Introduction to Machine Learning

Saurabh Sinha
Dept. of Biomedical Engineering,
Dept. of Industrial & Systems Engineering
Georgia Institute of Technology
saurabh.Sinha@bme.gatech.edu

A gentle introduction

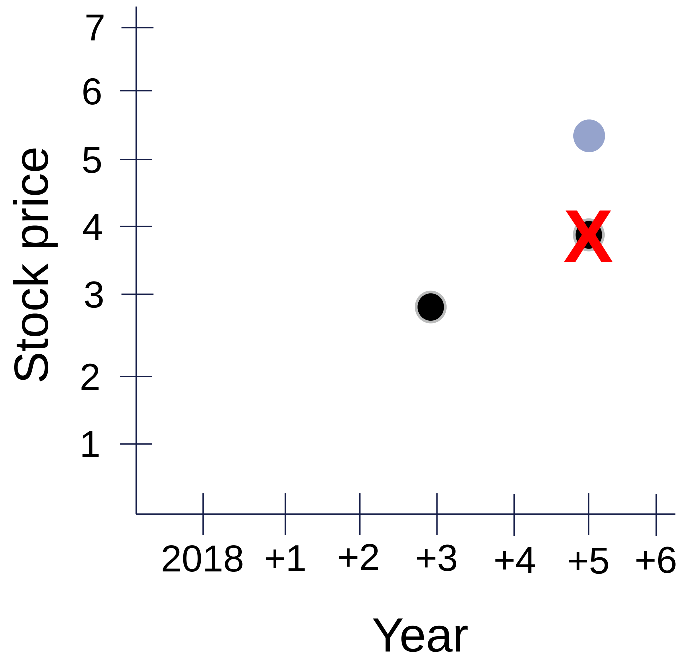
What is machine learning?

- Wikipedia: study of “algorithms that improve automatically through experience”
- Q: What is “experience”?
- A: Seeing more data
- Q: “improve” what?
- A: Ability to make predictions from data

What is machine learning?

Q: How does the computer “learn” to make predictions?

A: (Example) I buy a stock for $3 this year. What will be its price in the future?



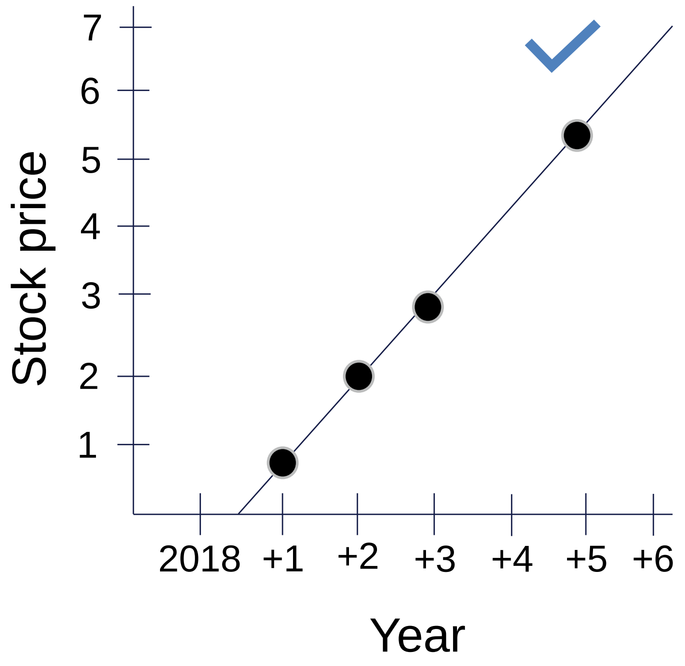
What will be the stock price in year 5?

Unable to make an accurate prediction with one data point.

What is machine learning?

Q: How does the computer “learn” to make predictions?

A: (Example) I buy a stock for $3 this year. What will be its price in the future?



What if we know the price in the past two years?

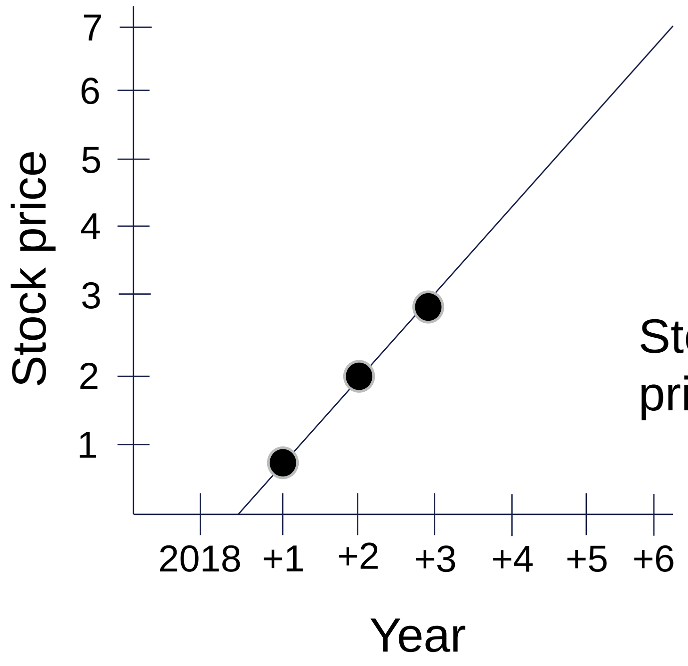
Now can we predict the stock price in year 5?

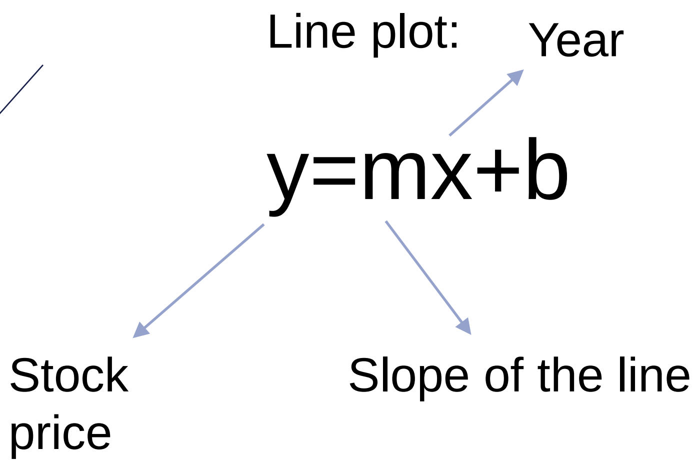
As we add additional data points (“training data”), an algorithm may be able to make such predictions more accurately.

What is machine learning?

Q: How does the computer “learn” to make predictions?

A: (Example) I buy a stock for $3 this year. What will be its price in the future?





Machine learning can use the same idea of “learning from data” with more complex functions than y=mx+b.

What is machine learning?

