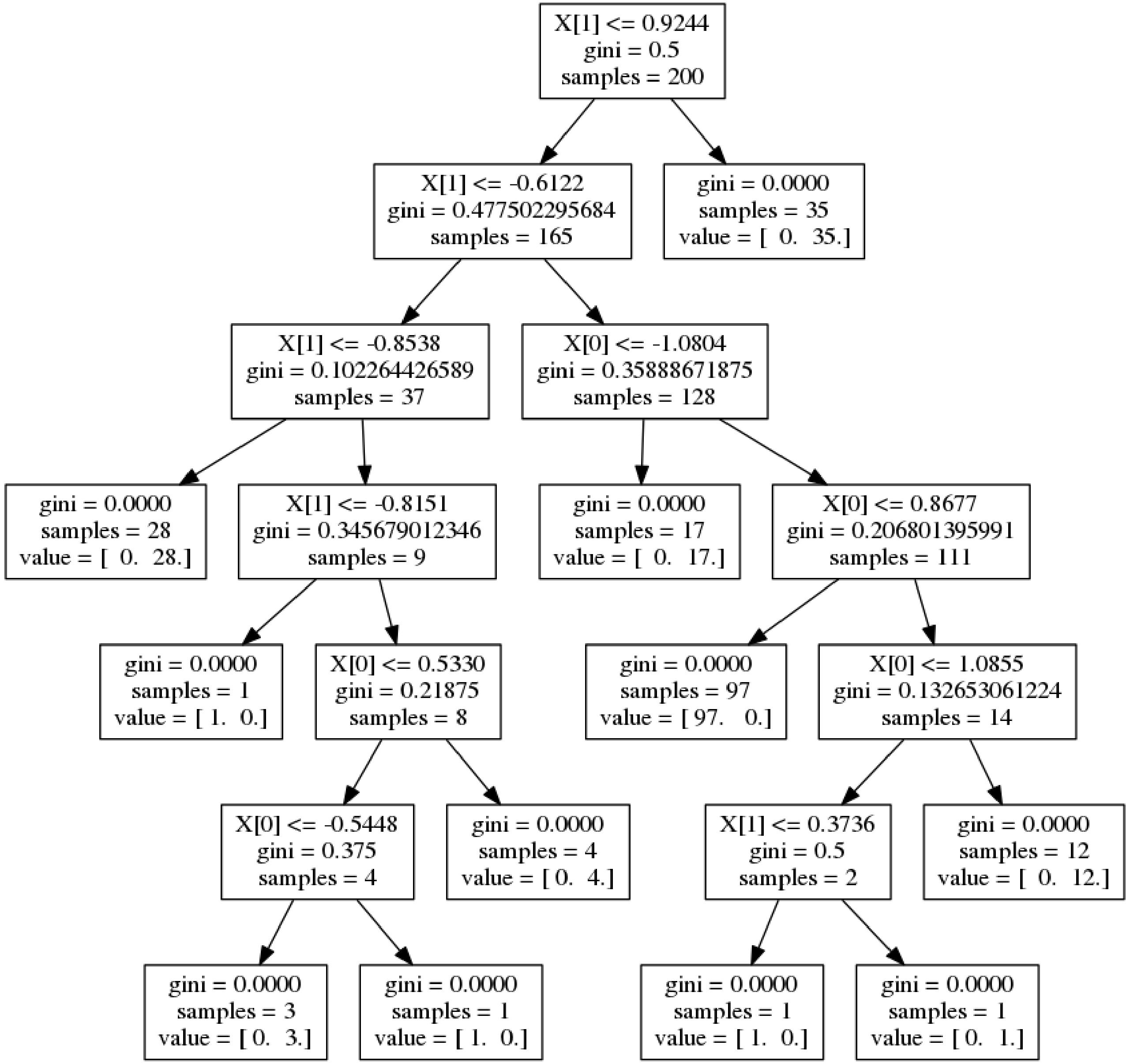
But now the expressions are learned!

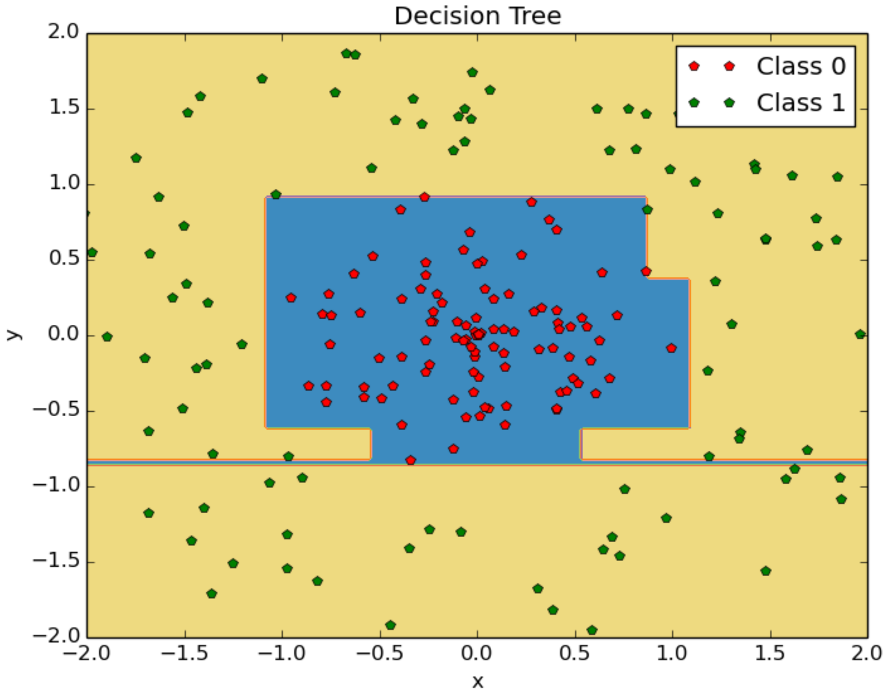


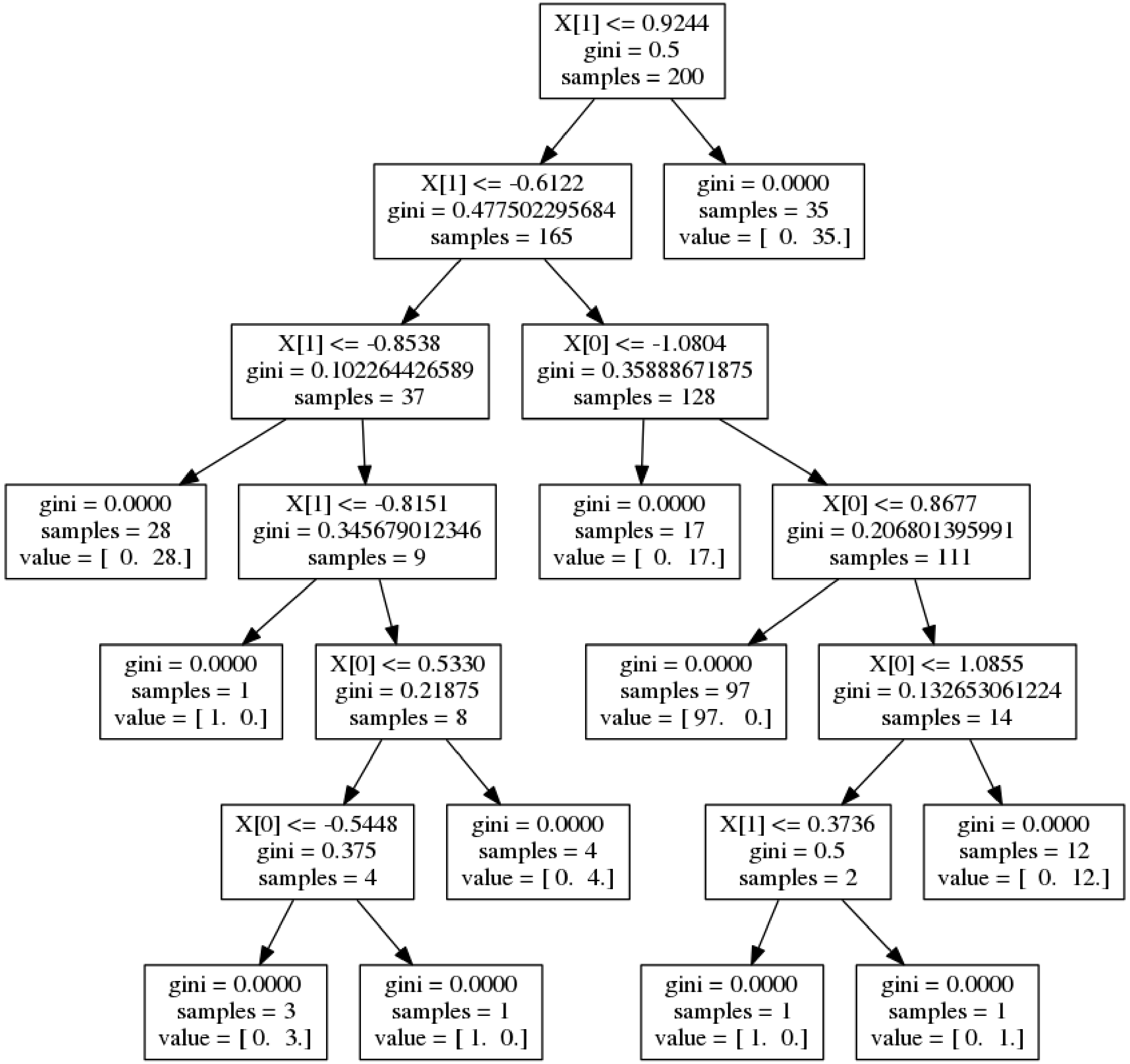

Is it possible to separate red and green dots with a sequence of if-else statements on x and y?