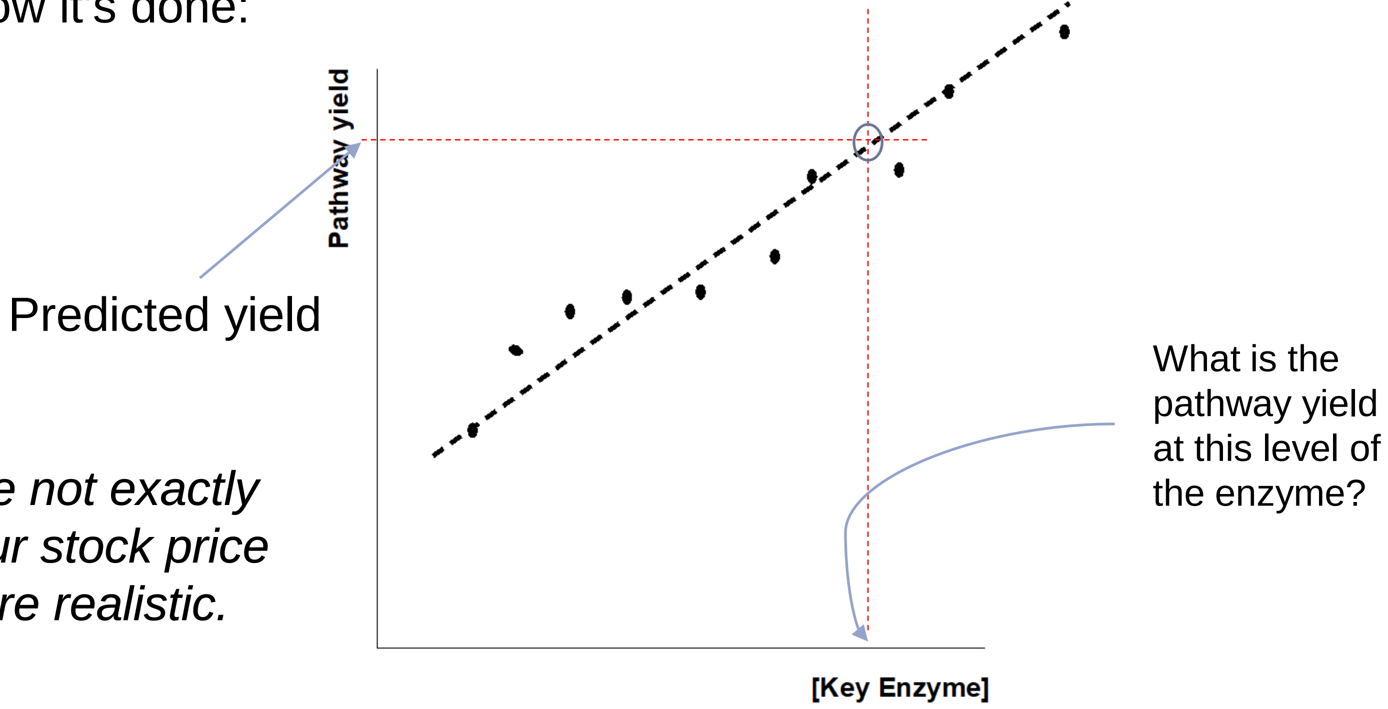
- What sorts of predictions can be made?
- A: (Toy example) Price of a stock in a future year
- A: (Example) Gene expression in a cellular condition.
  - A gene’s expression may be predictable if you know the levels of all transcription factors.
- A: (Example) Enzyme activity.
  - An enzyme’s ability to catalyze a reaction may be predictable from its sequence.

- An algorithm may be able to make such predictions accurately.

What is machine learning?

Q: Do you need to teach the algorithm all about chemistry or biology?

A: No. Here's how it's done:



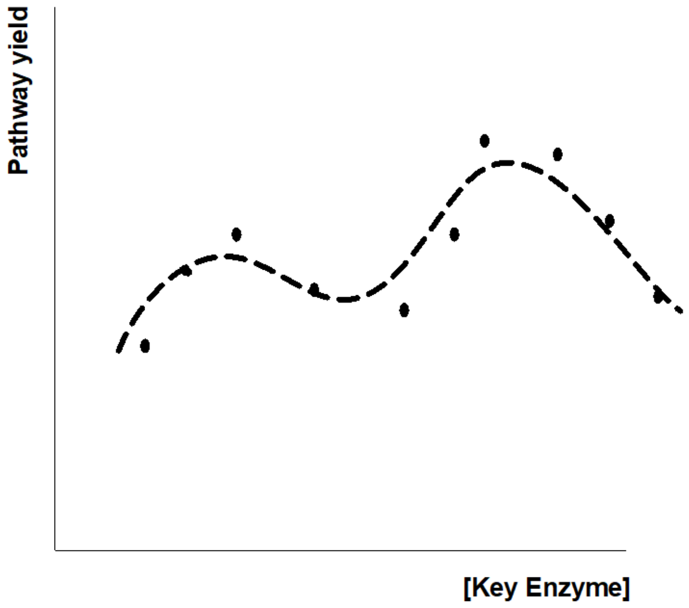
Side note: Points are not exactly on the line, unlike our stock price example. This is more realistic.

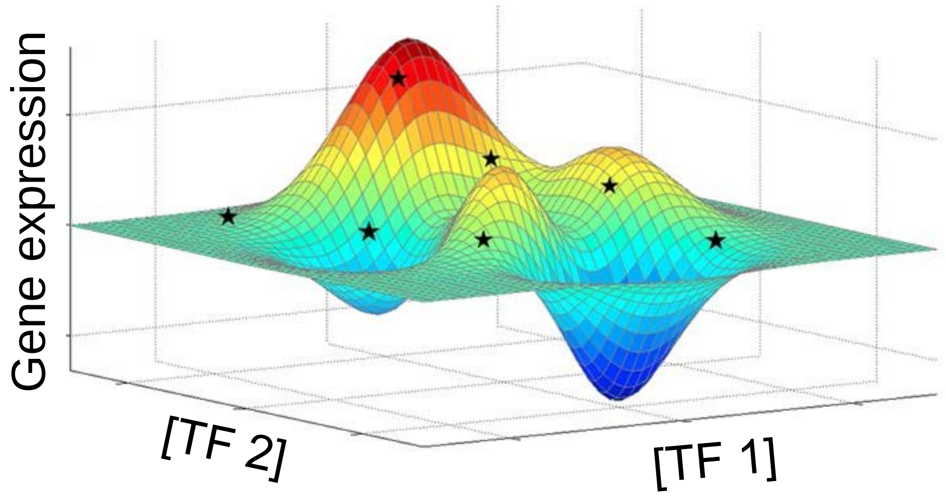
The algorithm LEARNS A FUNCTION mapping input to output.

What is machine learning?

Q: But that is too naïve. It won’t work in biology and medicine.

A: It’s not (too naïve). And it does work for many complex domains, including biology. Because the function learnt can be fairly complex:





Two major types of “prediction” tasks

Regression: Goal is to predict something numeric

Example:

- Input: Transcription factor concentrations
- Question: What is the gene’s expression level?
- Output: Could be an expression level in TPM

Classification: Goal is to predict “yes” or “no”

Example:

- Input: Protein sequence
- Question: Is it a biocatalyst?
- Output: Yes or no

Part 1: Classification

Example Motivation

- Predicting the function of genes by using gene expression data
- A gene that is expressed in samples from yeast colonies in oxygen-deprived conditions and not in other colonies may have a stress-response function.
- How can a Machine Learning program capture this intuition?

Supervised Learning & Training Data

- You have an **expression data set**, i.e., measurements of all genes’ expression in a range of biological conditions or individuals/organisms.

- Begin with a set of genes that have a common function, say “stress response”. This is called the “**positive set**”.