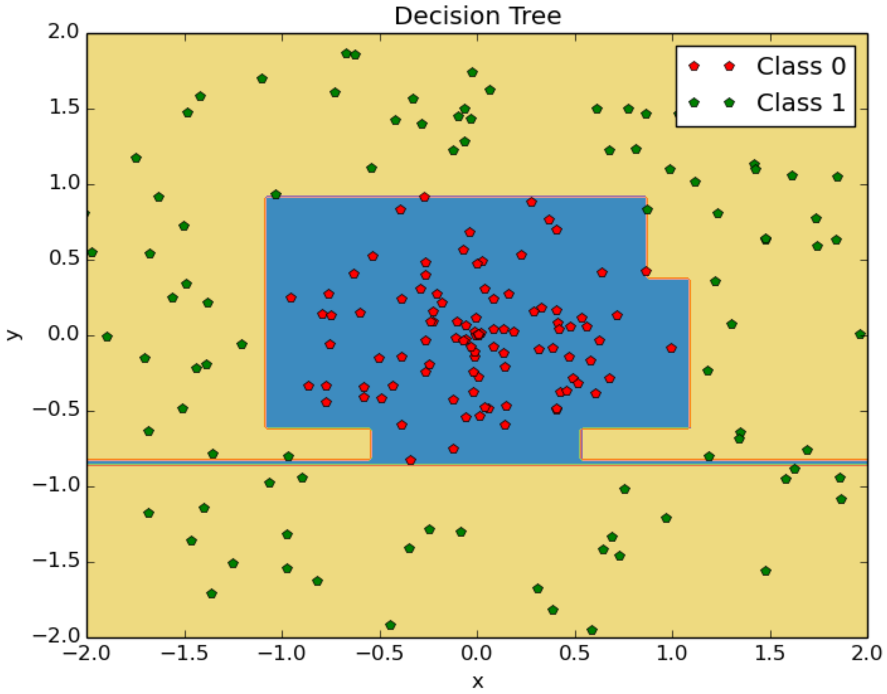

Any if-else statement on x or y ⟺ Horizontal or vertical line

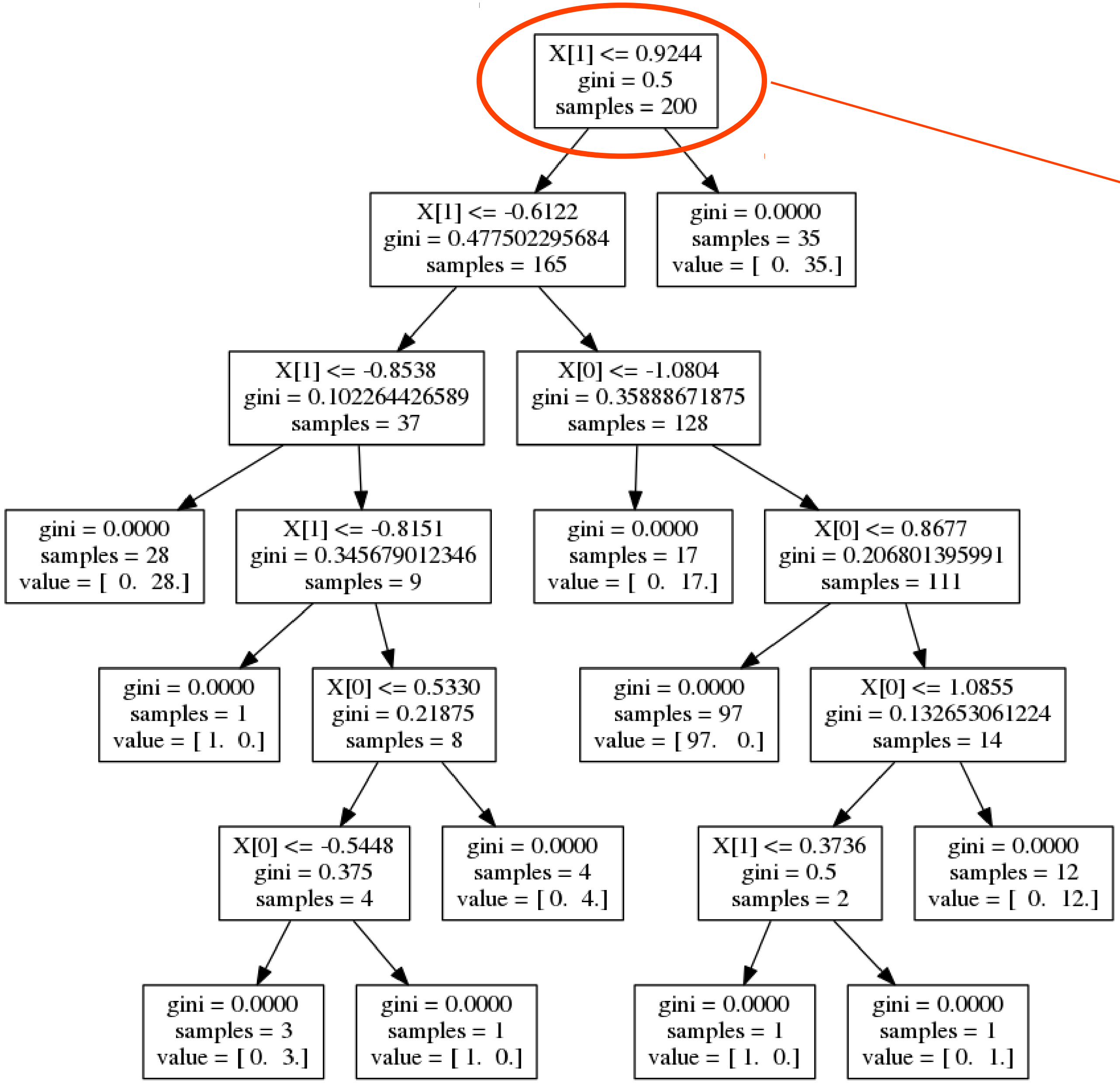

X[1] <= 0.9244 gini = 0.5 samples = 200
X[1] <= -0.6122 gini = 0.477502295684 samples = 165
gini = 0.0000 samples = 35 value = [ 0. 35.]
X[1] <= -0.8538 gini = 0.102264426589 samples = 37
X[0] <= -1.0804 gini = 0.35888671875 samples = 128
gini = 0.0000 samples = 28 value = [ 0. 28.]
X[1] <= -0.8151 gini = 0.345679012346 samples = 9
gini = 0.0000 samples = 17 value = [ 0. 17.]
X[0] <= 0.8677 gini = 0.206801395991 samples = 111
gini = 0.0000 samples = 1 value = [ 1. 0.]
X[0] <= 0.5330 gini = 0.21875 samples = 8
gini = 0.0000 samples = 97 value = [ 97. 0.]
X[0] <= 1.0855 gini = 0.132653061224 samples = 14
X[0] <= -0.5448 gini = 0.375 samples = 4
gini = 0.0000 samples = 4 value = [ 0. 4.]
X[1] <= 0.3736 gini = 0.5 samples = 2
gini = 0.0000 samples = 12 value = [ 0. 12.]
gini = 0.0000 samples = 3 value = [ 0. 3.]
gini = 0.0000 samples = 1 value = [ 1. 0.]
gini = 0.0000 samples = 1 value = [ 1. 0.]
gini = 0.0000 samples = 1 value = [ 0. 1.]


Decision Tree
Class 0
Class 1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0 −1.5 −1.0 −0.5 0.0 0.5 1.0 1.5 2.0
x
y

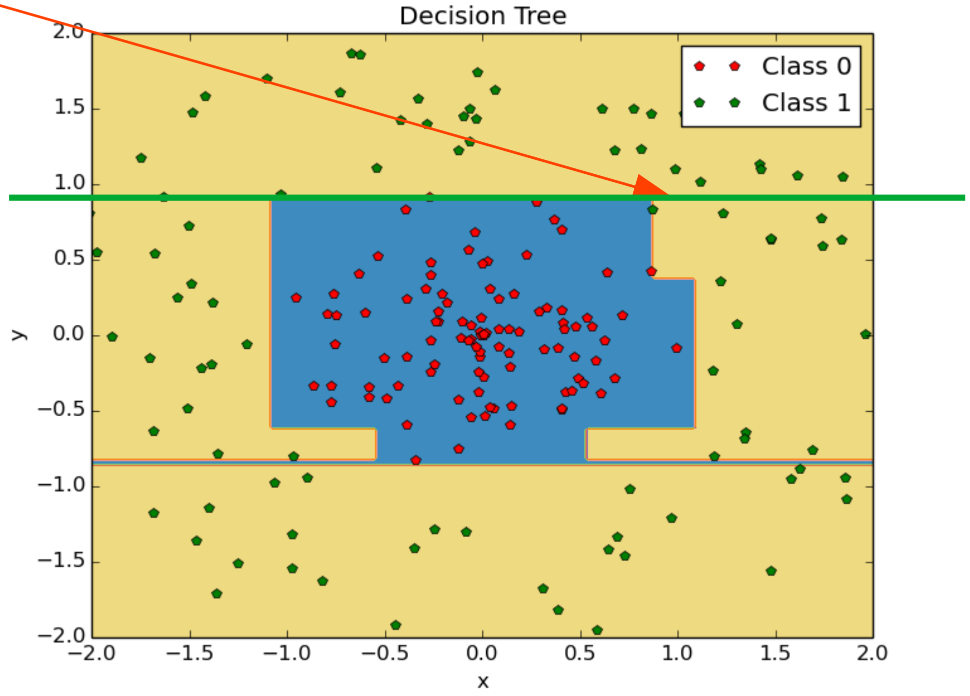

X[1] <= 0.9244
gini = 0.5
samples = 200
X[1] <= -0.6122
gini = 0.477502295684
samples = 165
gini = 0.0000
samples = 35
value = [ 0. 35.]
X[1] <= -0.8538
gini = 0.102264426589
samples = 37
X[0] <= -1.0804
gini = 0.35888671875
samples = 128
gini = 0.0000
samples = 28
value = [ 0. 28.]
X[1] <= -0.8151
gini = 0.345679012346
samples = 9
gini = 0.0000
samples = 17
value = [ 0. 17.]
X[0] <= 0.8677
gini = 0.206801395991
samples = 111
gini = 0.0000
samples = 1
value = [ 1. 0.]
X[0] <= 0.5330
gini = 0.21875
samples = 8
gini = 0.0000
samples = 97
value = [ 97. 0.]
X[0] <= 1.0855
gini = 0.132653061224
samples = 14
X[0] <= -0.5448
gini = 0.375
samples = 4
gini = 0.0000
samples = 4
value = [ 0. 4.]
X[1] <= 0.3736
gini = 0.5
samples = 2
gini = 0.0000
samples = 12
value = [ 0. 12.]
gini = 0.0000
samples = 3
value = [ 0. 3.]
gini = 0.0000
samples = 1
value = [ 1. 0.]
gini = 0.0000
samples = 1
value = [ 1. 0.]
gini = 0.0000
samples = 1
value = [ 0. 1.]

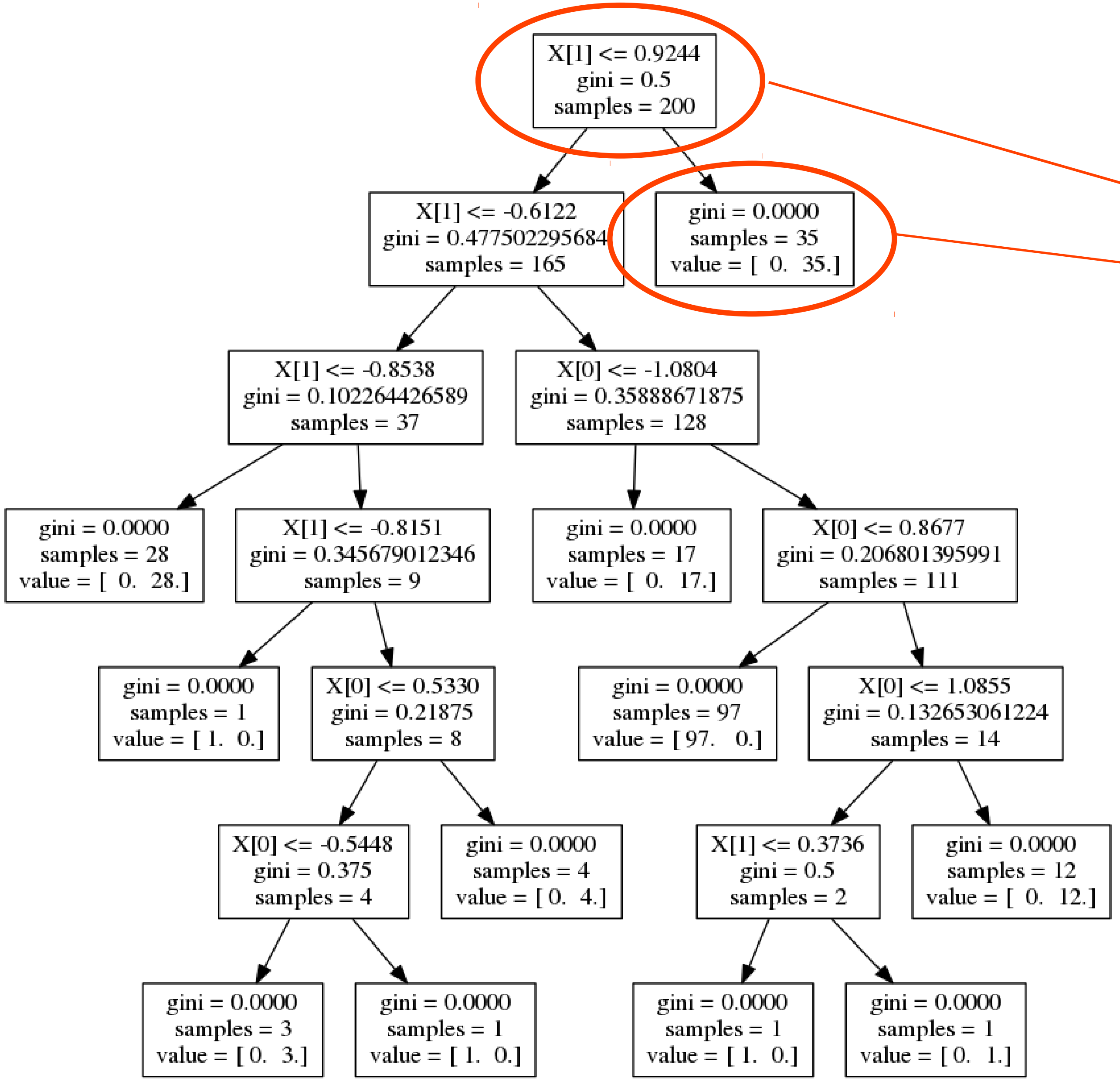
## X[0] = x
## X[1] = y


Decision Tree
Class 0
Class 1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
y
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x

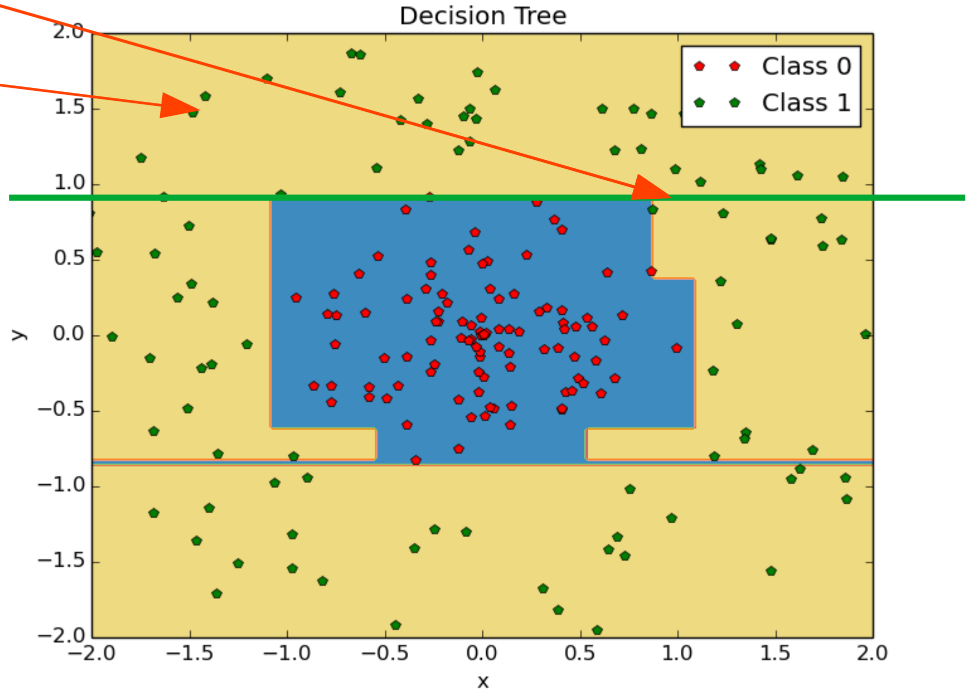

X[1] <= 0.9244
gini = 0.5
samples = 200
X[1] <= -0.6122
gini = 0.477502295684
samples = 165
gini = 0.0000
samples = 35
value = [ 0. 35.]
X[1] <= -0.8538
gini = 0.102264426589
samples = 37
X[0] <= -1.0804
gini = 0.35888671875
samples = 128
gini = 0.0000
samples = 28
value = [ 0. 28.]
X[1] <= -0.8151
gini = 0.345679012346
samples = 9
gini = 0.0000
samples = 17
value = [ 0. 17.]
X[0] <= 0.8677
gini = 0.206801395991
samples = 111
gini = 0.0000
samples = 1
value = [ 1. 0.]
X[0] <= 0.5330
gini = 0.21875
samples = 8
gini = 0.0000
samples = 97
value = [ 97. 0.]
X[0] <= 1.0855
gini = 0.132653061224
samples = 14
X[0] <= -0.5448
gini = 0.375
samples = 4
gini = 0.0000
samples = 4
value = [ 0. 4.]
X[1] <= 0.3736
gini = 0.5
samples = 2
gini = 0.0000
samples = 12
value = [ 0. 12.]
gini = 0.0000
samples = 3
value = [ 0. 3.]
gini = 0.0000
samples = 1
value = [ 1. 0.]
gini = 0.0000
samples = 1
value = [ 1. 0.]
gini = 0.0000
samples = 1
value = [ 0. 1.]

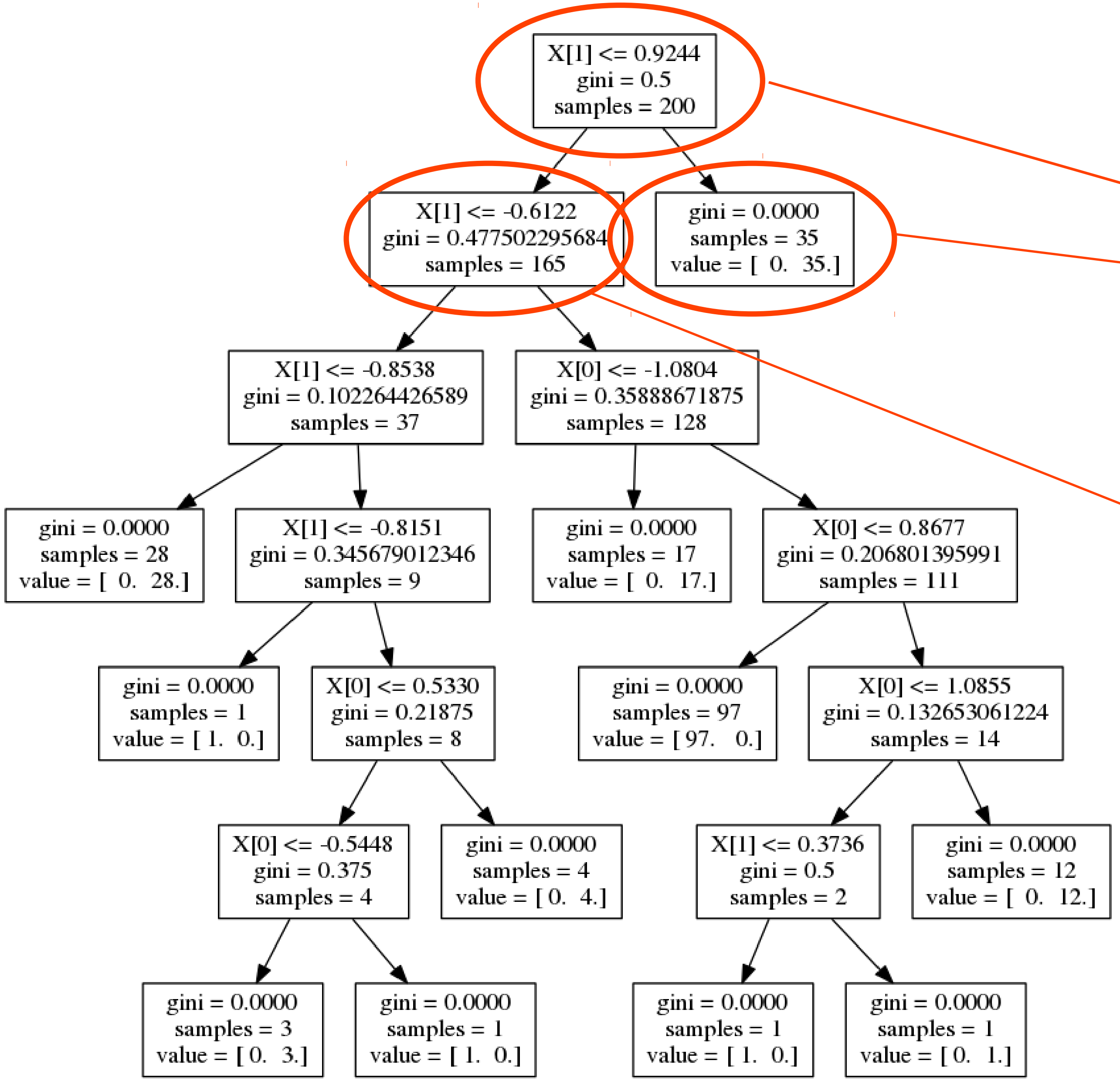
**X[0] = x**
**X[1] = y**


Decision Tree
Class 0
Class 1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
y
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x

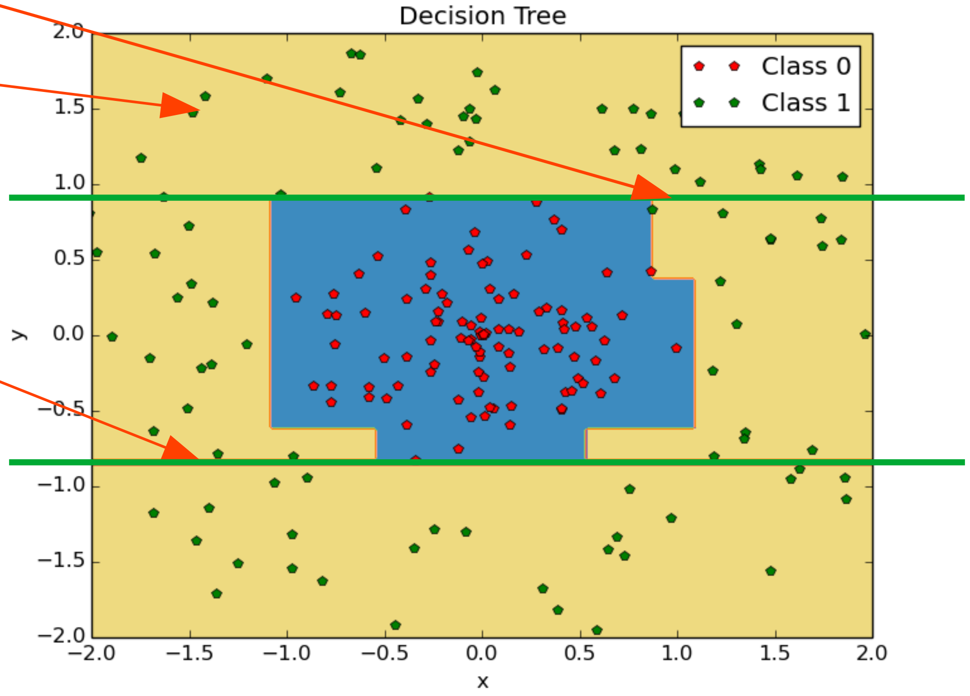

X[1] <= 0.9244
gini = 0.5
samples = 200
X[1] <= -0.6122
gini = 0.477502295684
samples = 165
gini = 0.0000
samples = 35
value = [ 0. 35.]
X[1] <= -0.8538
gini = 0.102264426589
samples = 37
X[0] <= -1.0804
gini = 0.35888671875
samples = 128
gini = 0.0000
samples = 28
value = [ 0. 28.]
X[1] <= -0.8151
gini = 0.345679012346
samples = 9
gini = 0.0000
samples = 17
value = [ 0. 17.]
X[0] <= 0.8677
gini = 0.206801395991
samples = 111
gini = 0.0000
samples = 1
value = [ 1. 0.]
X[0] <= 0.5330
gini = 0.21875
samples = 8
gini = 0.0000
samples = 97
value = [ 97. 0.]
X[0] <= 1.0855
gini = 0.132653061224
samples = 14
X[0] <= -0.5448
gini = 0.375
samples = 4
gini = 0.0000
samples = 4
value = [ 0. 4.]
X[1] <= 0.3736
gini = 0.5
samples = 2
gini = 0.0000
samples = 12
value = [ 0. 12.]
gini = 0.0000
samples = 3
value = [ 0. 3.]
gini = 0.0000
samples = 1
value = [ 1. 0.]
gini = 0.0000
samples = 1
value = [ 1. 0.]
gini = 0.0000
samples = 1
value = [ 0. 1.]