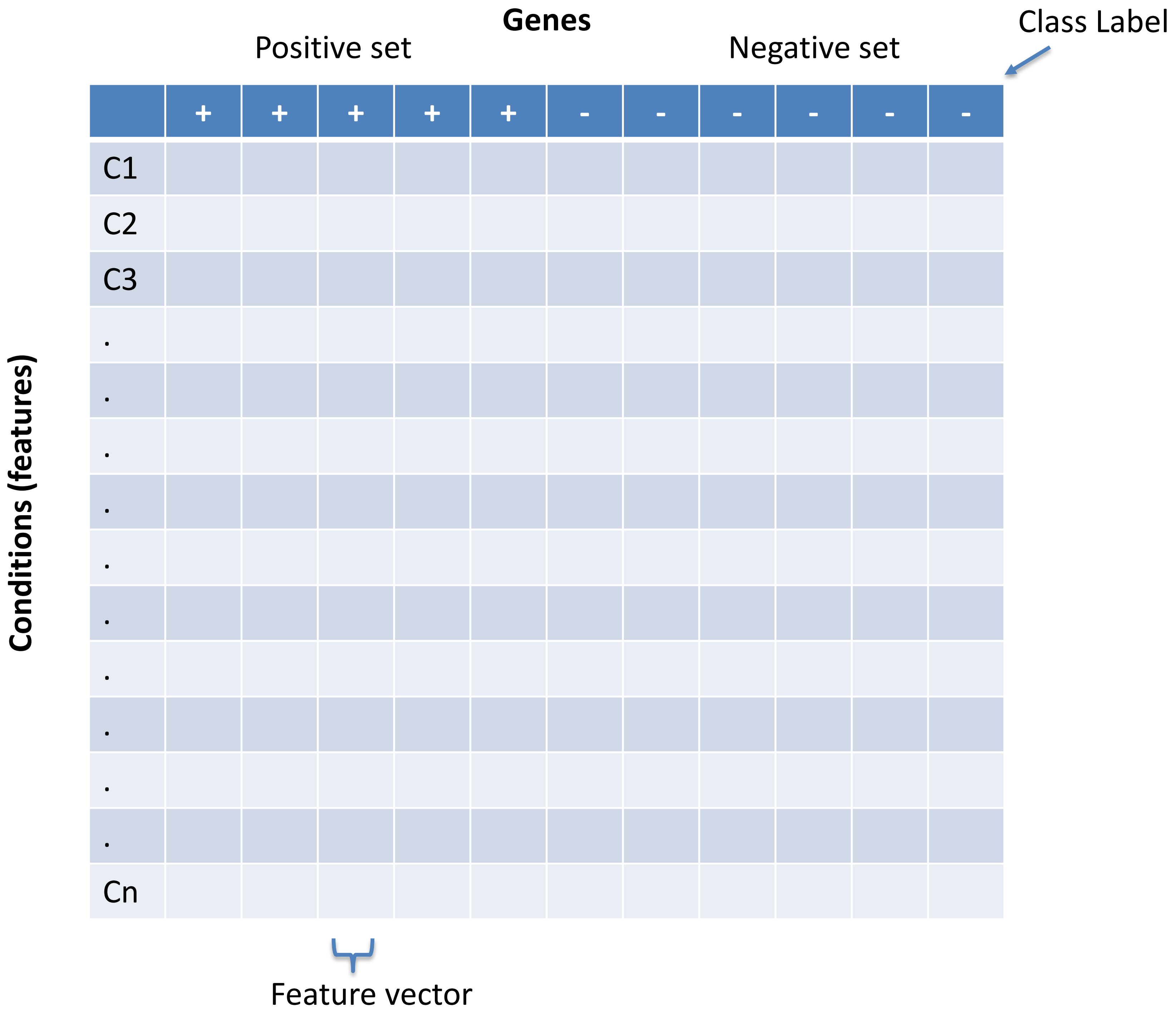
- Also find a set of genes known not to be members of that functional class (the “**negative set**”)

- Such sets can be assembled from the literature on gene functions

“Features”

- Each gene (positive or negative) is described by a vector of its expression levels in different conditions/individuals. This is the “**feature vector**”.

- The positive and negative gene sets, along with the feature vectors of those genes, form the “**training data**”


Genes
Positive set
Negative set
Class Label
+ + + + + - - - - - -
C1
C2
C3
.
.
.
.
.
.
.
.
.
.
Cn
Conditions (features)
Feature vector

Classification

- A **classification** algorithm will learn to discriminate between the genes in the two classes, based on their expression profiles (feature vectors)

- Once learning is done, the classifier may be presented with the expression profile of a previously unseen gene ("test data")

- Should be able to predict if the gene is a member or non-member of the functional class

Formally

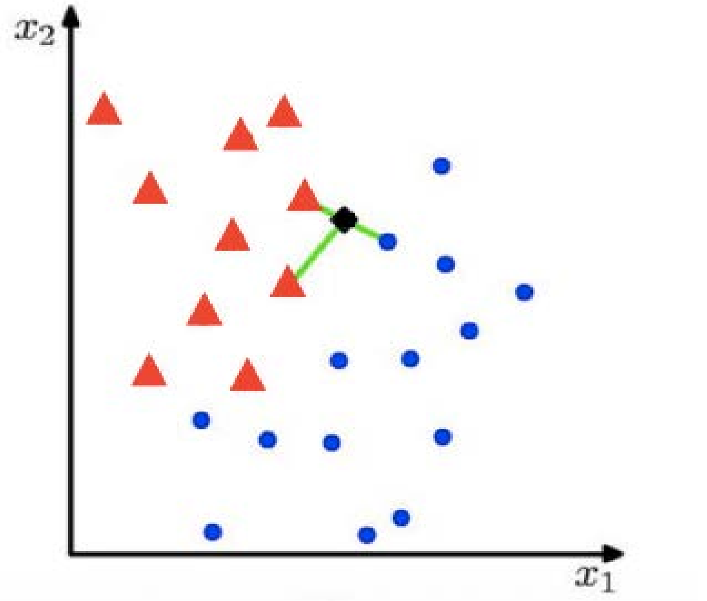
- Each sample (e.g., gene) is described by a "feature vector" $\mathbf{x}_i$
- This is a *d*-dimensional vector; i.e., $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{id})$, where *d* is the number of conditions for which expression data are available.
- Samples corresponding to class "-1" are $\mathbf{x}_1, \dots \mathbf{x}_n$
- Samples corresponding to class "+1" are $\mathbf{x}_{n+1}, \dots \mathbf{x}_{n+m}$
- All samples ($\mathbf{x}_1, \dots \mathbf{x}_{n+m}$) with their class labels (+1/-1) form the training data
- The problem is to design a function $f(\mathbf{x})$ that predicts the class label (+1 or -1) of any sample **x**. This is the classification problem.
- "Training the classifier" typically amounts to finding $f(\mathbf{x})$ such that $f(\mathbf{x}_i)=-1$ for all i=1 … n and $f(\mathbf{x}_i)=1$ for all i=n+1 … n+m

K-nearest neighbor (KNN) classification

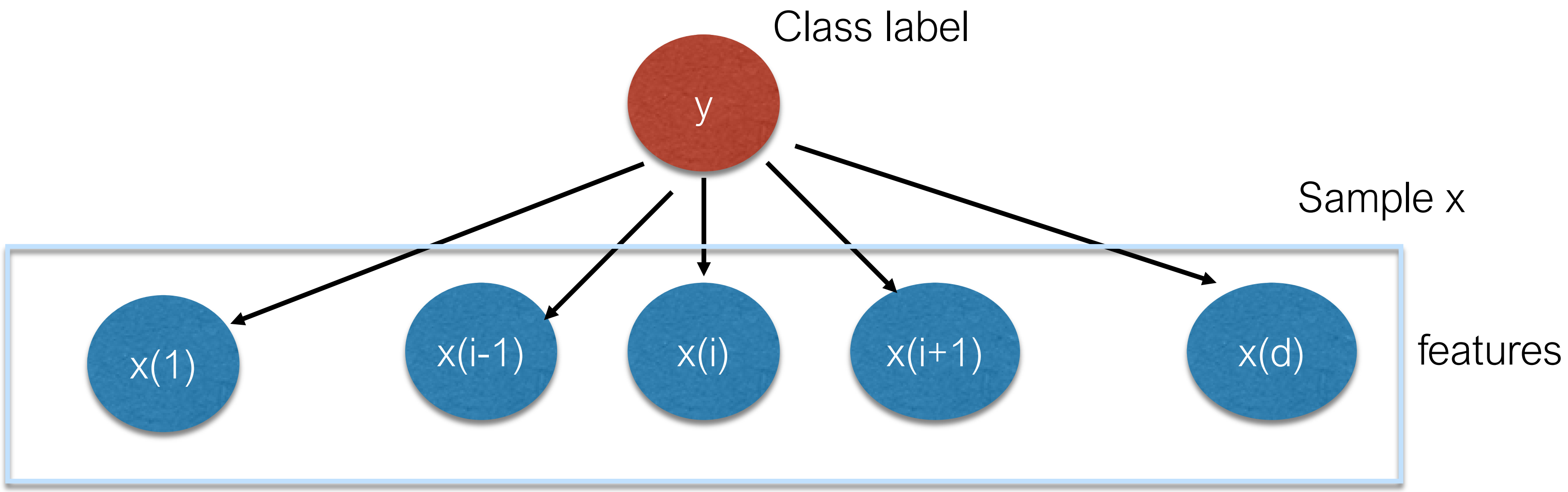
- For any test sample **x** (gene to be classified), find the K nearest samples (genes) in the training data.

- Classify the sample according to the majority class label of these neighbors.

Example:
- 2-dimensional feature vectors
- the red and blue points are the training samples, colored by class label (red = -, blue = +)
- Black point is the test sample
- K=3
- The K=3 nearest neighbors include two negatives and one positive; predict negative class



Naive Bayes Classifier



$$p(y|\boldsymbol{x}) = \frac{p(y)p(\boldsymbol{x}|y)}{p(\boldsymbol{x})}$$ Bayes rule

$$p(\boldsymbol{x}|y) = \prod_i p(x(i)|y)$$ Assume features independent

Choose y such that $p(y)\prod_i p(x(i)|y)$ is largest

What is P(x(i) | y)?

- From training data, compile all observed values of x(i), i.e., the ith feature, within class y.
- Fit a distribution to these values. Could be a Normal distribution, Poisson distribution, etc. This gives you P(x(i) | y) to use in the previous formula.
- This is how the computer “learns” from data

Linear Classifiers

$$f(x) = sign(w^T x + w_0)$$

- sign(z) is the “sign function”, = +1 if z is positive, = -1 if z is negative.

- *w* and w_0 are things (a vector and a scalar resp.) we have to find out: “free parameters”

Linear Classifiers

$$f(x) = sign(w^T x + w_0)$$

- Recall goal of training step: make correct predictions on all training examples: $f(\mathbf{x}_i) = -1$ for all i=1 … n and $f(\mathbf{x}_i) = +1$ for all i=n+1 … n+m

- We want to find w^T and w_0 such that $w^T x_i + w_0$ is < 0 for i = 1 .. n and > 0 for i = n+1 … n+m

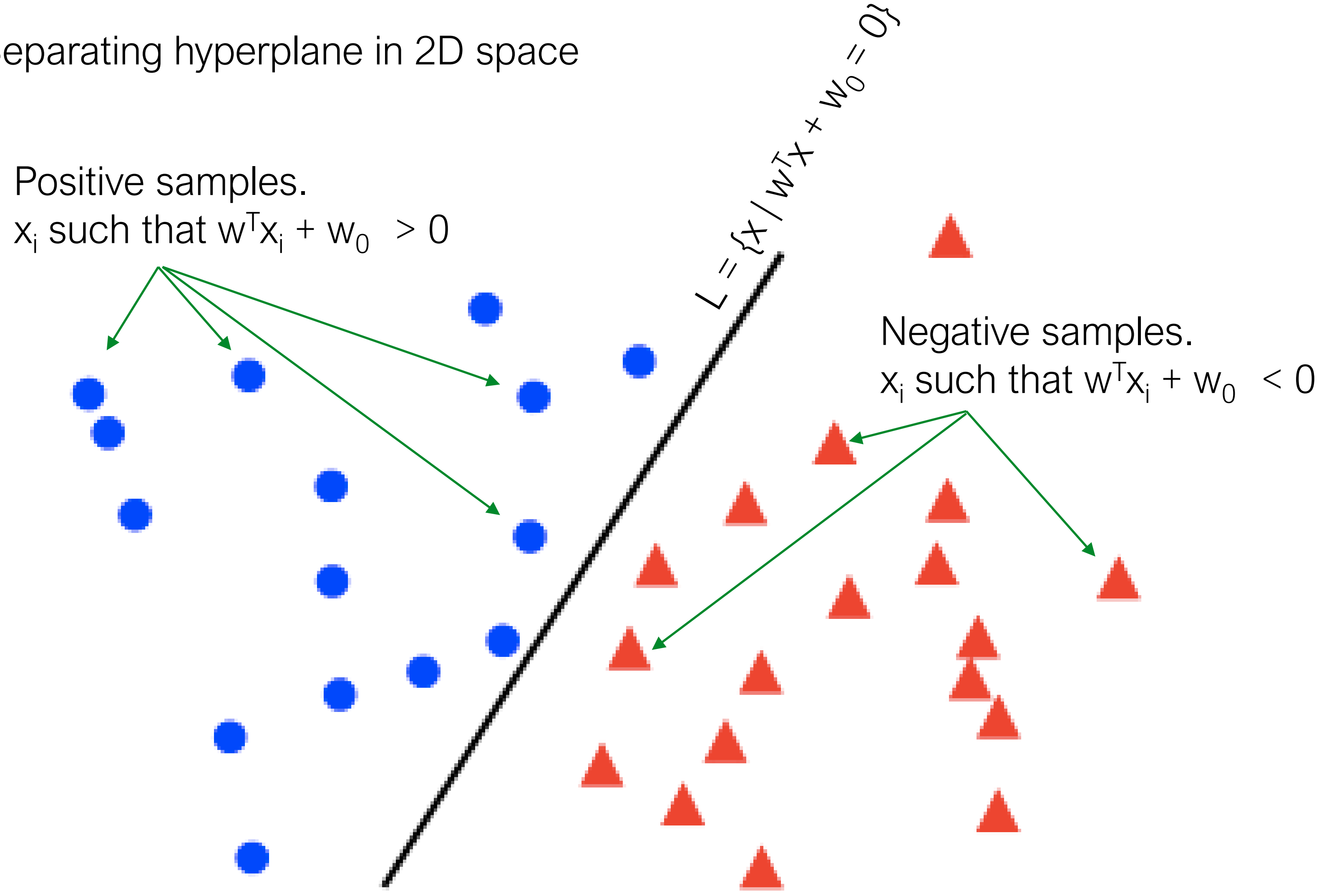
Linear Classifiers

$$f(x) = sign(w^T x + w_0)$$

- $L = \{x \mid w^T x + w_0 = 0\}$ is a "hyperplane".
- If input space is two-dimensional, the hyperplane is just a straight line; if input space is 3D, the hyperplane is a plane, etc.
- Points with $w^T x + w_0 < 0$ lie on one side of the hyperplane; points with $w^T x + w_0 > 0$ lie on the other side.
- Thus, $L = \{x \mid w^T x + w_0 = 0\}$ separates the positive samples from the negative samples. "Separating hyperplane"