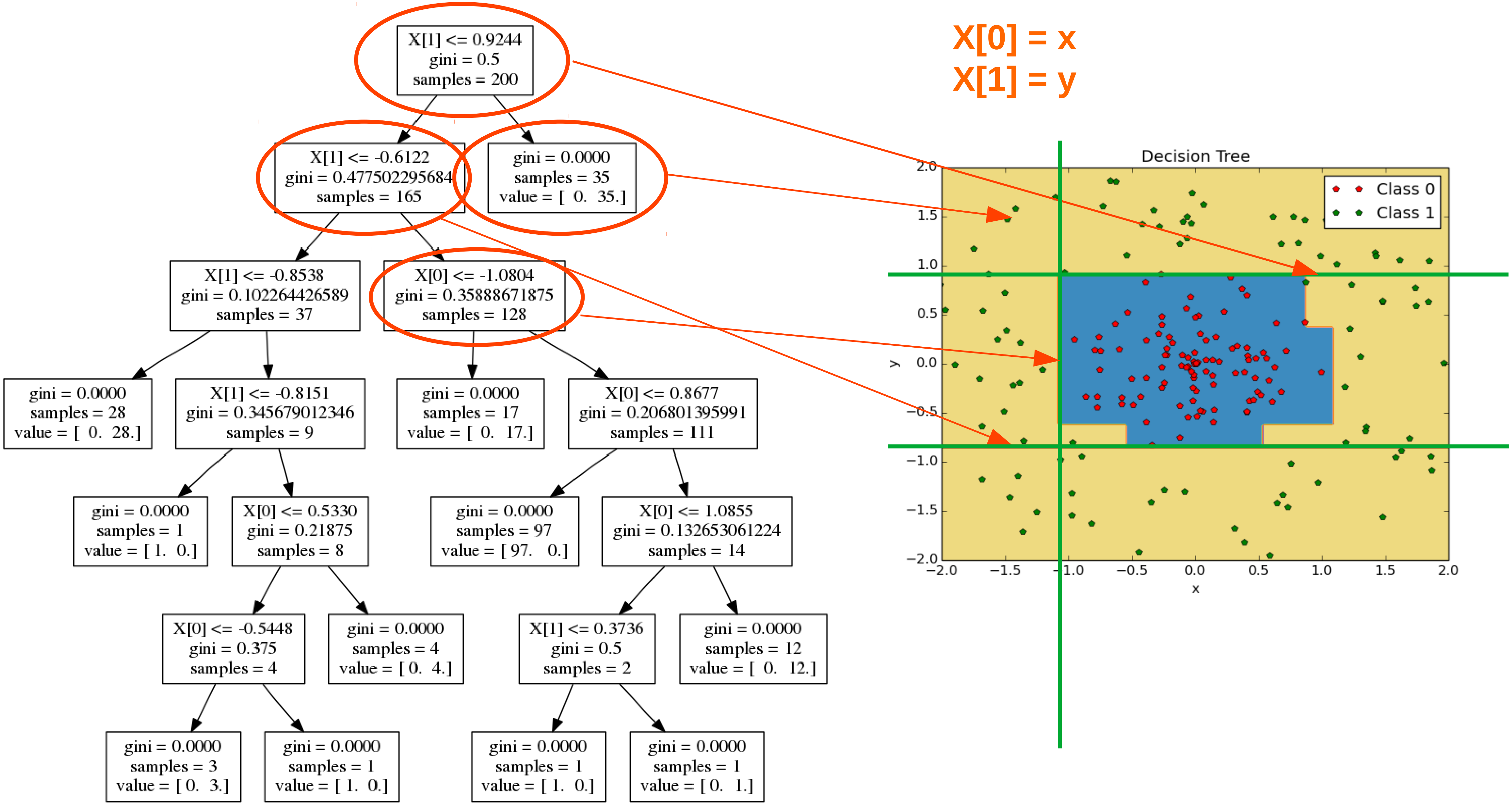
**X[0] = x**
**X[1] = y**


Decision Tree
Class 0
Class 1
y
x


X[1] <= 0.9244
gini = 0.5
samples = 200
X[1] <= -0.6122
gini = 0.477502295684
samples = 165
gini = 0.0000
samples = 35
value = [ 0. 35.]
X[1] <= -0.8538
gini = 0.102264426589
samples = 37
X[0] <= -1.0804
gini = 0.35888671875
samples = 128
gini = 0.0000
samples = 28
value = [ 0. 28.]
X[1] <= -0.8151
gini = 0.345679012346
samples = 9
gini = 0.0000
samples = 17
value = [ 0. 17.]
X[0] <= 0.8677
gini = 0.206801395991
samples = 111
gini = 0.0000
samples = 1
value = [ 1. 0.]
X[0] <= 0.5330
gini = 0.21875
samples = 8
gini = 0.0000
samples = 97
value = [ 97. 0.]
X[0] <= 1.0855
gini = 0.132653061224
samples = 14
X[0] <= -0.5448
gini = 0.375
samples = 4
gini = 0.0000
samples = 4
value = [ 0. 4.]
X[1] <= 0.3736
gini = 0.5
samples = 2
gini = 0.0000
samples = 12
value = [ 0. 12.]
gini = 0.0000
samples = 3
value = [ 0. 3.]
gini = 0.0000
samples = 1
value = [ 1. 0.]
gini = 0.0000
samples = 1
value = [ 1. 0.]
gini = 0.0000
samples = 1
value = [ 0. 1.]

X[0] = x
X[1] = y


Decision Tree
Class 0
Class 1
y
x
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


X[0] = x
X[1] = y
X[1] <= 0.9244
gini = 0.5
samples = 200
X[1] <= -0.6122
gini = 0.477502295684
samples = 165
gini = 0.0000
samples = 35
value = [ 0. 35.]
X[1] <= -0.8538
gini = 0.102264426589
samples = 37
X[0] <= -1.0804
gini = 0.35888671875
samples = 128
gini = 0.0000
samples = 28
value = [ 0. 28.]
X[1] <= -0.8151
gini = 0.345679012346
samples = 9
gini = 0.0000
samples = 17
value = [ 0. 17.]
X[0] <= 0.8677
gini = 0.206801395991
samples = 111
gini = 0.0000
samples = 1
value = [ 1. 0.]
X[0] <= 0.5330
gini = 0.21875
samples = 8
gini = 0.0000
samples = 97
value = [ 97. 0.]
X[0] <= 1.0855
gini = 0.132653061224
samples = 14
X[0] <= -0.5448
gini = 0.375
samples = 4
gini = 0.0000
samples = 4
value = [ 0. 4.]
X[1] <= 0.3736
gini = 0.5
samples = 2
gini = 0.0000
samples = 12
value = [ 0. 12.]
gini = 0.0000
samples = 3
value = [ 0. 3.]
gini = 0.0000
samples = 1
value = [ 1. 0.]
gini = 0.0000
samples = 1
value = [ 1. 0.]
gini = 0.0000
samples = 1
value = [ 0. 1.]
Decision Tree
Class 0
Class 1
y
x

# Python Code

```
#Decision Tree Classifier
from sklearn import tree

model = tree.DecisionTreeClassifier()
model.fit(features, labels) #train the model

model.predict(test_features) #make predictions
#See documentation for other functions
```



Red Hat

# Decision Trees: Advantages

Minimal pre-processing required - no normalization for example

No assumptions about distribution of data

Simple to interpret - if not too deep

Can discover interactions between features automatically

Run fast

# Decision Trees: Disadvantages

Use ”horizontal” and ”vertical” lines to learn decision boundaries
Result in over-complex trees

Can easily overfit data

Unstable: small changes in data can lead to
very different trees

Cannot extrapolate

# K-Nearest Neighbors

Given N examples with features $x_1, \ldots, x_n$

Given N results $y$

New input features:

Find K "closest" examples

Average their results

# K-Nearest Neighbors

| k (Number of neighbors considered) | Accuracy |
|---|---|
| 1 | 96.60% |
| 2 | 95.72% |
| 3 | 96.63% |
| 4 | 96.33% |
| 5 | 96.50% |
| 6 | 96.42% |
| 7 | 96.40% |
| 8 | 96.26% |
| 9 | 96.16% |
| 10 | 96.08% |

**12,600 samples in test set**

All neighbors given equal weight – take majority vote

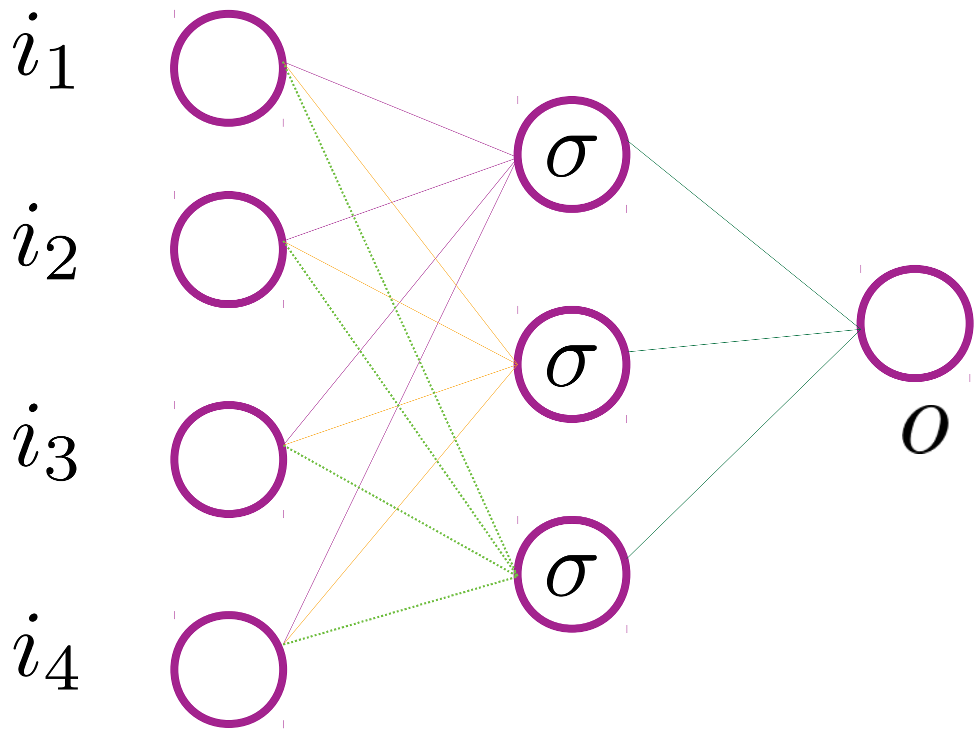


Red Hat

# K-Nearest Neighbors

- Labels not uniformly distributed in train data – what if digit “9” occupies 95% of data and the rest only 5% combined. Neighbors biased towards “9”.
- High dimensionality of feature space

# Neural Networks



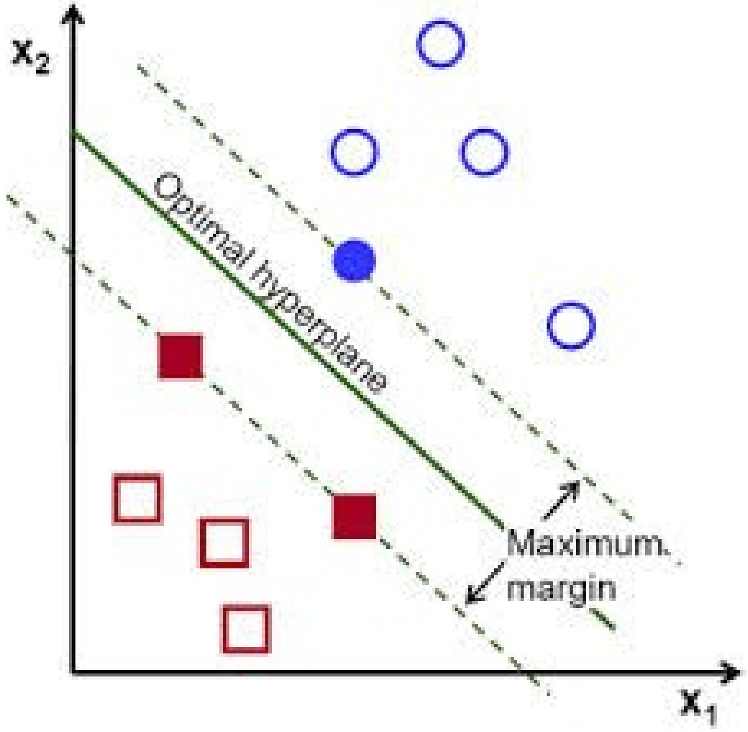

Layered switches that fire (output = 1) if input to switch exceeds threshold, otherwise don't fire (output = 0).

Switches can be combined to form complex logic gates that can “take decisions” based on input

Initially modeled on neurons in brain but anatomically, extremely crude approximation.

Analogy not useful any more.

# Support Vector Machines




Pick boundary that creates the largest “gap” between two classes
(Unfortunately, not covered here)

# Unsupervised Learning

# K-Means Clustering

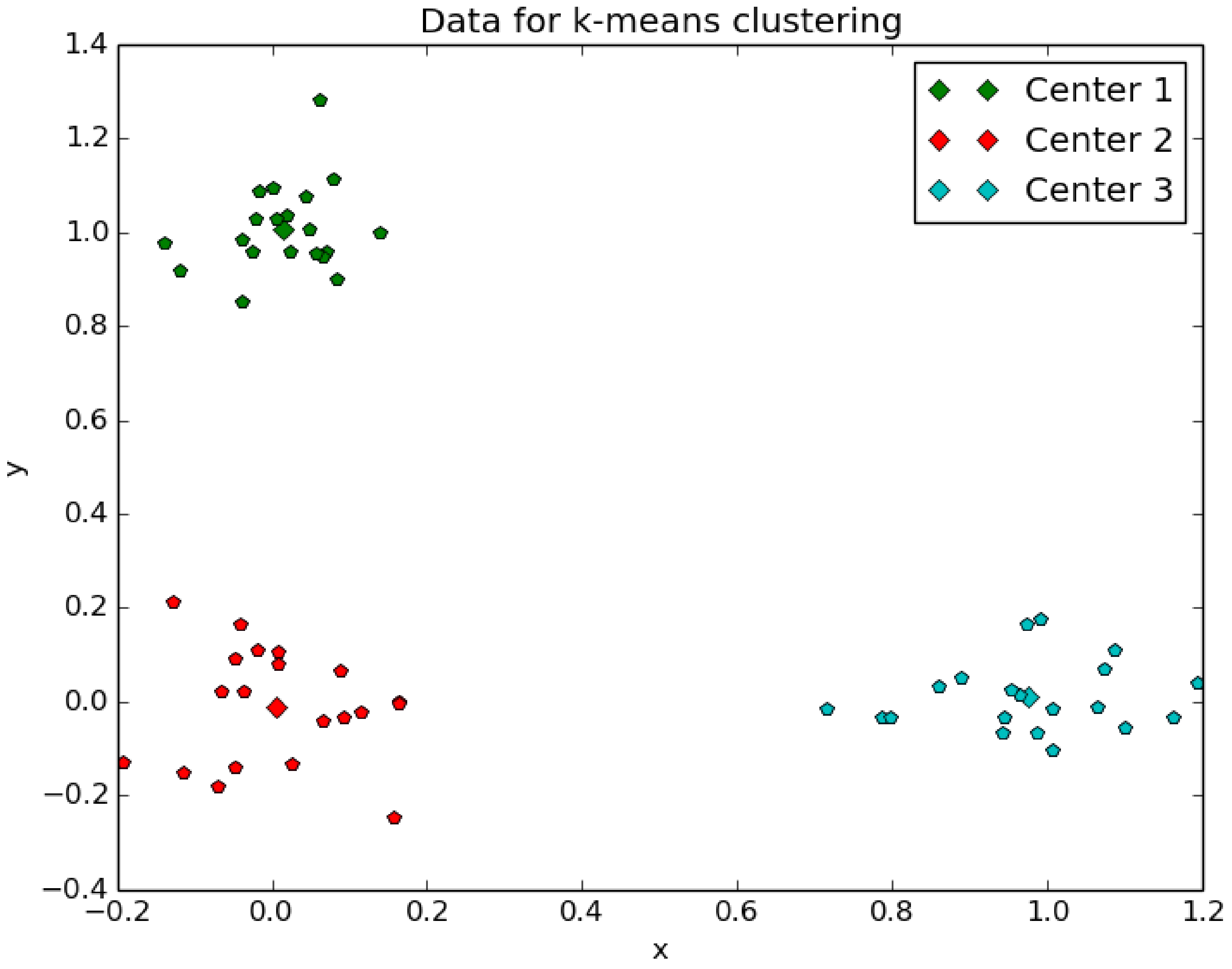
Define **distance** between two examples: $d(\vec{x}_i, \vec{x}_j) \in \mathbb{R}$

Features for example i

Features for example j

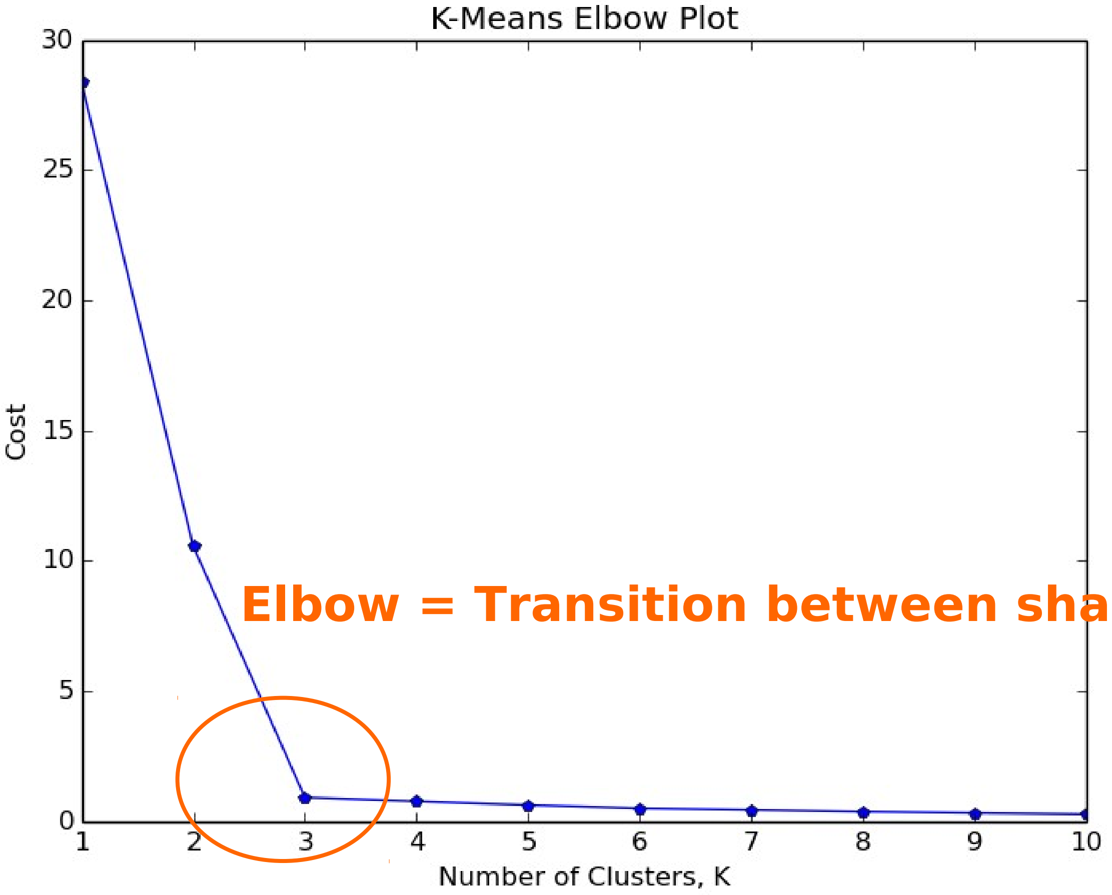
Analyze/plot each group/cluster separately