Linear Classifiers

Separating hyperplane in 2D space


$L = \{x \mid w^T x + w_0 = 0\}$
Positive samples.
x_i such that $w^T x_i + w_0 > 0$
Negative samples.
x_i such that $w^T x_i + w_0 < 0$

Linear Classifiers

$$f(x) = sign(w^T x + w_0)$$

$$w^T x_i + w_0 < 0 \qquad \text{for } i \leq n$$

$$w^T x_i + w_0 > 0 \qquad \text{for } i > n$$

- Computer solves this system of inequalities to figure out values of *w* and $w_{0.}$ This is how it "learns" from training data.

Assessing classifier accuracy

Errors

- False positive: Predicted to be positive, but wrong! (Real label is negative)

- False negative: Predicted to be negative, but wrong! (Real label is positive)

Class confusion matrix

	Predicted Positive	Predicted Negative
Real positive	12	3
Real Negative	0	15

Accuracy measures

- Accuracy: Percentage of labels correctly classified. (27/30)
- Sensitivity: Percentage of real positives correctly classified. (12/15)
- Recall = Sensitivity
- Precision: Percentage of predicted positives correctly classified. (12/12)

	Predicted Positive	Predicted Negative
Real positive	12	3
Real Negative	0	15

Cross validation

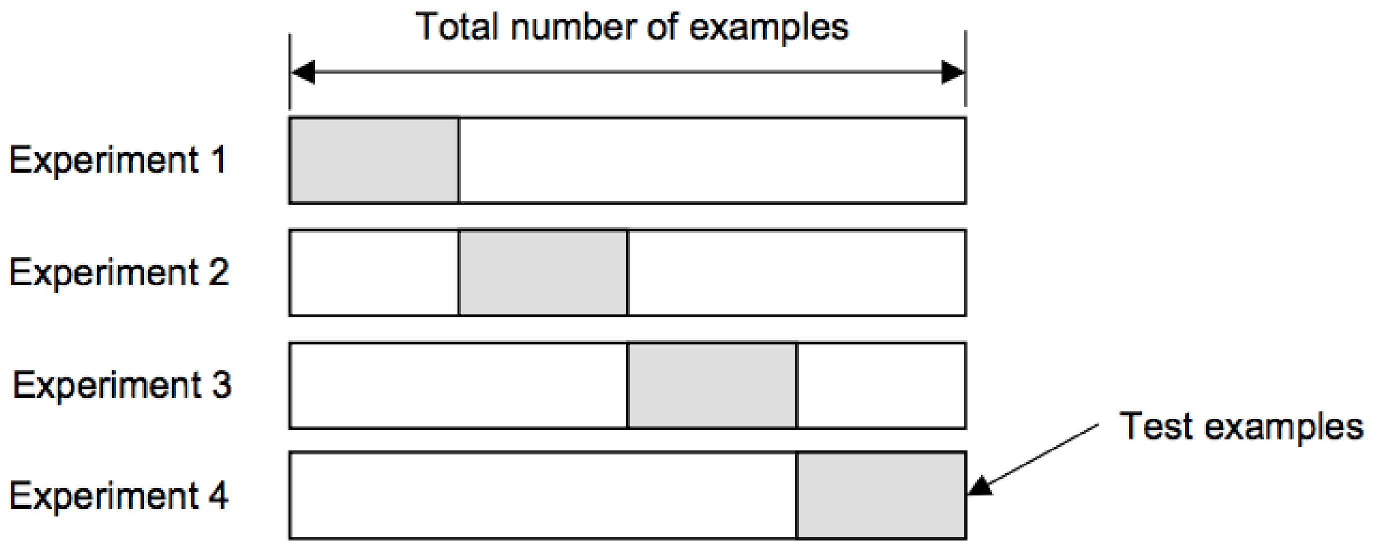
- A classifier is supposed to be trained on the training samples (e.g., genes with known functions) and used to make predictions on test data (genes with unknown function).

- How can we assess its predictions on the test data if we don’t know the truth about those genes?

- Solution: hide some of the labeled samples (genes with known functions) and make the classifier predict on those.

K-fold cross validation

- For each of K experiments, use K-1 folds for training and the remaining one for testing



Classification Error = Average classification error on K folds

Part 2: Regression

Classification vs Regression

"Classification"

Positive class

Negative class

Class Label y (Discrete)

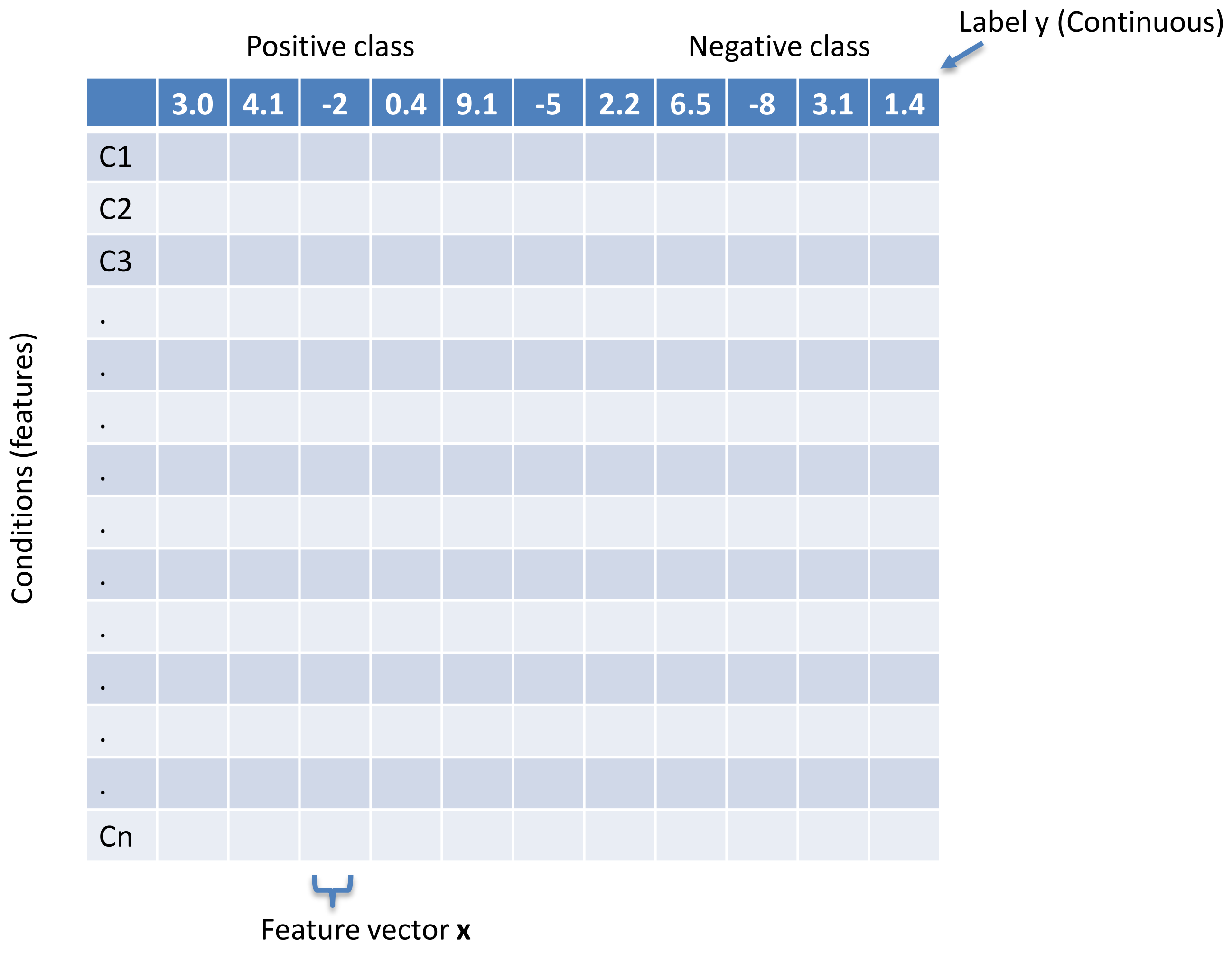
Conditions (features)

	+	+	+	+	+	-	-	-	-	-	-
C1											
C2											
C3											
.											
.											
.											
.											
.											
.											
.											
.											
.											
.											
Cn											

Feature vector **x**

Classification vs Regression

“Regression”



	3.0	4.1	-2	0.4	9.1	-5	2.2	6.5	-8	3.1	1.4
C1											
C2											
C3											
.											
.											
.											
.											
.											
.											
.											
.											
.											
.											
Cn											

Formally

- Each sample is "feature vector" $\mathbf{x_i}$.
- This is a *d*-dimensional vector, i.e., $\mathbf{x_i} = (x_{i1} , x_{i2} , \ldots , x_{id})$
- The features also go by other names: "independent variables", "predictor variables", "covariates", etc.
- Each sample also has a continuous valued "label" y_i . This sometimes goes by the name "response variable" or "dependent variable".
- Regression problem is to find a function $f(\boldsymbol{x})$ s.t. $f(\boldsymbol{x_i}) \approx y_i$ for all i

Nearest Neighbor Regression

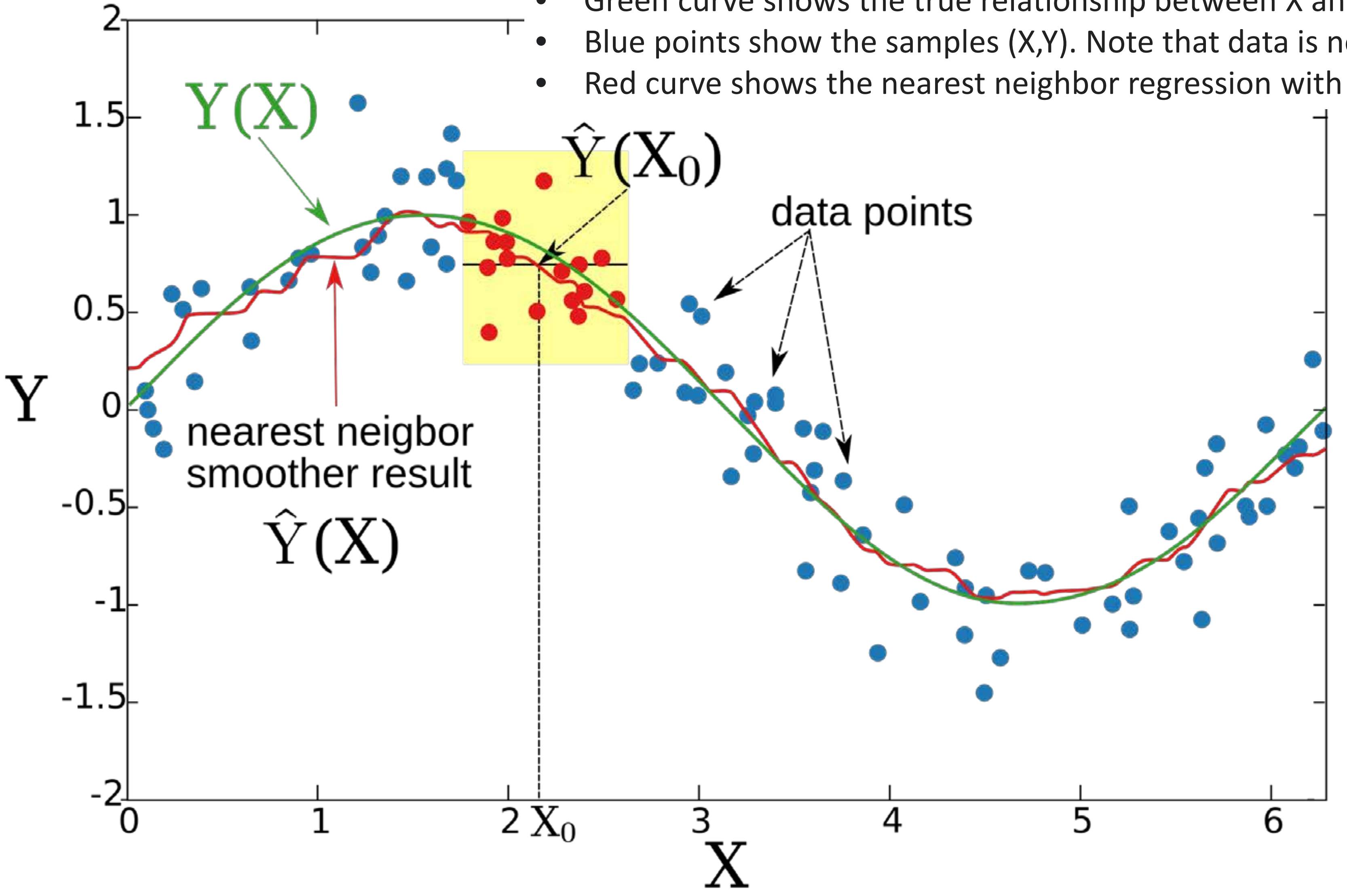
- Let **x** be a test sample.
- Collect the *K* nearest neighbors of **x** in the training data. Let the i^{th} of these be $\mathbf{x_i}$, with label y_i.
- One option is to define $f(\boldsymbol{x}) = \frac{\sum_i y_i}{K}$
- But some of the neighbors may be quite far from the test sample. We don't want them to contribute as much as the nearby samples. Hence, define

$$f(\boldsymbol{x}) = \frac{\sum_i w_i y_i}{\sum_i w_i}$$

w_i chosen to give greater weight to nearby samples. For instance, it could be set to inverse of distance from test sample.

Nearest neighbor regression

- In this example, *X* is one-dimensional.
- Green curve shows the true relationship between X and Y.
- Blue points show the samples (X,Y). Note that data is noisy
- Red curve shows the nearest neighbor regression with K=16



Source: wikipedia

Linear Regression

- Recall that we wish to find a function f(**x**) s.t. f($\mathbf{x}_i$) ≈ y_i for all i in the given data set.

- Let's say our function f(**x**) must be of the form $f(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{\beta}$, where $\boldsymbol{\beta}$ is a vector of numbers, of the same dimensionality as ***x***, and $\boldsymbol{x}^T \boldsymbol{\beta}$ refers to the dot product of ***x*** and $\boldsymbol{\beta}$. The $\boldsymbol{\beta}$ vector is also called the vector of "coefficients".

- Note that if $x = (x_1, x_2, \ldots, x_d)$ and $\beta = (\beta_1, \beta_2, \ldots, \beta_d)$ then
$$x^T \beta = \sum_{i=1}^{d} \beta_i x_i$$

- So f(**x**) is just the weighed sum of features, with $\boldsymbol{\beta}$ being the weights

Linear Regression

- Recall that we wish to find a function f(**x**) s.t. f($\mathbf{x}_i$) ≈ y_i for all i in the given data set.

- Let's say our function f(**x**) must be of the form $f(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{\beta}$, where $\boldsymbol{\beta}$ is a vector of numbers, of the same dimensionality as $\boldsymbol{x}$, and $\boldsymbol{x}^T\boldsymbol{\beta}$ refers to the dot product of $\boldsymbol{x}$ and $\boldsymbol{\beta}$. The $\boldsymbol{\beta}$ vector is also called the vector of "coefficients".

- One little problem though: This will force the zero vector $\boldsymbol{x} = (0,0,\ldots 0)$ to necessarily map to $y = 0$. We may not want that.

- So, we say that f(**x**) must be of the form $f(\boldsymbol{x}) = \beta_0 + \boldsymbol{x}^T\boldsymbol{\beta}$, where β_0 is a scalar.

Linear Regression

- $f(\boldsymbol{x}) = \beta_0 + \boldsymbol{x}^T\boldsymbol{\beta}$. How do we decide on values of $\boldsymbol{\beta}$ and β_0?

- We will set $\boldsymbol{\beta}$ and β_0 so that $f(\boldsymbol{x_i}) \approx y_i$ for all i in the given data set of $(\boldsymbol{x_i}, y_i)$. How to make this precise?

- We will make sure $\sum_i (f(\boldsymbol{x_i}) - y_i)^2$ is small. (This will be sort of like having $f(\boldsymbol{x_i}) \approx y_i$.)

- So we need to find values of $\boldsymbol{\beta}$ and β_0 that *minimize* $\sum_i (f(\boldsymbol{x_i}) - y_i)^2$. This is called "Least squares regression".

Residuals

- Note that for each sample $\boldsymbol{x_i}$, the model predicts the response variable (y) to be $f(\boldsymbol{x_i})$, while the true value of y is y_i. That is, it makes an error of $f(\boldsymbol{x_i}) - y_i$.

- This is called the "residual". It's the portion of y_i that the model couldn't explain!

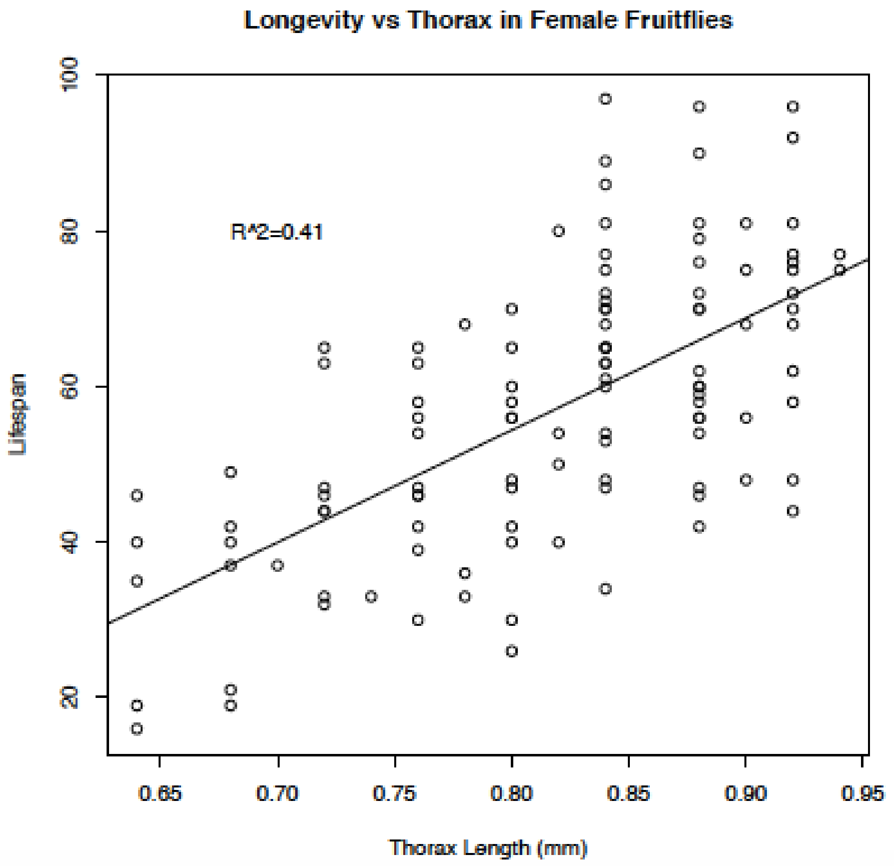
- In particular, the residual $\boldsymbol{e}$ is the vector of errors made on every sample, i.e., $\boldsymbol{e} = \{y_i - f(x_i)\}_i$.

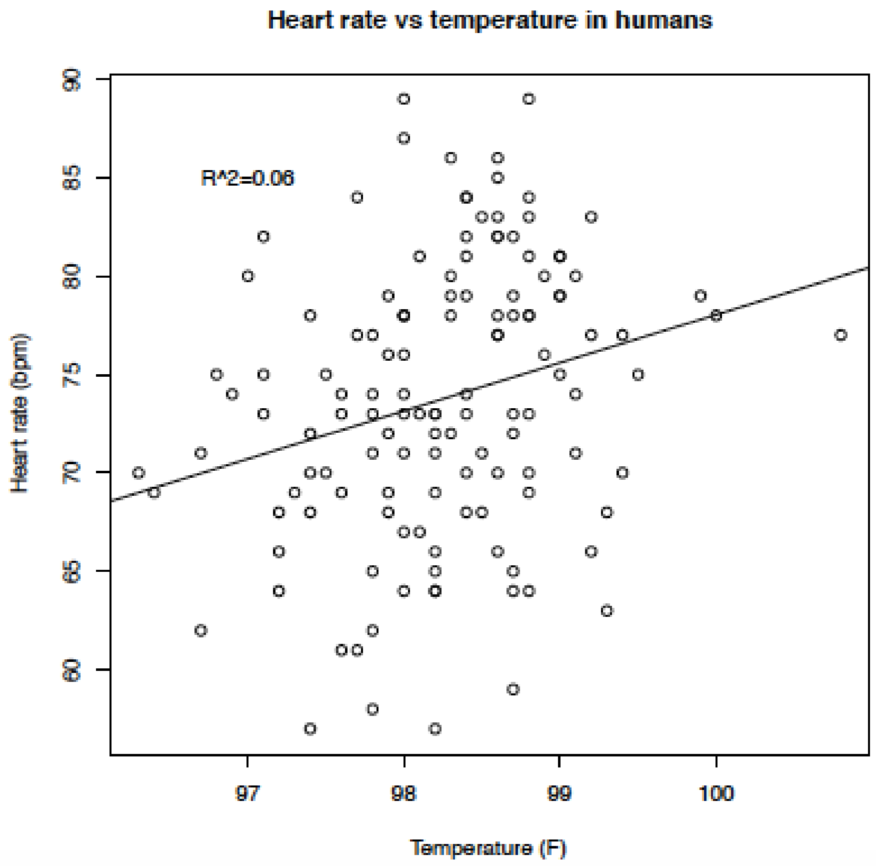
Assessing accuracy: “R-squared”

- We can show that $Var(y) = Var(x^T\beta) + Var(e)$
- The closer $\boldsymbol{e}$ is to the zero vector, $Var(e)$ is smaller, and $Var(x^T\beta)$ is closer to $Var(y)$. That is, our function $f(x) = x^T\beta$ “explains” more of the variance in y.
- This means we can think of regression as “explaining the variance of y”.
- We can formalize this idea by measuring the “goodness” of our regression model by $\frac{Var(x^T\beta)}{Var(y)}$
- This is called the “R^2” of the regression model. It is the fraction of the variance of y that is explained by the model $f(x) = x^T\beta$

R squared

$$R^2 = \frac{Var(x^T\beta)}{Var(y)}$$



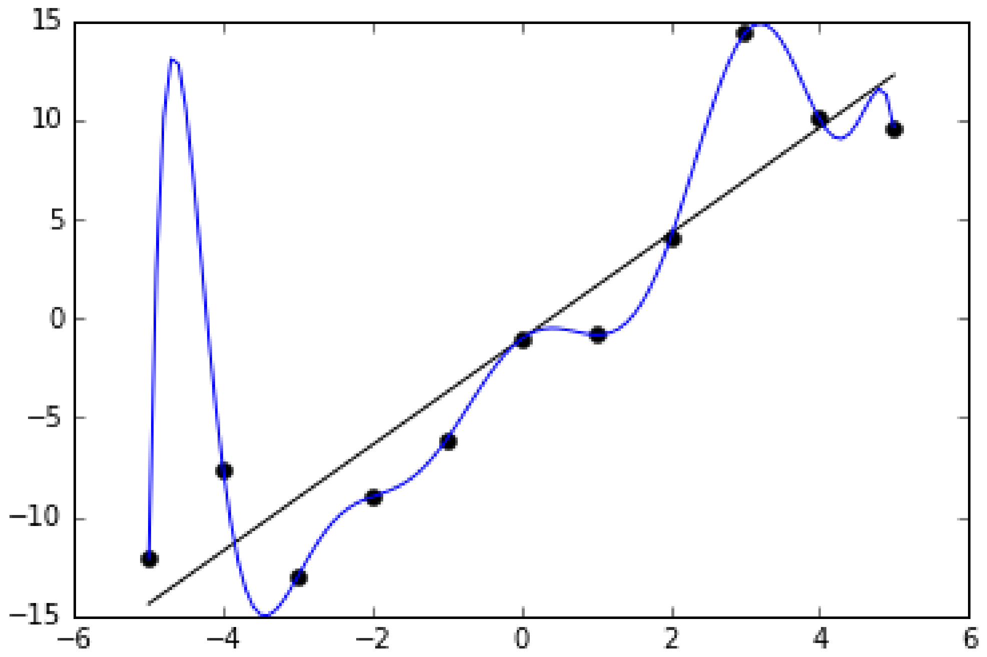


How much of the variance in y can be explained

Overfitting

- In least squares regression, we need to find values of $\boldsymbol{\beta}$ and β_0 that *minimize* $\sum_i (f(\boldsymbol{x_i}) - y_i)^2 = \sum_i \left(y_i - \beta_0 - x_i^T \beta\right)^2$
- Note that the prediction $f(\boldsymbol{x_i})$ includes the term $x_i^T \beta$, which is a weighted sum of the features describing $\boldsymbol{x_i}$
- If $\boldsymbol{x_i}$ is high dimensional, then there are a large number of weights β to be tuned. This often becomes a problem.
- Not just because there's more tuning to do, but also because the result of such extensive tuning of weights often end up being useless in practice.

Intuition



Source: https://en.wikipedia.org/wiki/Overfitting#/media/File:Overfitted_Data.png

- Tuning weights β in regression is akin to fitting a curve to a given set of points.
- Given the 11 points shown above, you could "fit" a straight line (black) that does not exactly go through all points but seems to get the "trend" right.
- Or you could fit a high-degree polynomial (blue) that does go through all the points.
- The straight line is a more robust fit, and is likely to be closer to the underlying truth.
- In fitting an unnecessarily complex model (high degree polynomial, blue), you probably overdid the fitting. Overfitting!

Addressing overfitting: Ridge regression

- In least squares regression, we need to find values of $\boldsymbol{\beta}$ and β_0 that *minimize* $\sum_i (f(\boldsymbol{x_i}) - y_i)^2 = \sum_i \left(y_i - \beta_0 - x_i^T \beta\right)^2$

- We want *not* to do extensive tuning of $\boldsymbol{\beta}$. One way to achieve this is to tie our hands when tuning $\boldsymbol{\beta}$.

- In particular, we decide to favor $\boldsymbol{\beta}$ with small weights and penalize $\boldsymbol{\beta}$ with large weights.

- For this, we find values of $\boldsymbol{\beta}$ and β_0 that *minimize*

$$\sum_{i=1} (y_i - x_i^T \beta)^2 + \lambda \sum_{j=1} \beta_j^2$$

The usual "least squares" term

A penalty term that favors small weights

Addressing overfitting: LASSO

- In least squares regression, we need to find values of $\boldsymbol{\beta}$ and β_0 that *minimize* $\sum_i (f(\boldsymbol{x_i}) - y_i)^2 = \sum_i \left(y_i - \beta_0 - x_i^T \beta\right)^2$

- We want *not* to do extensive tuning of $\boldsymbol{\beta}$. One way to achieve this is to tie our hands when tuning $\boldsymbol{\beta}$.

- In particular, we decide to favor $\boldsymbol{\beta}$ with few non-zero weights and penalize $\boldsymbol{\beta}$ with many non-zero weights.

- For this, we find values of $\boldsymbol{\beta}$ and β_0 that *minimize*

$$\|y - X\beta\|_2^2 + \lambda \sum_{j=1} |\beta_j|$$

The usual "least squares" term

A penalty term that favors setting most weights to zero

Some next steps

- Classification:
    - Support Vector Machines (linear classifier with some additional good properties)
    - Decision Trees and Random Forests (non-linear)
    - Artificial Neural Networks (non-linear)

- Regression:
    - All of the above classification methods have regression counterparts too.