# K-Means Clustering




**3 Clusters**

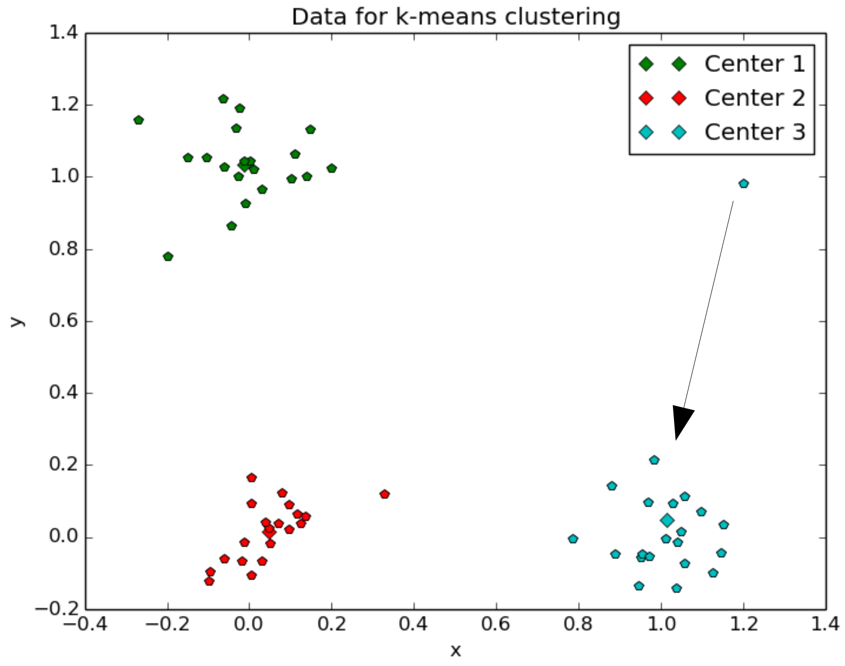
# Choosing k – Elbow Plot

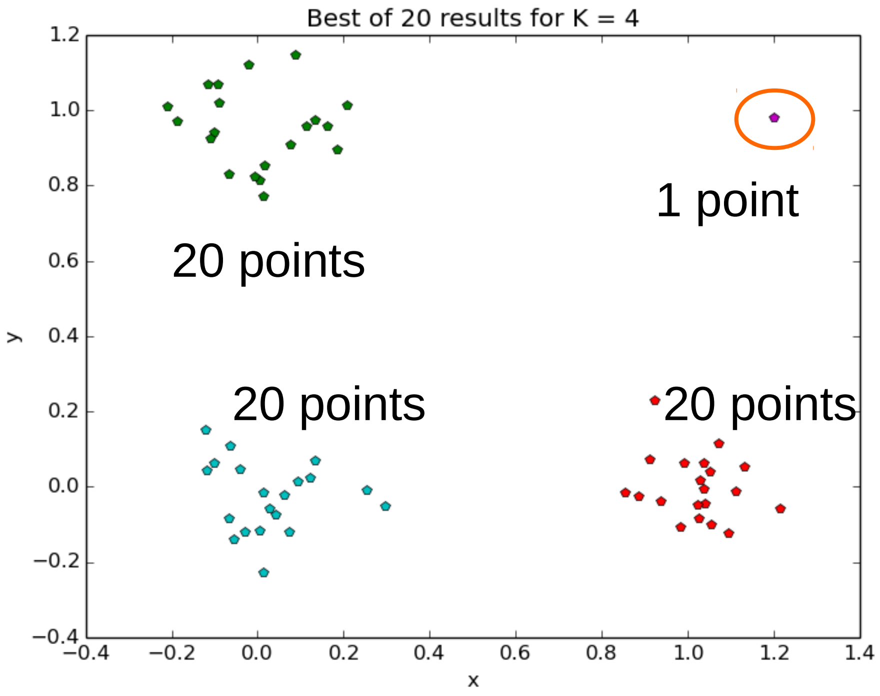



**Elbow = Transition between sharp drop and stabilization**

**Choose K = 3**

# Detect Outliers




**3 clusters**

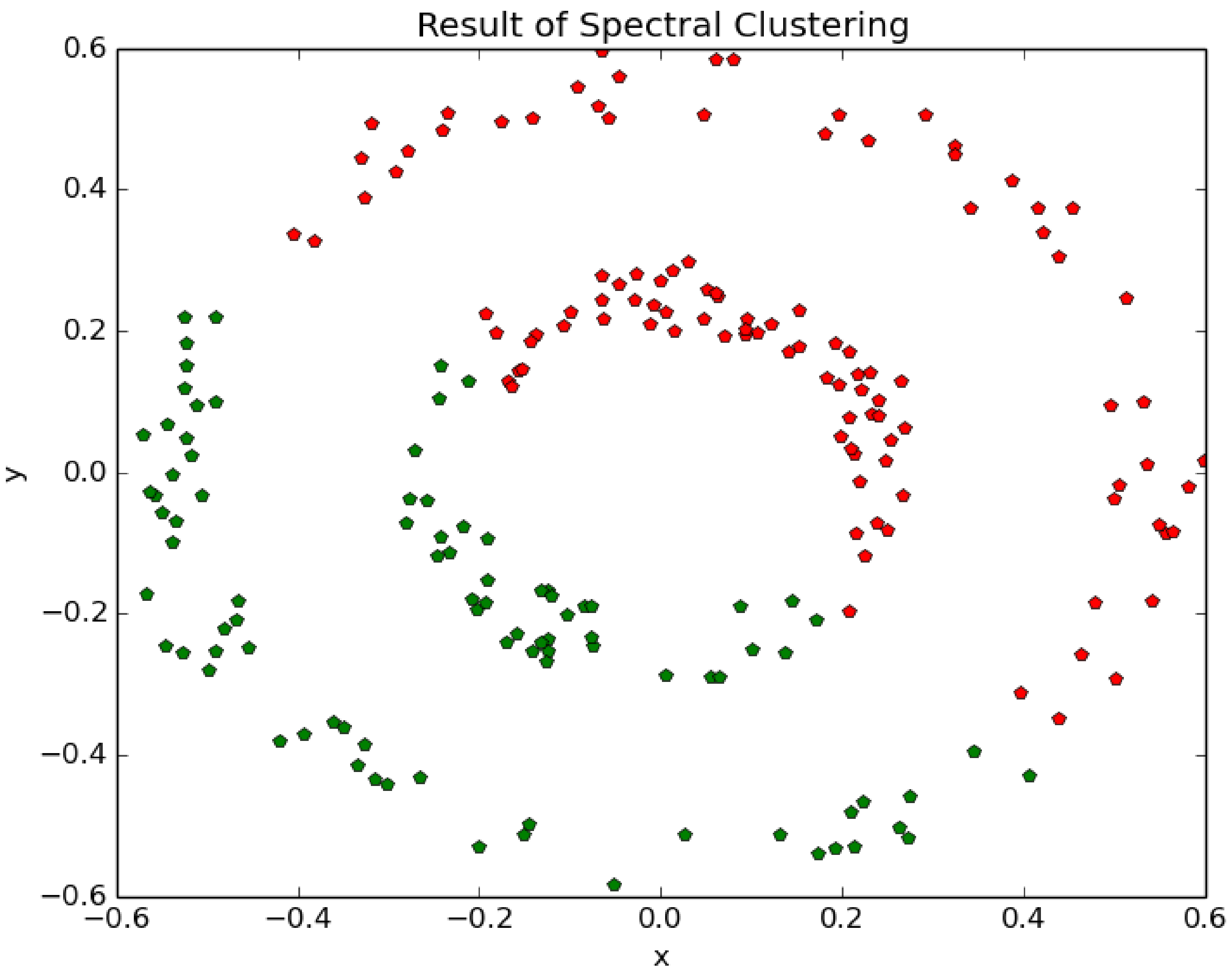
**Cost = 2.05**

**4 clusters**

**Cost = 1.19**

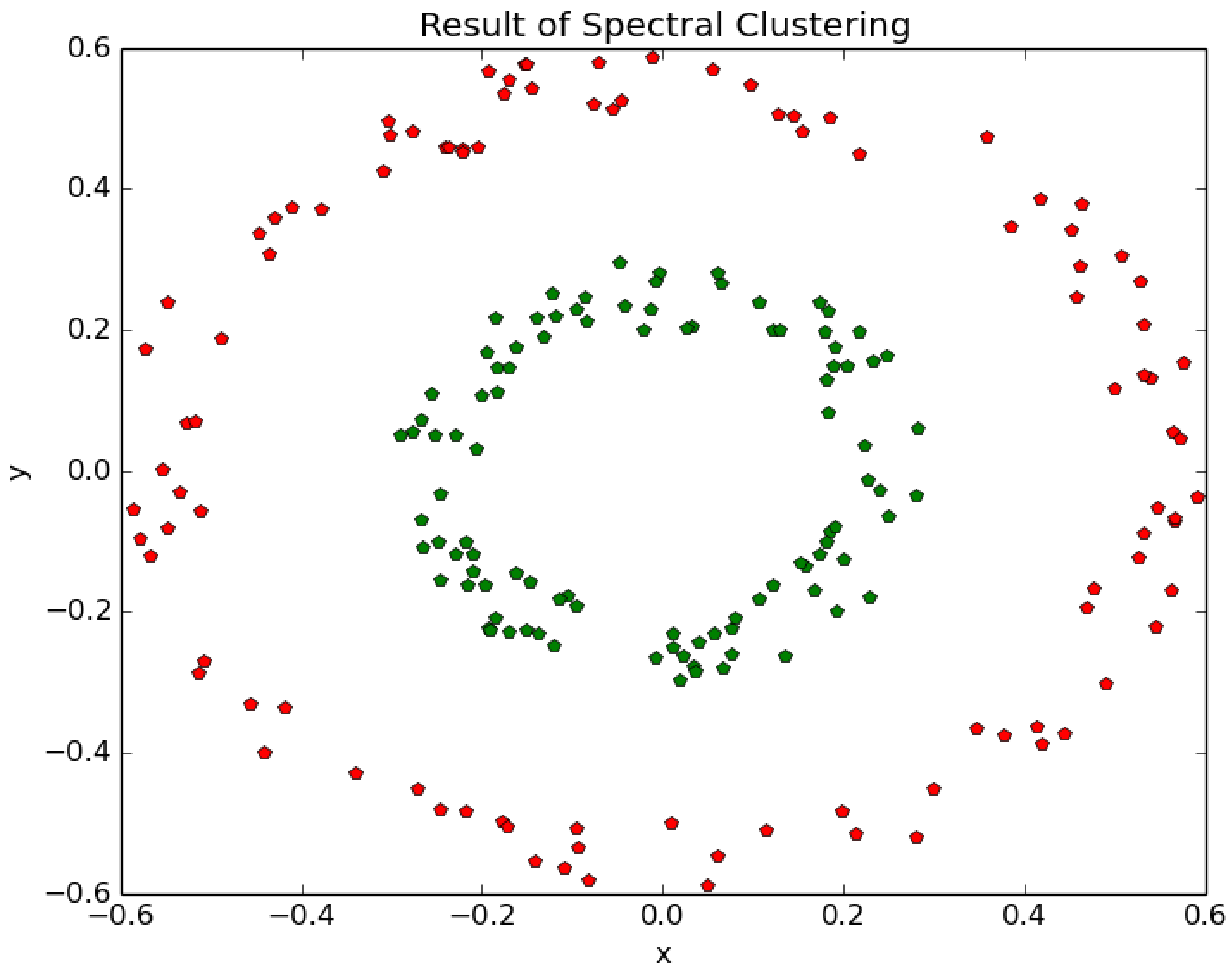
**Best solution over 20 runs**

# Spectral Clustering




Try K-Means with K = 2

# Spectral Clustering




Try spectral clustering with $K = 2$

# Python Code

```
#Clustering with K-means or Spectral clustering
from sklearn.cluster import KMeans, SpectralClustering

model = KMeans(n_clusters = 2)
model.fit(features) #find clusters
ids = model.predict(features) #predict cluster IDs

model = SpectralClustering(n_clusters = 2)
model.affinity("nearest_neighbors")
model.fit(features)
ids = model.predict(features)
```

k-means

Spectral

# Warning with Clustering

Good to re-scale each feature to have $\underbrace{\mu = 0}_{\text{Mean}}$ and $\underbrace{\sigma = 1}_{\text{Standard Deviation}}$

$$x_m \rightarrow x'_m \equiv \frac{x_m - \mu(x_m)}{\sigma(x_m)}$$

Want numbers in similar range for each features

$$\sqrt{\underbrace{(x_1^{(1)} - x_1^{(2)})^2}_{\text{Feature 1}} + \underbrace{(x_2^{(1)} - x_2^{(2)})^2}_{\text{Feature 2}}}$$

If Feature 2 $\approx$ 10*Feature 1, distance calculation dominated by Feature 2

$$10^2 + 1000^2 \approx 1000^2$$

# Principal Component Analysis

Have n features: $x_1, \ldots, x_n$

Each example is a point in n-dimensional space

Can we visualize the data?

Do we really need all the features?

<u>Example:</u> Plane described by: $4x + 5y - 6z = 1$

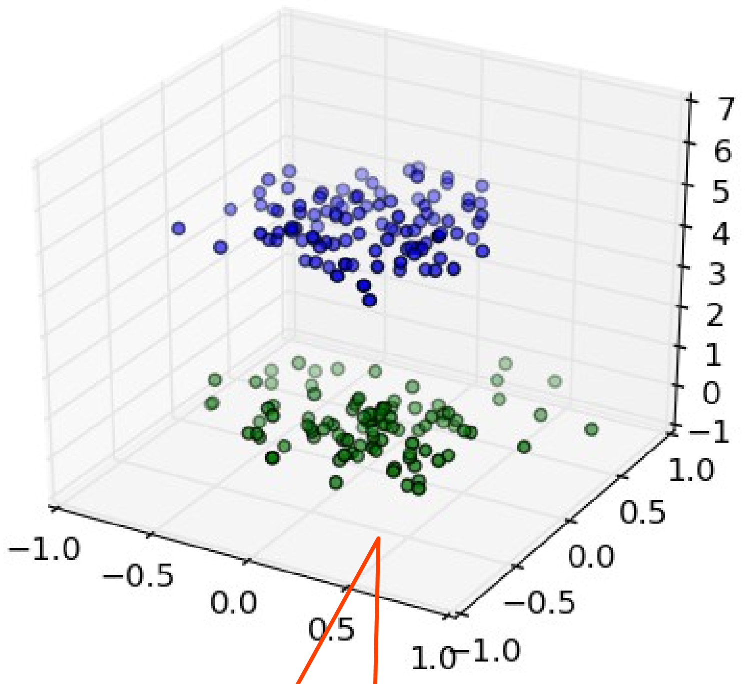
2-dimensional surface embedded in 3 dimensions

Need only 2 coordinates not 3

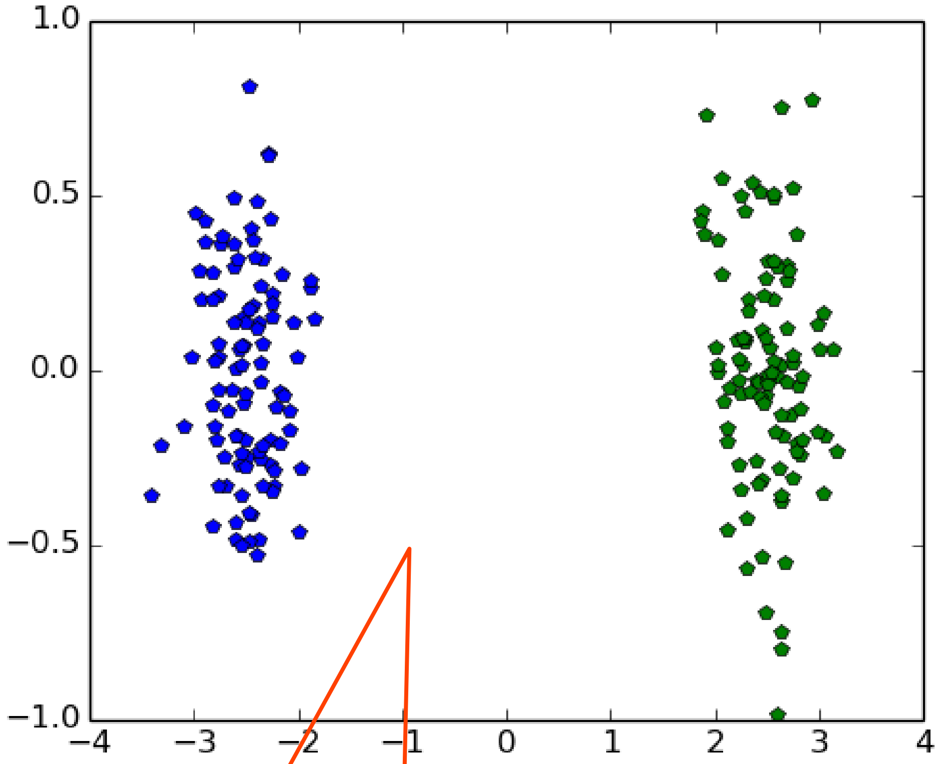


Red Hat

# Principal Component Analysis

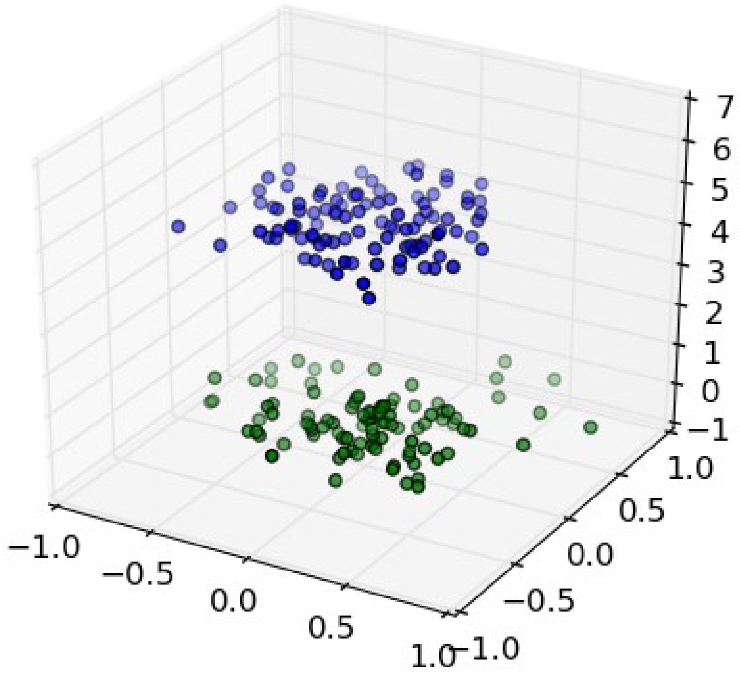


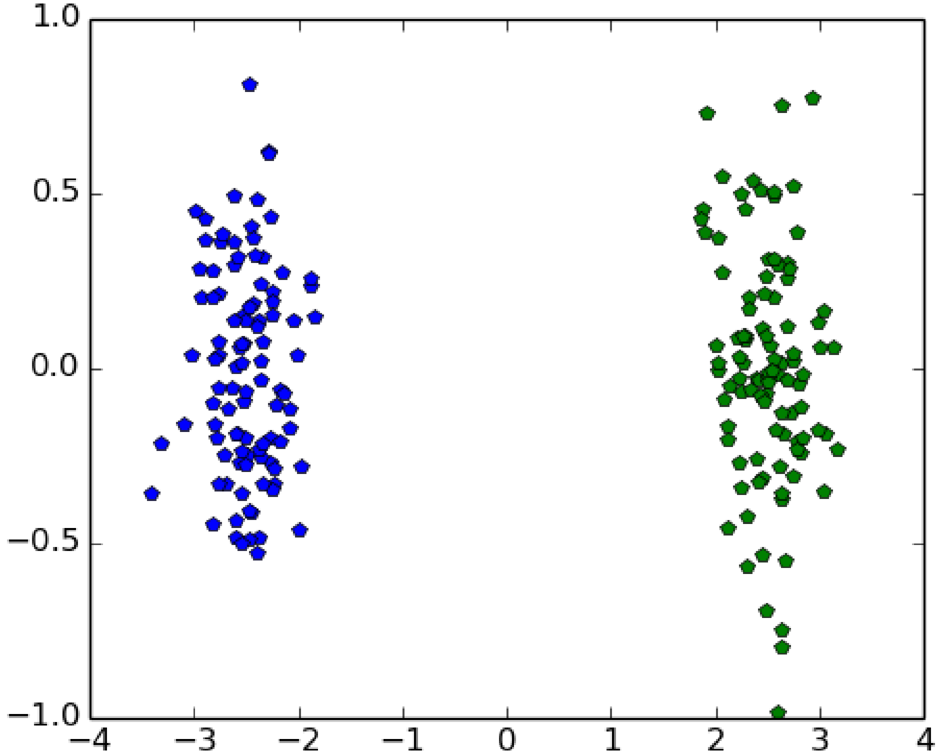

Random 3-D data set




Projected to 2-D

# Principal Component Analysis

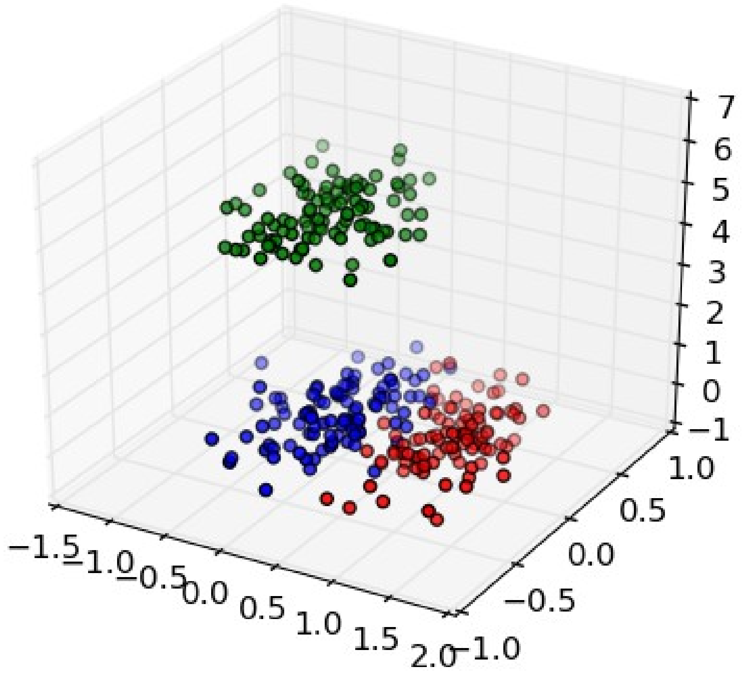


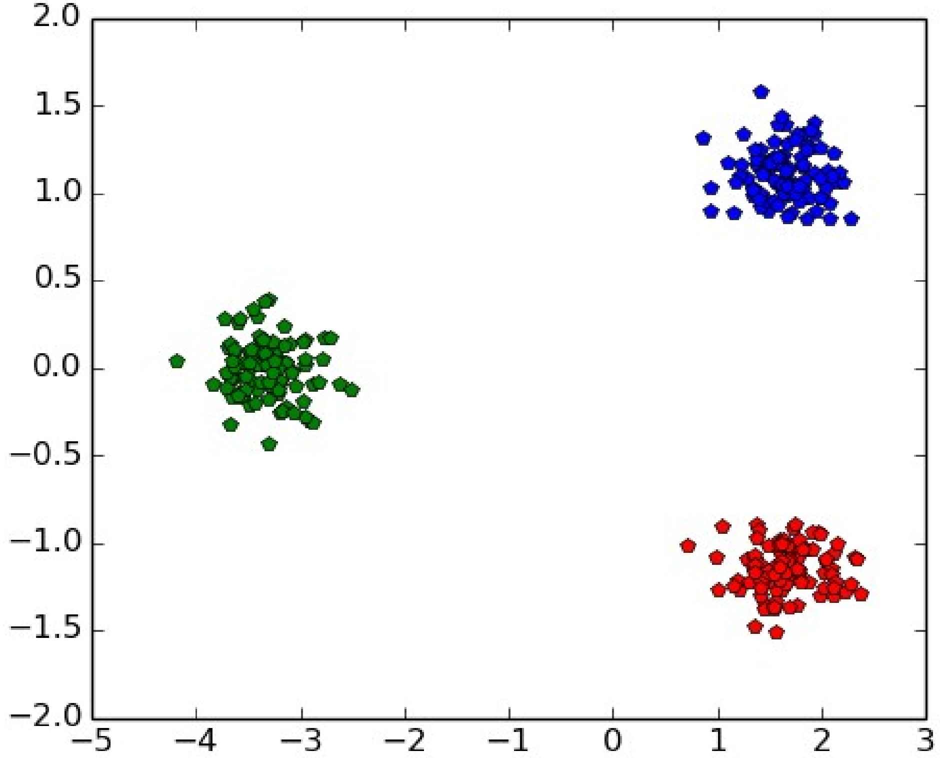





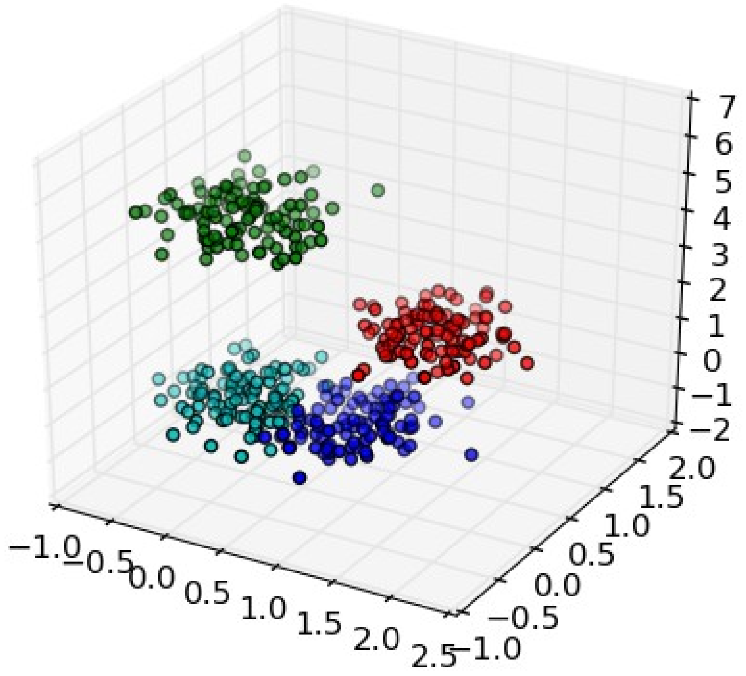

Red Hat

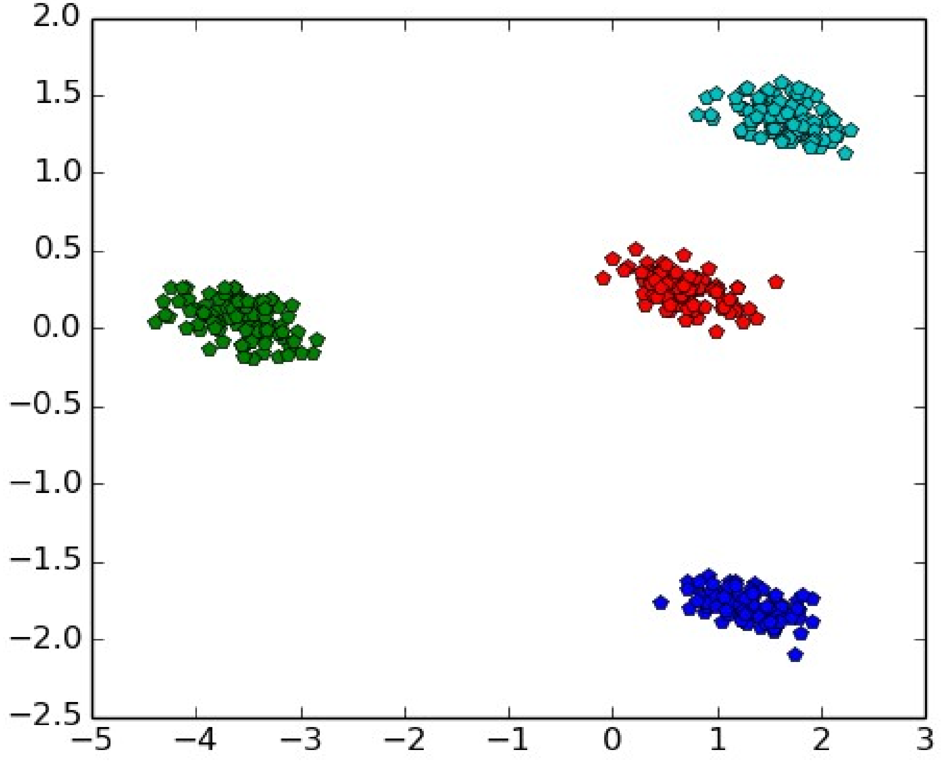
# Principal Component Analysis



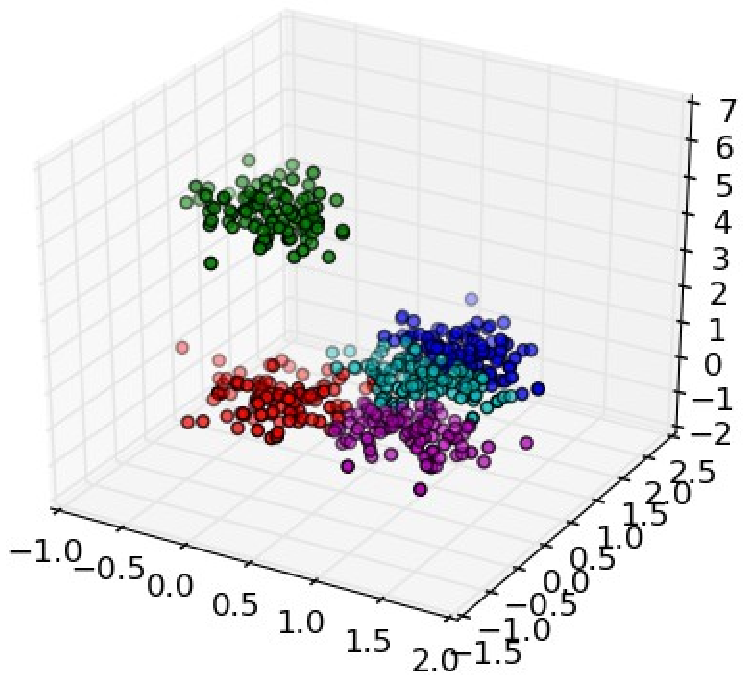



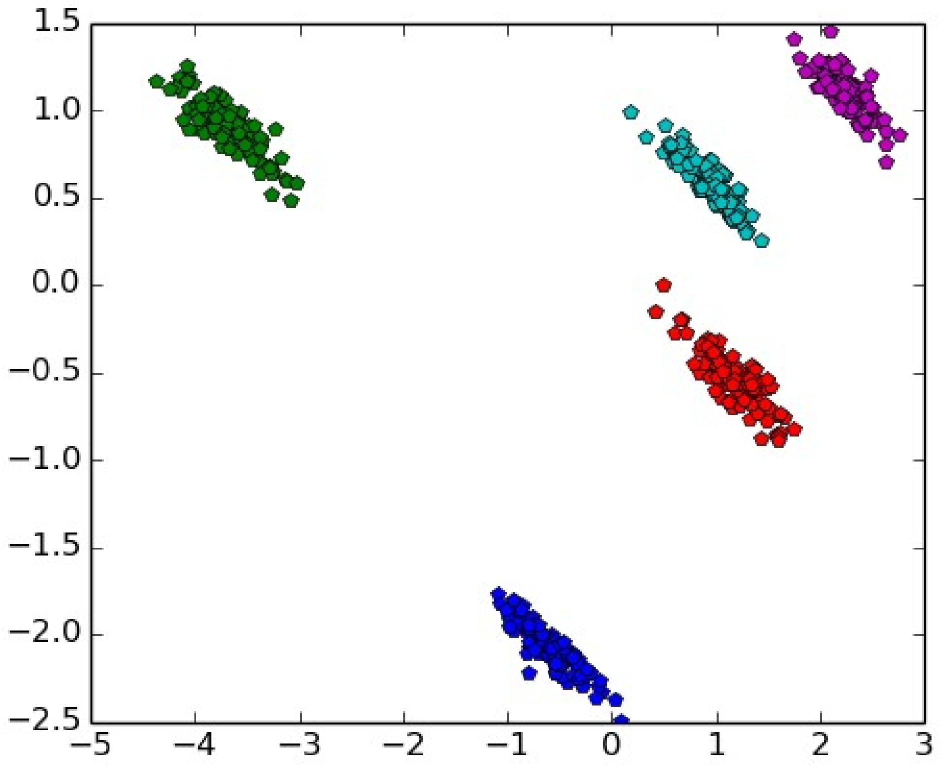
# Principal Component Analysis



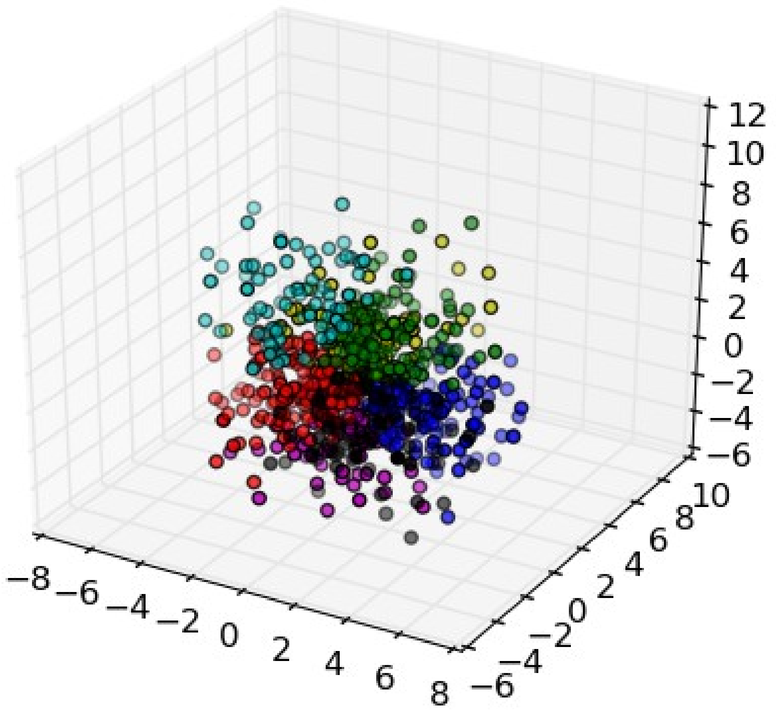



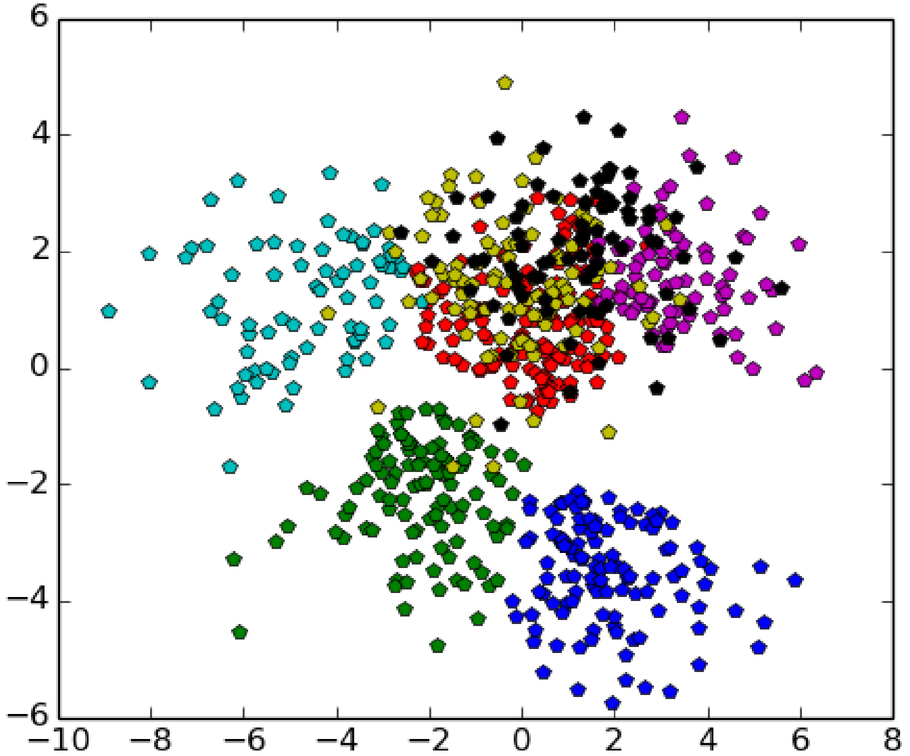
# Principal Component Analysis



# Principal Component Analysis






**Standard deviation of Gaussian = 2: mixed results**

Red Hat

# Principal Component Analysis

```
#Principal Component Analysis
from sklearn.decomposition import PCA
model = PCA(n_components = 10)

model.fit(features)
transformed_features = model.transform(features)

model.components_ #new variables
model.explained_variance_ratio #% of variance explained by each component
```

**Each feature should have mean = 0!**

# General Modeling Concepts

# Choosing the “best” model

You built a model

How do you know if it works?

What does “working” mean?

# Typical Scientific Process

Do experiments. Collect data.

# Typical Scientific Process

Do experiments. Collect data.

Build model/equations to explain data.

# Typical Scientific Process

Do experiments. Collect data.

Build model/equations to explain data.

Make predictions about new experiments.

# Typical Scientific Process

Do experiments. Collect data.

Build model/equations to explain data.

Make predictions about new experiments.

Do new experiment. Compare predictions to experimental results.

# Typical Scientific Process

Do experiments. Collect data.

Build model/equations to explain data.

Make predictions about new experiments.

Do new experiment. Compare predictions to experimental results.

If results match, model might be correct.

# Typical Scientific Process

Do experiments. Collect data.

Build model/equations to explain data.

Make predictions about new experiments.

Do new experiment. Compare predictions to experimental results.

If results don’t match, tweak model or build new one.

# Typical Scientific Process

Model might work well on already observed data.

After all, we are building the model after looking at this data.

Do experiments. Collect data.

Build model/equations to explain data.

Model might fail terribly when it comes to making predictions, explaining new data

Make predictions about new experiments.

Do new experiment. Compare predictions to experimental results.

If results match, model might be correct.

If results don't match, tweak model or build new one.

# Model Validation

Have N rows of data with features (inputs) and results (outputs)

Split into two pieces, say 70%-30%
(just a proposal – nothing sacred about these numbers)

Training data: 70%

Test data: 30%

# Model Validation

Have N rows of data with features (inputs) and results (outputs)

Split into two pieces, say 70%-30%
(just a proposal – nothing sacred about these numbers)

Training data: 70%

**Build model using this data**

Test data: 30%

**Pretend this doesn't exist**

# Model Validation

Have N rows of data with features (inputs) and results (outputs)

Split into two pieces, say 70%-30%
(just a proposal – nothing sacred about these numbers)

Training data: 70%

**Build model using this data**

**Evaluate performance here according to some metrics**

Test data: 30%

**Pretend this doesn't exist**

# Model Validation

Have N rows of data with features (inputs) and results (outputs)

Split into two pieces, say 70%-30%
(just a proposal – nothing sacred about these numbers)

Training data: 70%

**Build model using this data**

**Evaluate performance here according to some metrics**

Test data: 30%

**Pretend this doesn't exist**

**Make predictions here and compute same metrics**

# Model Validation

Have N rows of data with features (inputs) and results (outputs)

Split into two pieces, say 70%-30%
(just a proposal – nothing sacred about these numbers)

Training data: 70%

**Build model using this data**

**Evaluate performance here according to some metrics** ►

Test data: 30%

**Pretend this doesn't exist**

**Make predictions here and compute same metrics**

**If metrics have similar values, model works well on "new" test data**

# Model Validation

Have N rows of data with features (inputs) and results (outputs)

Split into two pieces, say 70%-30%
(just a proposal – nothing sacred about these numbers)

| <u>Training data</u>: 70% | <u>Test data</u>: 30% |
|---|---|
| **Build model using this data** | **Pretend this doesn't exist** |
| **Evaluate performance here according to some metrics** → | **Make predictions here and compute same metrics** |



**If metrics have very different values, model works very well on training data but doesn't capture underlying model well and so works badly on test data - <u>overfitting</u>**

# Model Validation

Have N rows of data with features (inputs) and results (outputs)

Split into two pieces, say 70%-30%
(just a proposal – nothing sacred about these numbers)

| Training data: 70% | Test data: 30% |
|---|---|
| **Build model using this data** | **Pretend this doesn't exist** |
| **Evaluate performance here according to some metrics** → | **Make predictions here and compute same metrics** |



**If model performs badly on training data itself - underfitting**

# Model Validation

Good results on training **AND** test sets

**DON'T GUARANTEE**

good results on future data

# Model Validation

Good results on training **AND** test sets

**DON'T GUARANTEE**

good results on future data

**Assumption 1**:
Statistical distributions in train and test set are similar and will stay so in future data

**Assumption 2**:
Fundamental rules/processes governing the system won't change

# Cross-Validation

Some models have parameters that can't be decided by training data

Parameters label family of models

Example: Linear Regression with "Regularization"

Penalty term introduced in cost term to prevent overfitting

Free parameter, $\alpha$

# Cross-Validation

Example: Linear Regression with "Regularization"

Penalty term introduced in cost term to prevent overfitting

Free parameter, $\alpha$

```
model = linear_model.Ridge(alpha = 0.5) #"Ridge" regression with one free parameter

model = linear_model.Lasso(alpha = 0.5) #"Lasso" regression with one free parameter
```

**How do we choose this parameter?**

**In other words, how do we choose the optimal model from this family of models?**

# Cross-Validation

Have N rows of data with features and results

Split into three pieces, say 60%-20%-20%

(just a proposal!!!)

<u>Training data:</u> 60%

<u>Test data:</u> 20%

<u>Cross-validation data:</u> 20%

# Cross-Validation

Training data: 60%

Train models for different $\alpha$ using this

Cross-validation data: 20%

Apply here

Pick $\alpha$ such that error minimized on cross-validation data

Apply to test data

# Model Selection

- Limited amount of data available
- Reuse data (in a controlled fashion)
- K-fold:
  - Split data into k chunk
  - Use k-1 chunks for training, remaining for testing
  - Loop, selecting a different chunk as testing set each time
  - Average errors over all loop iterations

# Model Selection – Python Code

```
#Cross-validation

from sklearn import cross_validation

train_features, train_labels, test_features, test_labels = cross_validation.train_test_split(features, labels, test_size = 0.30)


from sklearn.cross_validation import KFold

kf = KFold(4, n_folds = 4)

for train, test in kf:
    print train, test
    train_features, test_features, train_labels, test_labels = features[train], features[test], labels[train], labels[test]

    #Do model fitting/predicting here
```

Split data into 4 chunks. Use 3 for training, 1 for testing, then average.

Check out sklearn.cross_validation

http://scikit-learn.org/stable/model_selection.html#model-selection

# Some Take-away Points

There are many techniques and new ones are always being invented

Guidelines but no strict rules

Be skeptical. Algorithms will always produce **some numbers**

Try techniques on as many datasets as you can!

RH problems Kaggle problems data.gov

# Some Take-away Points

There are many techniques and new ones are always being invented

Guidelines but no strict rules

Be skeptical. Algorithms will always produce **some numbers**

Try techniques on as many datasets as you can!

RH problems Kaggle problems data.gov

Don't be afraid of being wrong!

Talk to people

Keep playing with datasets and algorithms



Red Hat

# Thank you

Red Hat is the world's leading provider of

enterprise open source software solutions.

Award-winning support, training, and

consulting services make

Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat