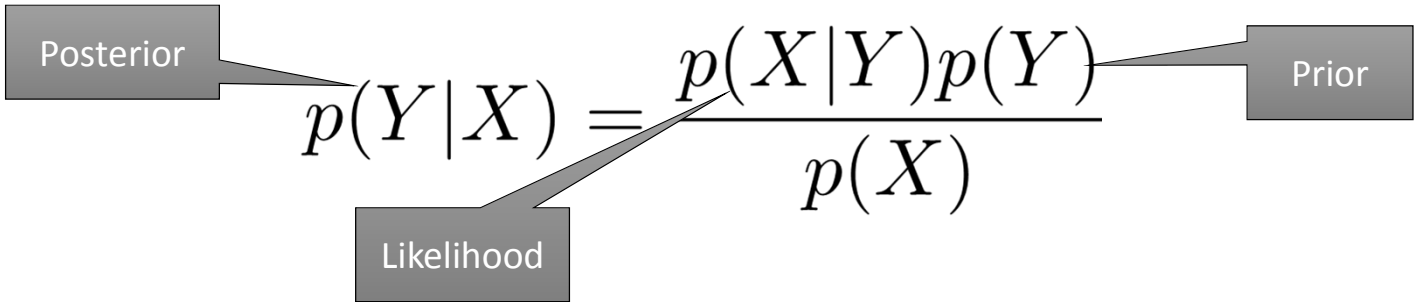
Introduction to Machine Learning and Data Mining Lecture-3: Scikit-learn, Regression and Decision Trees

Prof. Eugene Chang

Today

- Finish Math Primer
- Python tools
  - Scikit-learn
- Regression
  - Linear and Logistic
- Decision Trees
- Homework #1

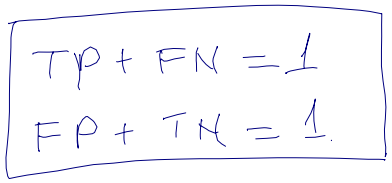
Interpretation of Bayes Rule

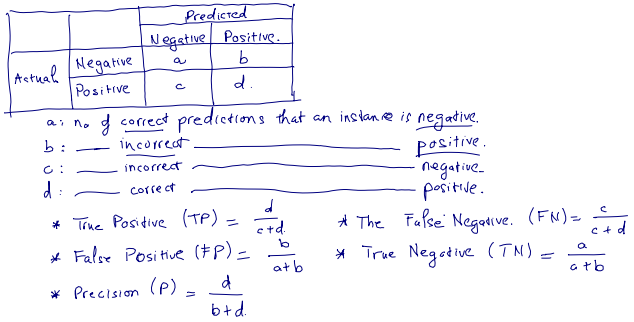


- **Prior**: Information we have before observation.
- **Posterior**: The distribution of Y after observing X
- **Likelihood:** The likelihood of observing X given Y

$$p(y|x) = \frac{P(x|y)\,P(y)}{P(x)}.$$

Binary Classification Performance





- Conditional probability
  - P(feature | class-1)
- Sensitivity
  - P(positive | class-1): True Positive (TP) rate or recall rate
  - measures the proportion of positives which are correctly identified
  - False Negative (FN) P(~positive | class-1) = 1-TP
- Specificity
  - P(negative | ~class-1): True negative rate (TN)
  - measures the proportion of negatives which are correctly identified
  - False Positive (FP) P(positive | ~class-1) = 1-TN

Cancer Screening Test

- A priori: In America, 10% women may develop breast cancer in her lifetime
  - P(Can) = 0.1
  - P(~Can) = 0.9
- Screening test sensitivity
  - TP = P(positive | Can) = 0.8
  - FN = P(negative | Can) = 0.2
- Screening test specificity
  - TN = P(negative | ~Can) = 0.7
  - FP = P(positive | ~Can) = 0.3
- What is the posterior probability?
  - P(positive) = P(positive | Can)*P(Can) + P(positive | ~Can) * P(~Can) = 0.8 * 0.1 + 0.3 * 0.9 = 0.35
  - p(Can | positive) = P(positive |Can) P(Can) / P(positive) = (0.8*0.1) / 0.35 = 0.2286

Spam Detection

- A priori: In general, 10% emails are spam
  - P(Spam) = 0.1
  - P(~Spam) = 0.9
- Spam detection based on length (< 100 words)
  - Sensitivity p(positive | Spam) = 0.8
  - Length test specificity
    - P(negative | ~Spam) = 0.7
    - P(positive | ~Spam) = 0.3
- What is the posterior probability?
  - P(positive) = P(positive | Spam)*P(Spam) + P(positive | ~Spam) * P(~Spam) = 0.8 * 0.1 + 0.3 * 0.9 = 0.35
  - P(Spam | positive) = P(positive |Spam) P(Spam) / P(positive) = (0.8*0.1) / 0.35 = 0.2286

$$P(S|P) = \frac{P(P|S) \times P(S)}{P(P)}$$

(TP)

(FN) Negative | Spam) = 0.2

(TN)

(FP)

Spam Detection #2

- A priori: In general, 10% emails are spam
  - P(Spam) = 0.2
  - P(~Spam) = 0.8
- Spam detection based on length (< 100 words)
  - Sensitivity P(< 100| Spam) = 0.8
  - Specificity
    - P(>= 100| ~Spam) = 0.7
    - P(< 100| ~Spam) = 0.3
- Spam detection based on keyword "hello"
  - Sensitivity P("hello"| Spam) = 0.6
  - Specificity
    - P(~"hello"| ~Spam) = 0.6
    - P("hello"| ~Spam) = 0.4

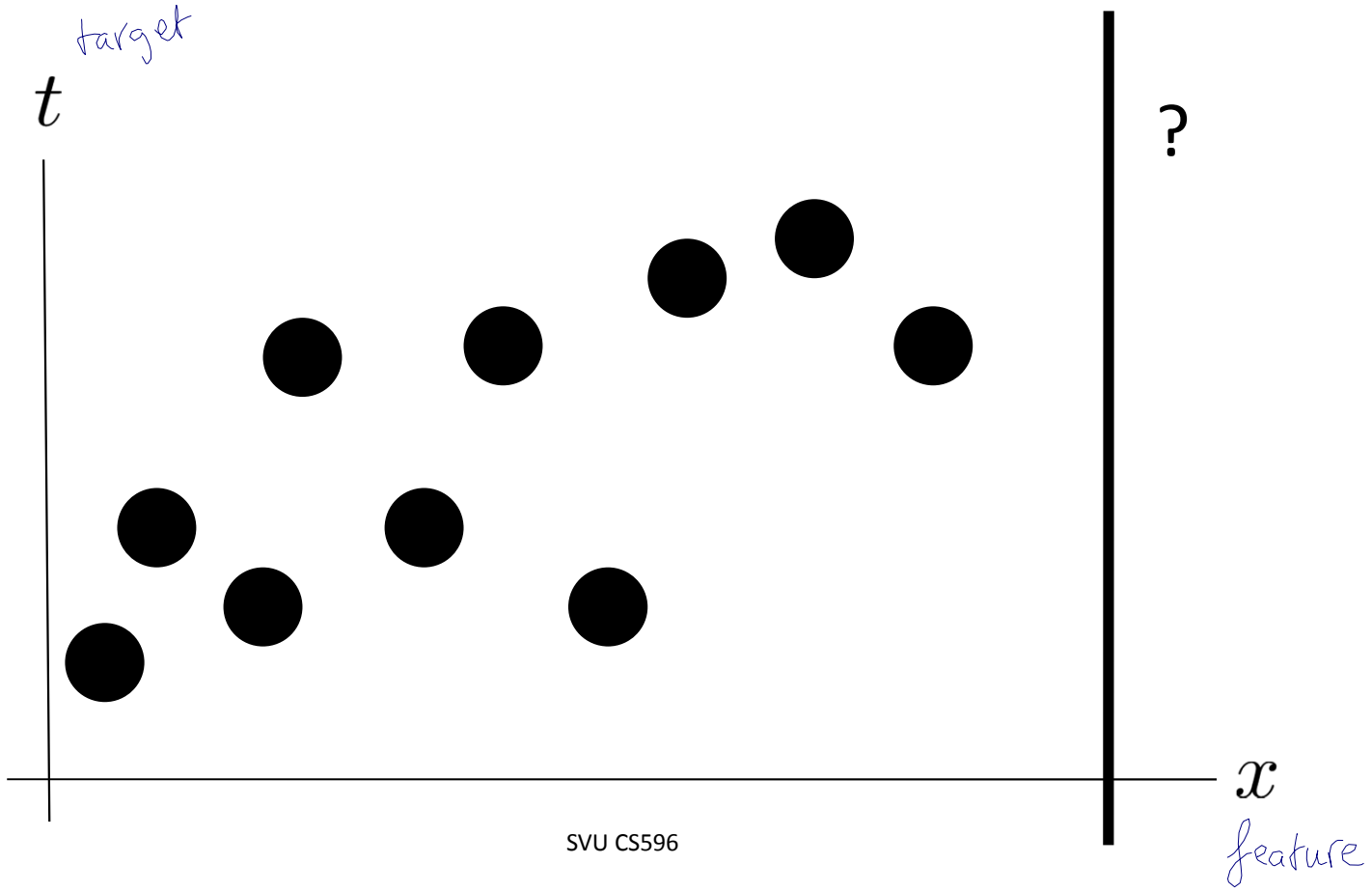
Linear Regression

Basics of Linear Regression

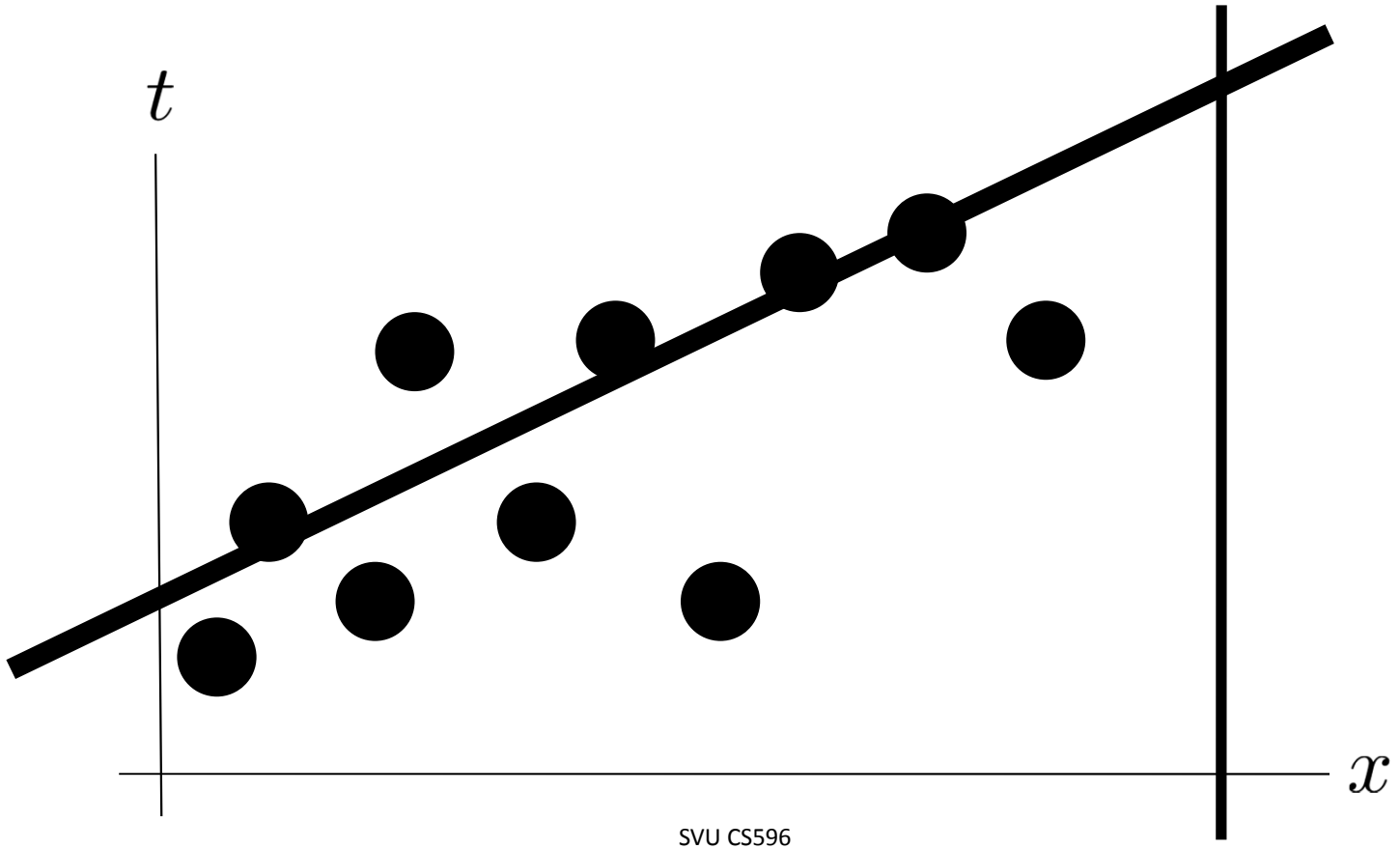
- **Regression** algorithm
- **Supervised** technique.
- In one dimension: $y : \mathbb{R} \rightarrow \mathbb{R}$
  - Identify
- In D-dimensions: $y : \mathbb{R}^D \rightarrow \mathbb{R}$ ← Target still 1-D
  - Identify
- Given: training data: $\{\vec{x}_0, \vec{x}_1, \ldots, \vec{x}_N\}$
  - And targets:

$$\{t_0, t_1, \ldots, t_N\}$$

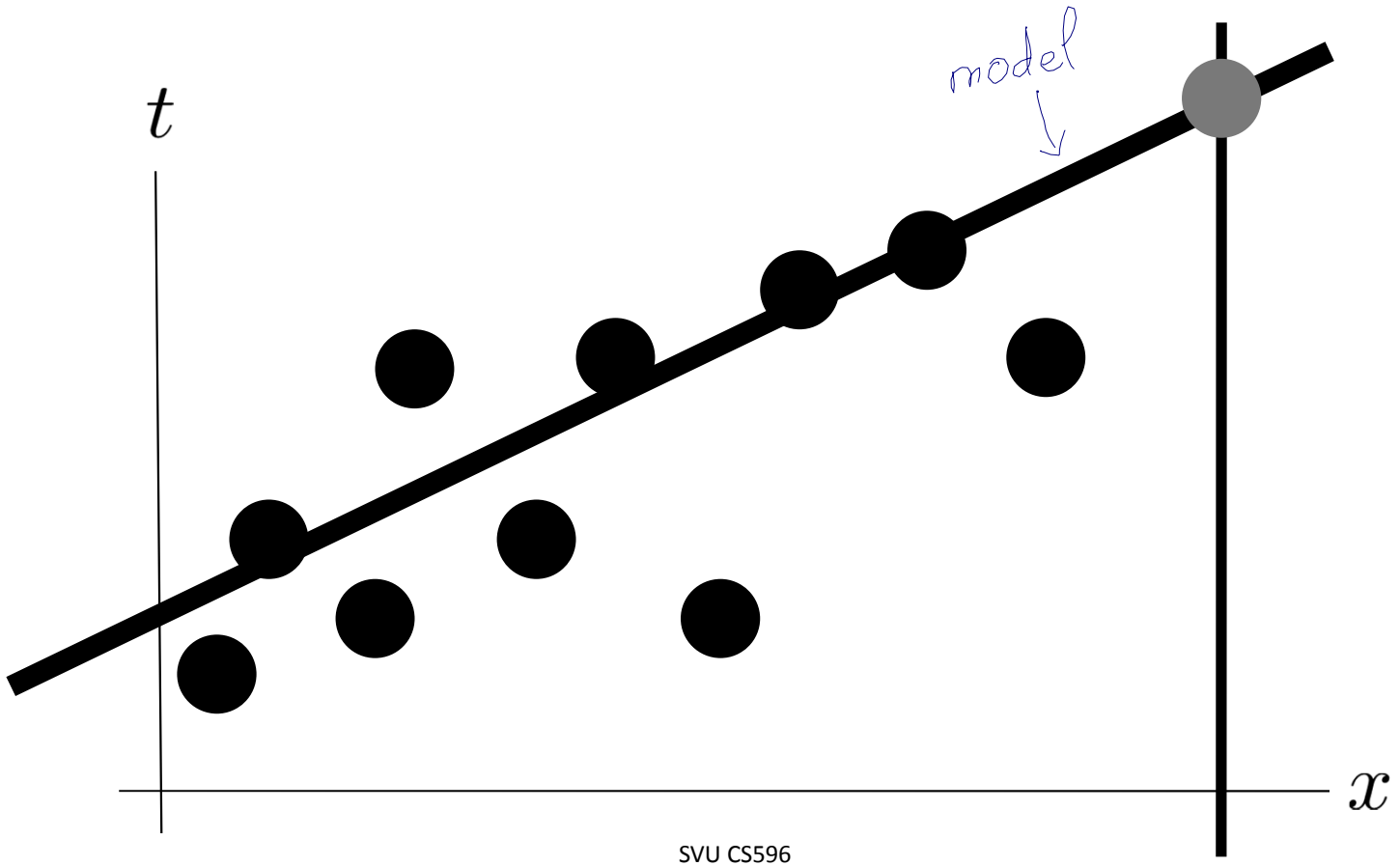
Graphical Example of Regression



Graphical Example of Regression



Graphical Example of Regression



Definition

no sqrt, no sin, cosin etc. only + */

- In **linear regression**, we assume that the model that generates the data involved **only** a linear combination of input variables.

$$y(\vec{x}, \vec{w}) = w_0 + w_1 x_1 + \ldots + w_D x_D$$

$$y(\vec{x}, \vec{w}) = w_0 + \sum_{j=1}^{D} w_j x_j$$

Where **w** is a vector of weights which define the D parameters of the model

Evaluation

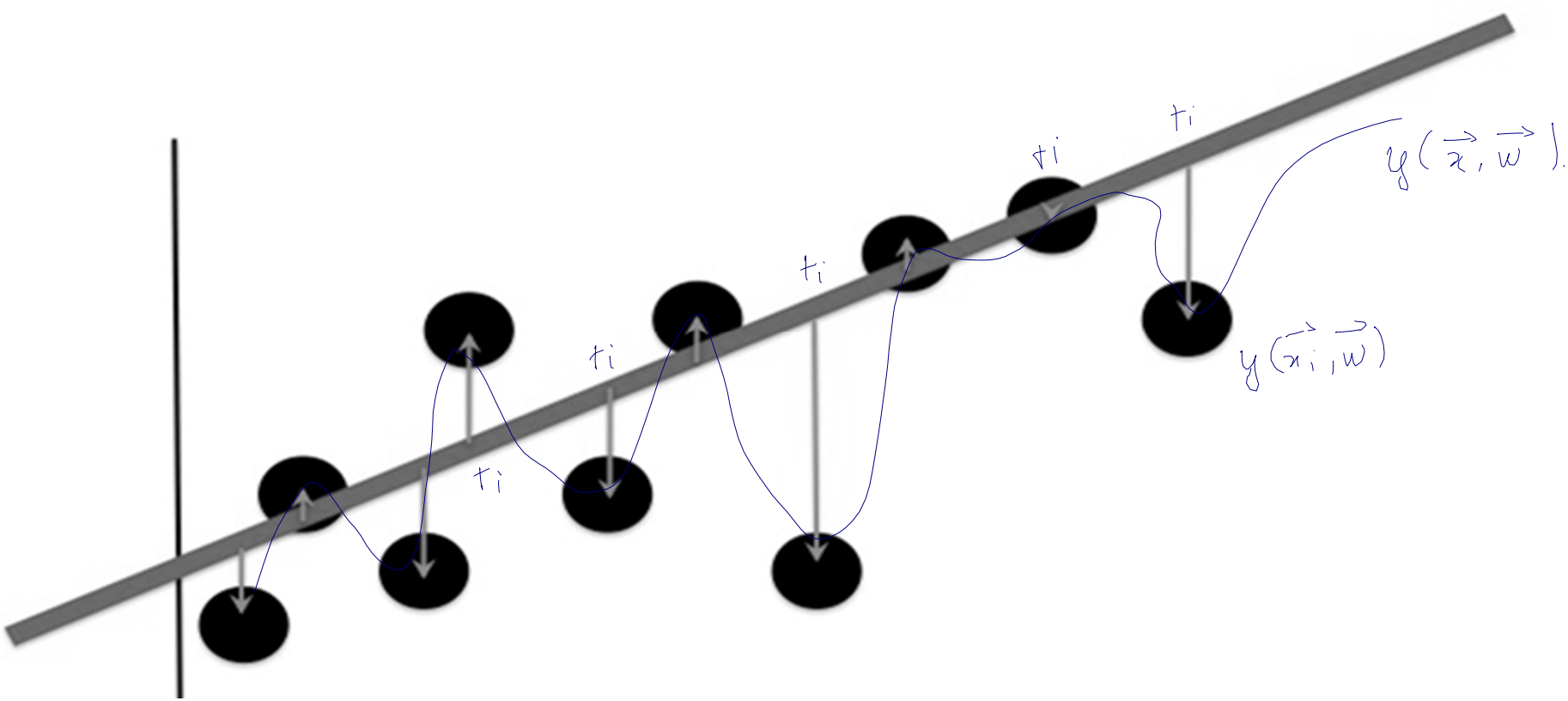
- How can we evaluate the performance of a regression solution?
- Error Functions (or Loss functions)
  - Squared Error

$$E(t_i, y(\vec{x}_i, \vec{w})) = \frac{1}{2}(t_i - y(\vec{x}_i, \vec{w}))^2$$

  - Linear Error

$$E(t_i, y(\vec{x}_i, \vec{w})) = |t_i - y(\vec{x}_i, \vec{w})|$$

Regression Error



How do we "learn" parameters

- For the 2-*d* problem (line) there are coefficients for the bias and the independent variable (*y*-intercept and slope)

$$Y = w_0 + w_1 X$$

- To find the values for the coefficients which minimize the objective function we take the partial derivates of the objective function (SSE) with respect to the coefficients. Set these to 0, and solve.

$$w_1 = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - \left(\sum x\right)^2}$$

$$w_0 = \frac{\sum y - w_1 \sum x}{n}$$

Least Square Solution Examples

Y	3	5	7
X	1	2	3

Solution: $w_1 = \frac{n\Sigma xy - \Sigma x \Sigma y}{n\Sigma x^2 - (\Sigma x)^2} = \frac{3 + 10 + 21 - 6 \times 15}{1+4+9 - 36}$

$= \frac{-56}{-22} = 2.545$

$Y = 1 + 2X$

not match

Y	3	6	6
X	1	2	3

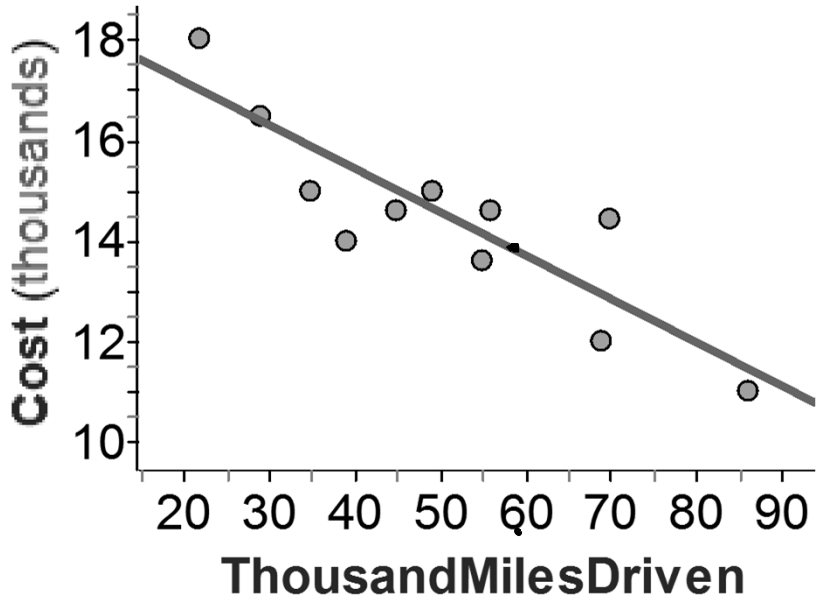
Solution: $w_1 = \frac{\sum xy - \sum x \sum y}{\sum x^2 - (\sum x)^2} = \frac{(3+12+18) - 6 \times 15.}{(1+4+9) - 36} = \frac{-57}{-22} = 2.59$

$Y = 2 + 1.5X$

not match

The following data shows the number of miles driven and advertised price for 11 used Honda CR-Vs from the 2002-2006 model years (prices found at www.carmax.com). The scatterplot below shows a strong, negative linear association between number of miles and advertised cost. The correlation is -0.874. The line on the plot is the regression line for predicting advertised price based on number of miles.

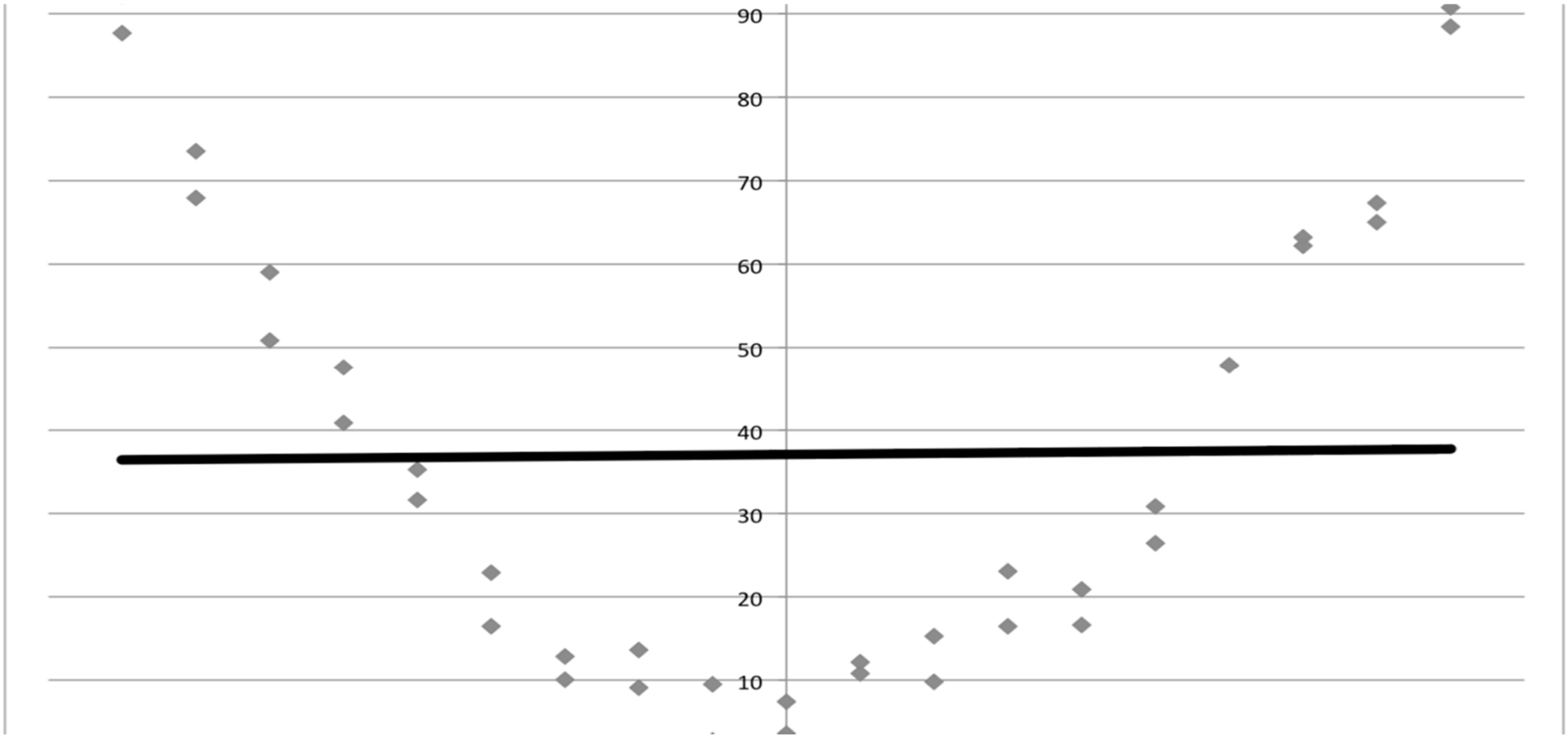
Thousand Miles Driven	Cost (dollars)
22	17998
29	16450
35	14998
39	13998
45	14599
49	14988
55	13599
56	14599
69	11998
70	14450
86	10998



Fat vs Calories in Burgers

Fat (g)	Calories
19	410
31	580
34	590
35	570
39	640
39	680
43	660

Extension to polynomial regression



Extension to polynomial regression

$$y = c_0 + c_1 x_1 + c_2 x_2$$

$$y = c_0 + c_1 x + c_2 x^2$$

- Polynomial regression is the same as linear regression in D dimensions

Generate new features

Standard Polynomial with coefficients, **w**

$$y(x, \vec{w}) = \sum_{d=1}^{D} w_d x^d + w_0$$

Risk

$$R = \frac{1}{2} \left\| \begin{bmatrix} t_0 \\ t_1 \\ \vdots \\ t_{n-1} \end{bmatrix} - \begin{bmatrix} 1 & x_0 & \dots & x_0^p \\ 1 & x_1 & \dots & x_1^p \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n-1} & \dots & x_{n-1}^p \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_p \end{bmatrix} \right\|^2$$

Generate new features

Feature Trick: To fit a *D* dimensional polynomial, Create a *D*-element vector from x_i

$$\vec{x}_i = \left[\begin{array}{cccc} x_i^0 & x_i^1 & \ldots & x_i^P \end{array} \right]^T$$

Then standard linear regression in *D* dimensions

How is this still linear regression?

- The regression is **linear** in the parameters, despite projecting x_i from one dimension to *D* dimensions.
- Now we fit a plane (or hyperplane) to a representation of x_i in a higher dimensional feature space.
- This generalizes to any set of functions

$$\phi_i : \mathbb{R} \rightarrow \mathbb{R}$$

$$\vec{x}_i = \begin{bmatrix} \phi_0(x_i) & \phi_1(x_i) & \dots & \phi_P(x_i) \end{bmatrix}^T$$

Scikit-learn: Machine learning in Python

Brian Holt

Project goals and vision

- Machine learning for applications
  - Ease of use
  - Light and easy to install package
  - A general-purpose high level language: Python
- High standards
  - State-of-the-art algorithms
  - High quality bindings: performance and fine control
- Open Source
  - BSD license
  - Community driven
- http://scikit-learn.org/stable/

API and design 1

- Design principles
  - Minimise number of object interfaces
  - Build abstractions for recurrant usecases
  - Simplicity, Simplicity, Simplicity
    - (no framework, no pipelines, no dataset objects)
- Code sample

```
>>> from sklearn import svm
>>> classifier = svm.SVC()
>>> classifier.fit(X_train, y_train)
>>> y_pred = clf.predict(X_test)
```

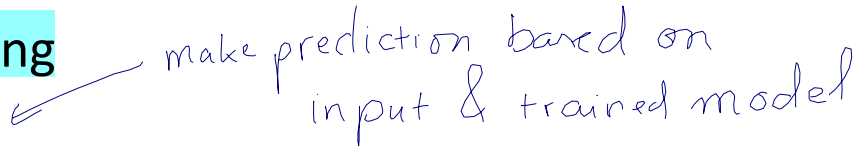
supervised learning

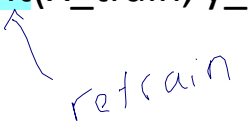
(training)

(calculate result)

API and design 2

- All objects
  - estimator.fit(X_train, y_train)
- Classification, regression, clustering
  - y_pred = estimator.predict(X_test)
- Filters, dimension reduction, latent variables
  - X_new = estimator.transform(X_test)
- Predictive models, density estimation
  - test_score = estimator.score(X_test)
- One day: On-line learning
  - estimator.refit(X_train, y_train)





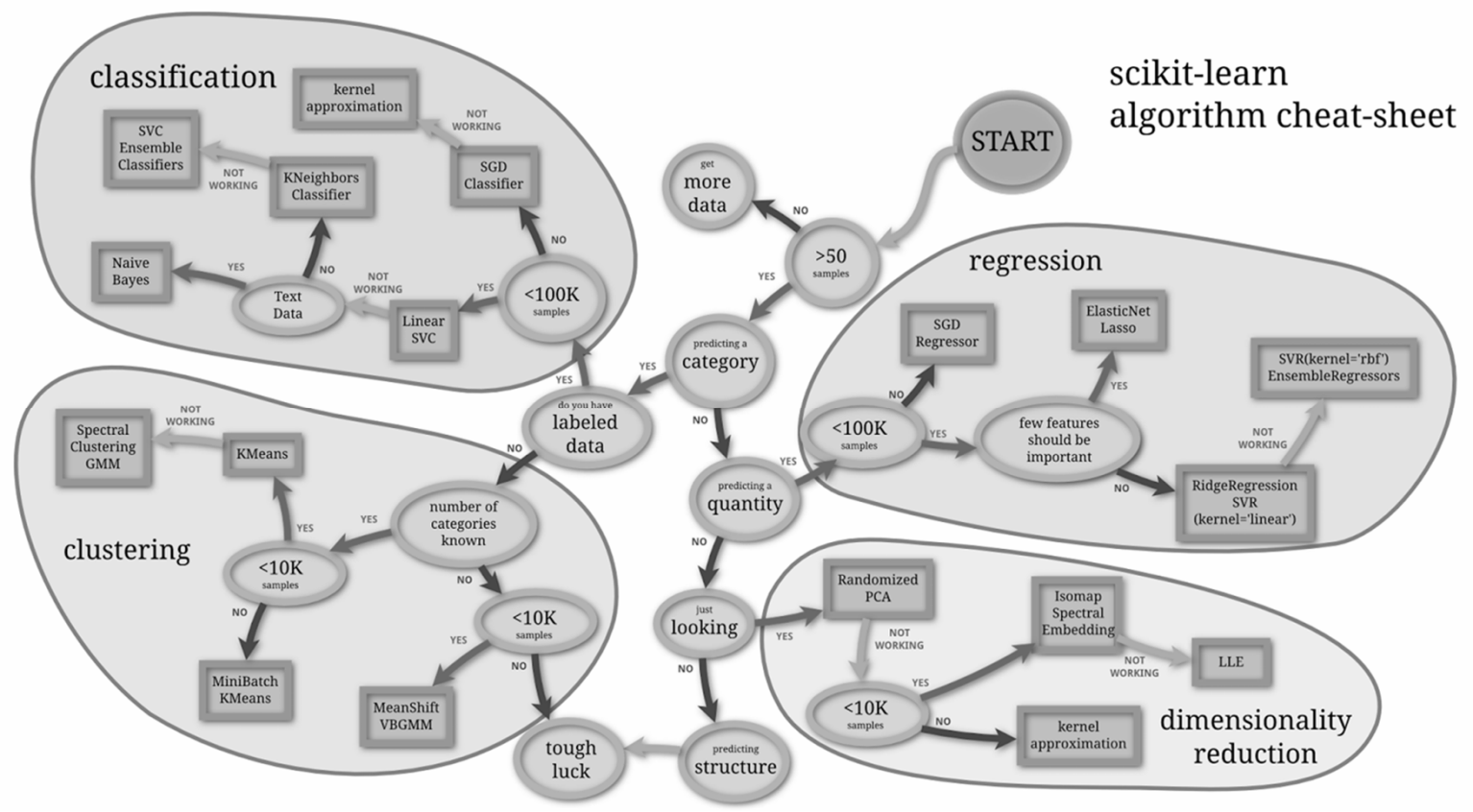
Main features and algorithms

- Supervised learning (classification and regression)
  - SVM - high quality libsvm bindings
  - Generalised linear models
    - Least squares, ridge regression, Orthogonal matching pursuit...
  - Nearest Neighbours
    - KDTree, BallTree
  - Gaussian Processes
  - Decision tree (CART)
  - Ensembles (boosting, bagging)

Main features and algorithms cont.

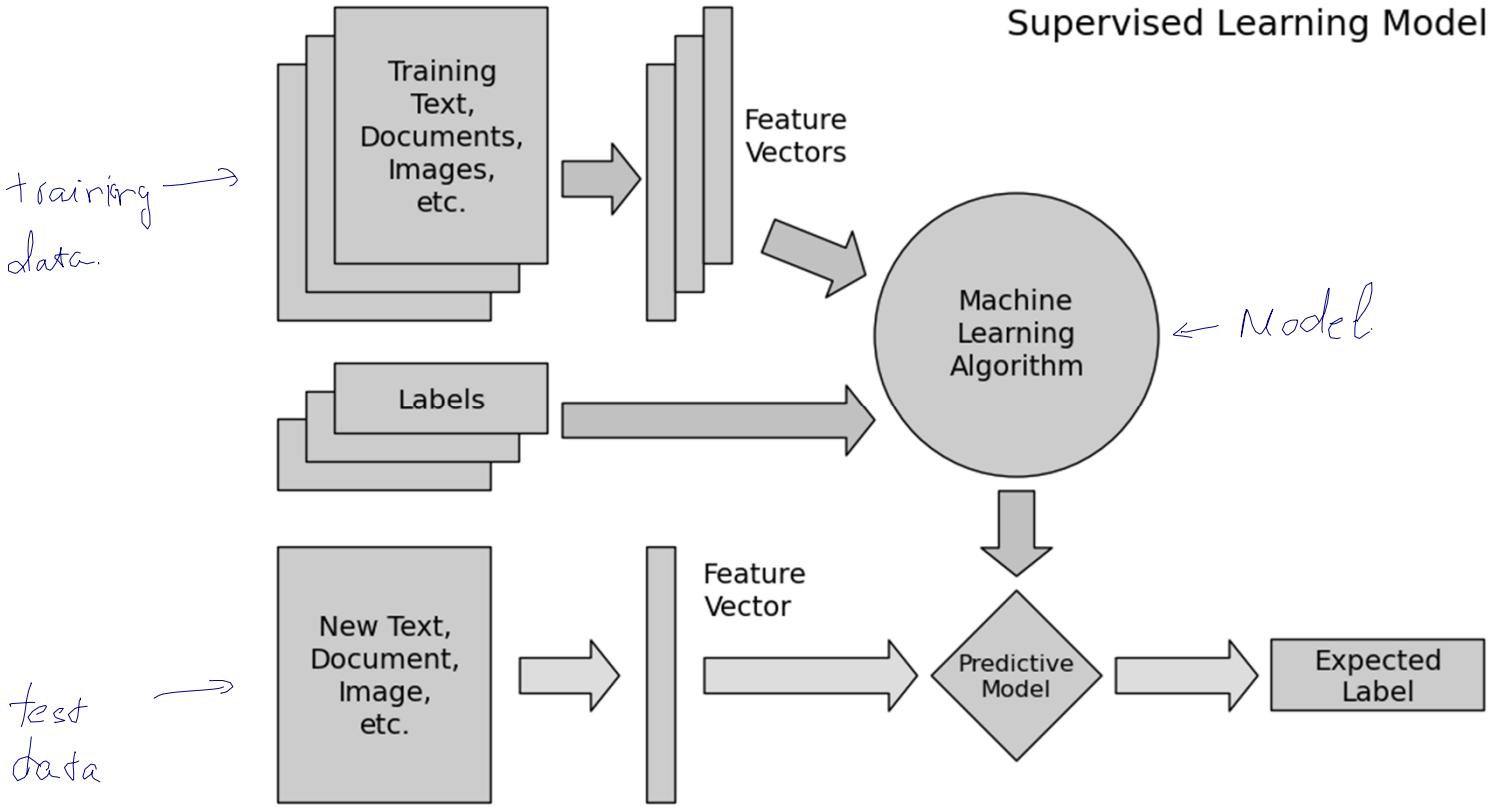
- Unsupervised learning
  - Gaussian mixture models
  - Manifold learning
  - Clustering
- Model selection
  - Cross-Validation
  - Grid Search
- Dimensionality Reduction
- Preprocessing

Cheat Sheet

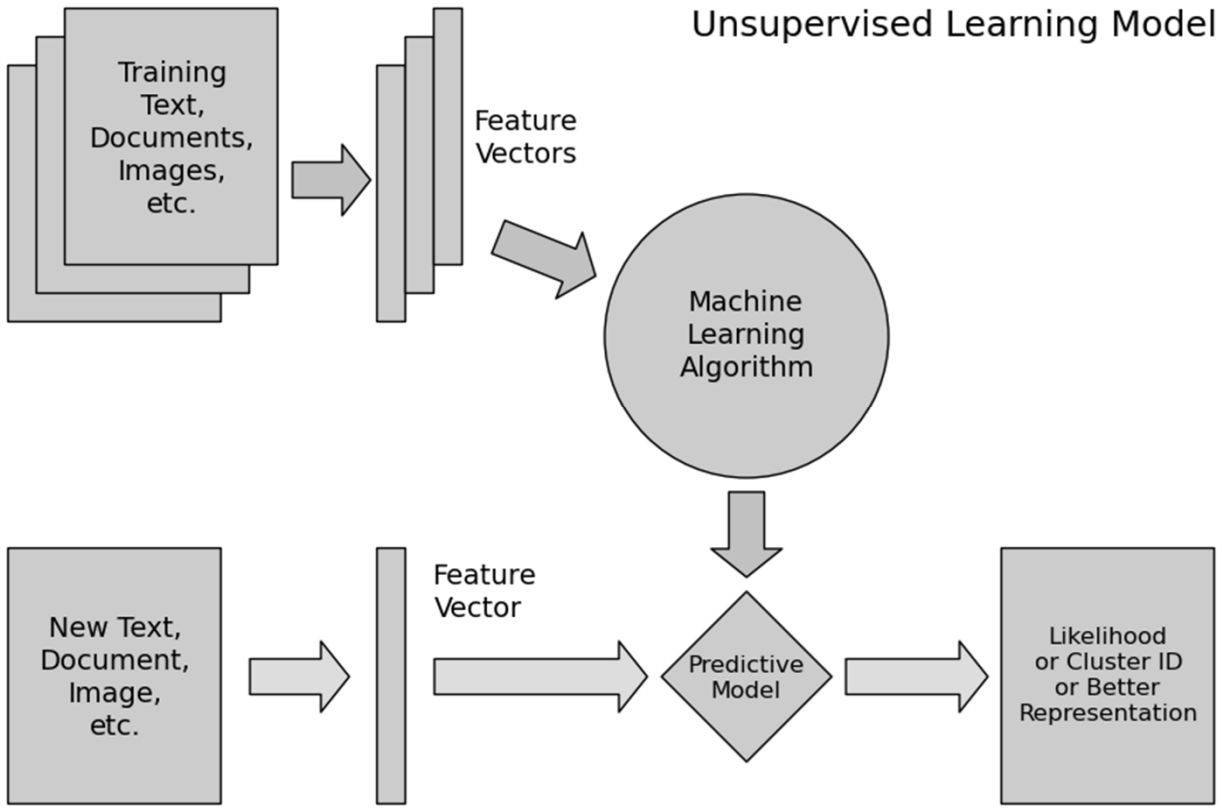

scikit-learn
algorithm cheat-sheet
START
get more data
>50 samples
predicting a category
do you have labeled data
predicting a quantity
just looking
predicting structure
tough luck
classification
kernel approximation
SVC Ensemble Classifiers
KNeighbors Classifier
SGD Classifier
Naive Bayes
Text Data
Linear SVC
<100K samples
clustering
Spectral Clustering GMM
KMeans
number of categories known
<10K samples
MiniBatch KMeans
MeanShift VBGMM
regression
SGD Regressor
ElasticNet Lasso
SVR(kernel='rbf') EnsembleRegressors
<100K samples
few features should be important
RidgeRegression SVR (kernel='linear')
dimensionality reduction
Randomized PCA
Isomap Spectral Embedding
LLE
<10K samples
kernel approximation
YES
NO
NOT WORKING

we have training data & the corresponding target.

Supervised Learning Model



Unsupervised Learning Model



Training data vs. Test Data

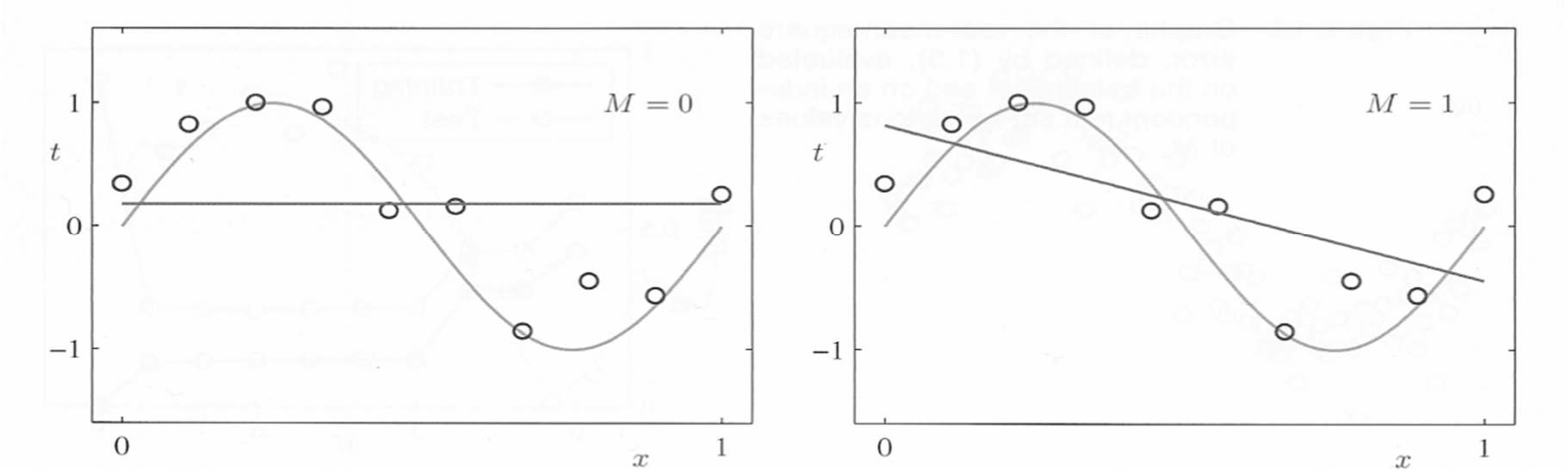
- Evaluating the performance of a classifier on training data is **meaningless**.
- With enough parameters, a model can simply memorize (**encode**) every training point
- To evaluate performance, data is divided into **training** and **testing** (or **evaluation**) data.
  - **Training** data is used to learn model parameters
  - **Testing** data is used to evaluate performance

Examples

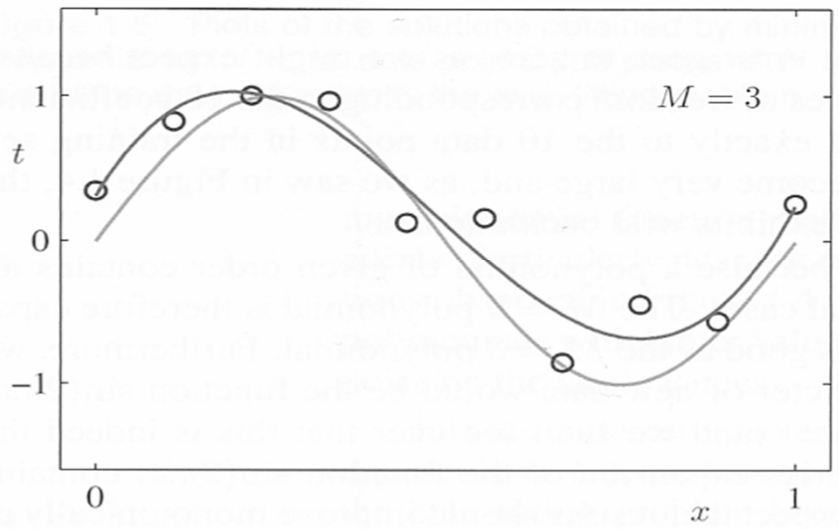
- Plot_ols
- Plot_ransac

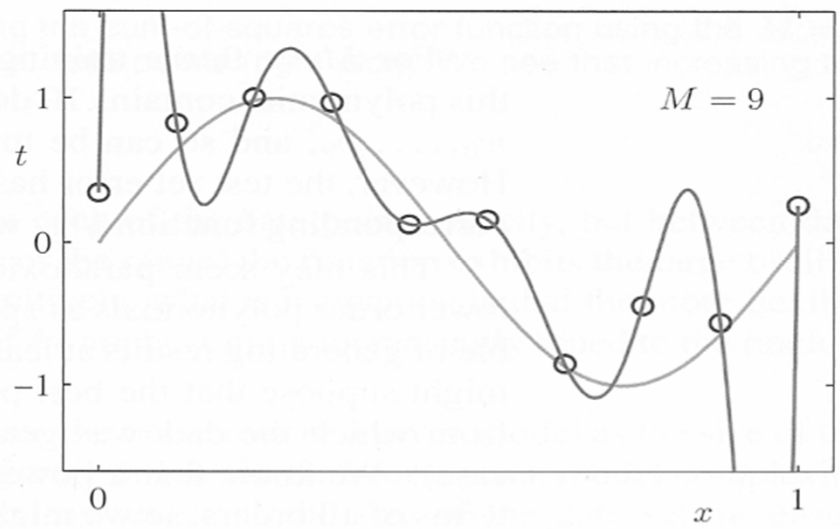
Linear Regression Continued

Overfitting

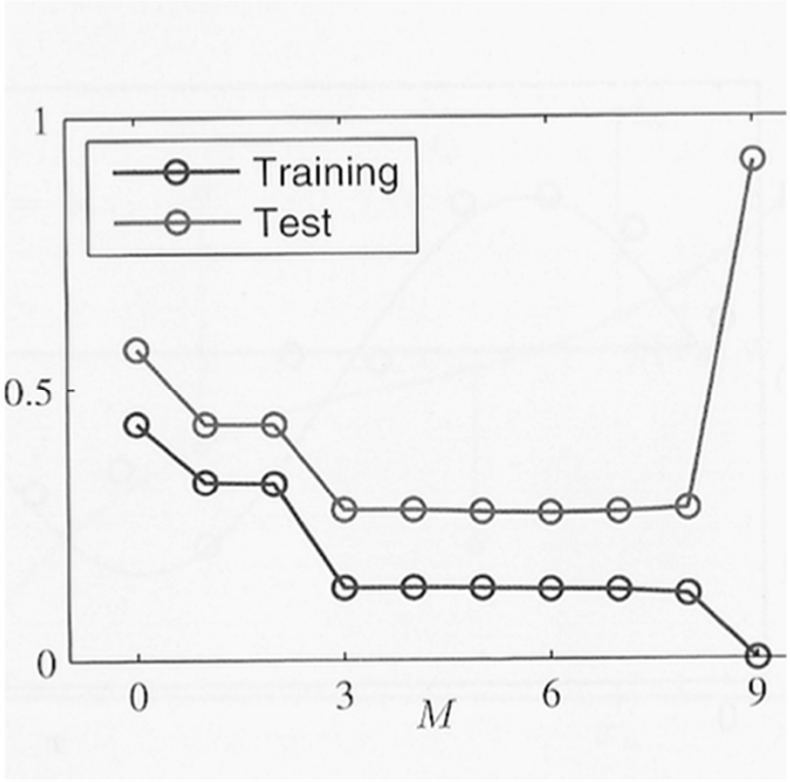


Overfitting





Overfitting performance



Definition of overfitting

- When the model describes the noise, rather than the signal.

- How can you tell the difference between overfitting, and a bad model?

Possible detection of overfitting

- Stability
  - An appropriately fit model is stable under different samples of the training data
  - An overfit model generates inconsistent performance
- Performance
  - A good model has low test error
  - A bad model has high test error

What is the optimal model size?

- The **best** model size generalizes to unseen data the best.
- Approximate this by testing error.
- One way to optimize parameters is to minimize testing error.
  - This operation uses testing data as **tuning** or **development** data
- Sacrifices training data in favor of parameter optimization
- Can we do this without *explicit* evaluation data?

Dealing with Overfitting

- Use more data
- Use a tuning set
- **Regularization**
- Be a Bayesian

Regularization

- In a linear regression model overfitting is characterized by large weights.

large weights to avoid.

	M = 0	M = 1	M = 3	M = 9
w_0	0.19	0.82	0.31	0.35
w_1		-1.27	7.99	232.37
w_2			-25.43	-5321.83
w_3			17.37	48568.31
w_4				-231639.30
w_5				640042.26
w_6				-1061800.52
w_7				1042400.18
w_8				-557682.99
w_9				125201.43

Penalize large weights

- Introduce a penalty term in the loss function.

$$E(\vec{w}) = \frac{1}{2}\sum_{n=0}^{N-1}\{t_n - y(x_n, \vec{w})\}^2$$

Regularized Regression
(L2-Regularization or Ridge Regression)

control variable
weight of the coef

$$E(\vec{w}) = \frac{1}{2}\sum_{n=0}^{N-1}(t_n - y(x_n, \vec{w}))^2 + \frac{\lambda}{2}\|\vec{w}\|^2$$

Regularization Derivation

$$\nabla_{\vec{w}}(E(\vec{w})) = 0$$

$$\nabla_{\vec{w}} \left(\frac{1}{2} \sum_{i=0}^{N-1} (y(x_i, \vec{w}) - t_i)^2 + \frac{\lambda}{2} \|\vec{w}\|^2 \right) = 0$$

$$\nabla_{\vec{w}} \left(\frac{1}{2} \|\vec{t} - \vec{X}\vec{w}\|^2 + \frac{\lambda}{2} \|\vec{w}\|^2 \right) = 0$$

$$\nabla_{\vec{w}} \left(\frac{1}{2} (\vec{t} - \vec{X}\vec{w})^T (\vec{t} - \vec{X}\vec{w}) + \frac{\lambda}{2} \vec{w}^T \vec{w} \right) = 0$$

$$\nabla_{\vec{w}} \left(\frac{1}{2}(\vec{t} - \vec{X}\vec{w})^T(\vec{t} - \vec{X}\vec{w}) + \frac{\lambda}{2}\vec{w}^T\vec{w} \right) = 0$$

$$-\vec{X}^T\vec{t} + \vec{X}^T\vec{X}\vec{w} + \nabla_{\vec{w}} \left(\frac{\lambda}{2}\vec{w}^T\vec{w} \right) = 0$$

$$-\vec{X}^T\vec{t} + \vec{X}^T\vec{X}\vec{w} + \lambda\vec{w} = 0$$

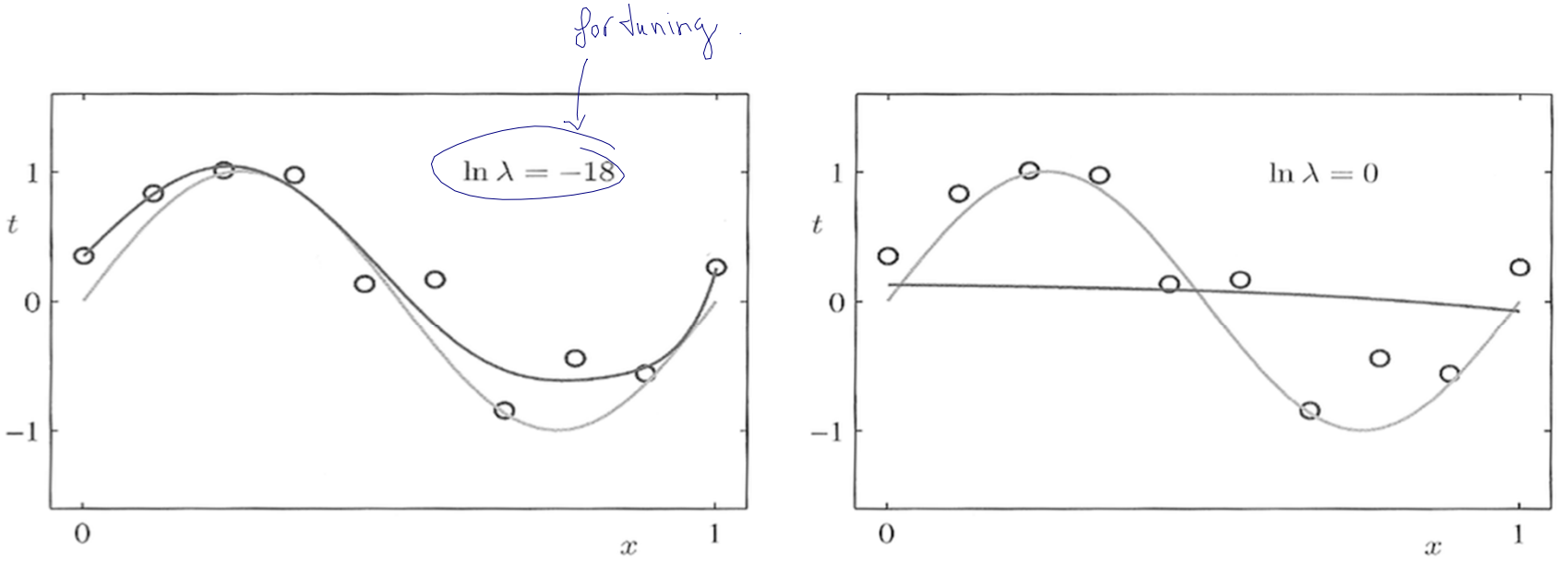
$$-\vec{X}^T\vec{t} + \vec{X}^T\vec{X}\vec{w} + \lambda\vec{I}\vec{w} = 0$$

$$-\vec{X}^T\vec{t} + (\vec{X}^T\vec{X} + \lambda\vec{I})\vec{w} = 0$$

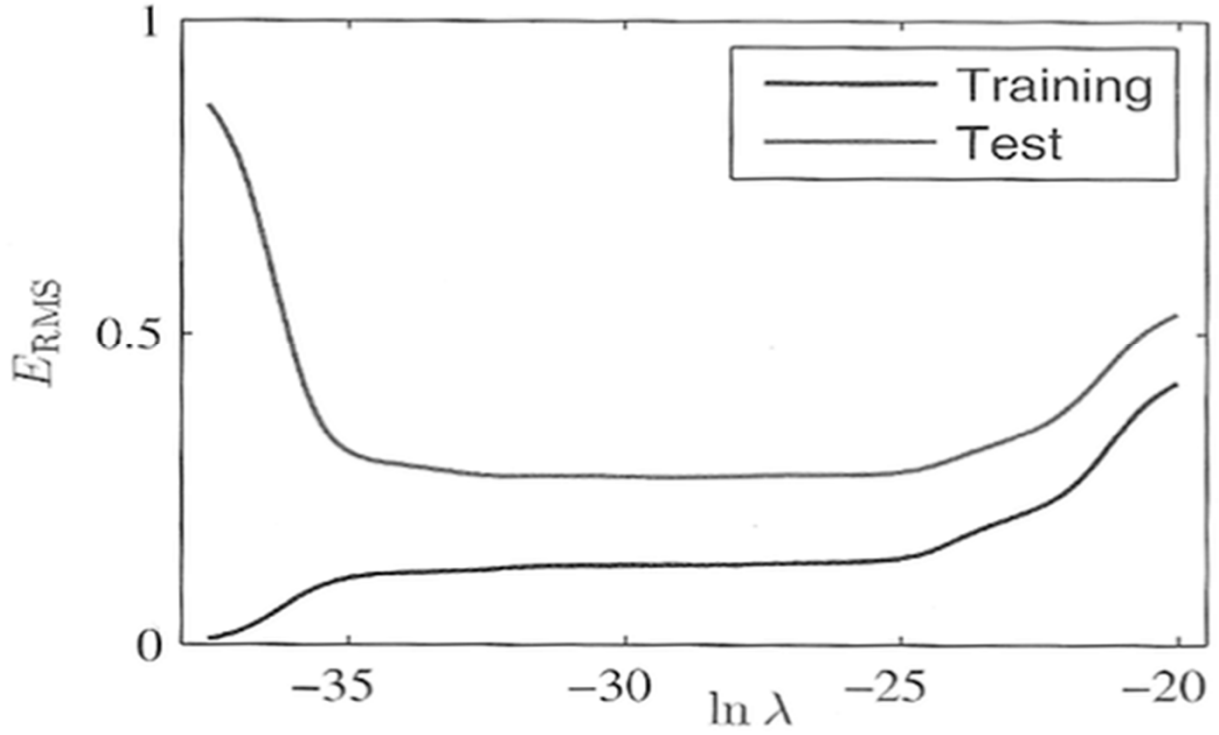
$$(\vec{X}^T\vec{X} + \lambda\vec{I})\vec{w} = \vec{X}^T\vec{t}$$

$$\vec{w} = (\vec{X}^T\vec{X} + \lambda\vec{I})^{-1}\vec{X}^T\vec{t}$$

Regularization in Practice



Regularization Results



More regularization

- The penalty term defines the styles of regularization
- L2-Regularization

$$E(\vec{w}) = \frac{1}{2}\sum_{n=0}^{N-1}(t_n - y(x_n, \vec{w}))^2 + \frac{\lambda}{2}\|\vec{w}\|^2$$

- L1-Regularization

$$E(\vec{w}) = \frac{1}{2}\sum_{n=0}^{N-1}(t_n - y(x_n, \vec{w}))^2 + \lambda|\vec{w}|_1$$

- L0-Regularization
  - **L0-norm** is the optimal subset of features

$$E(\vec{w}) = \frac{1}{2}\sum_{n=0}^{N-1}(t_n - y(x_n, \vec{w}))^2 + \lambda\sum_{n=0}^{N-1}\delta(w_n \neq 0)$$

Stopped on May 29th.

Curse of dimensionality

skipped

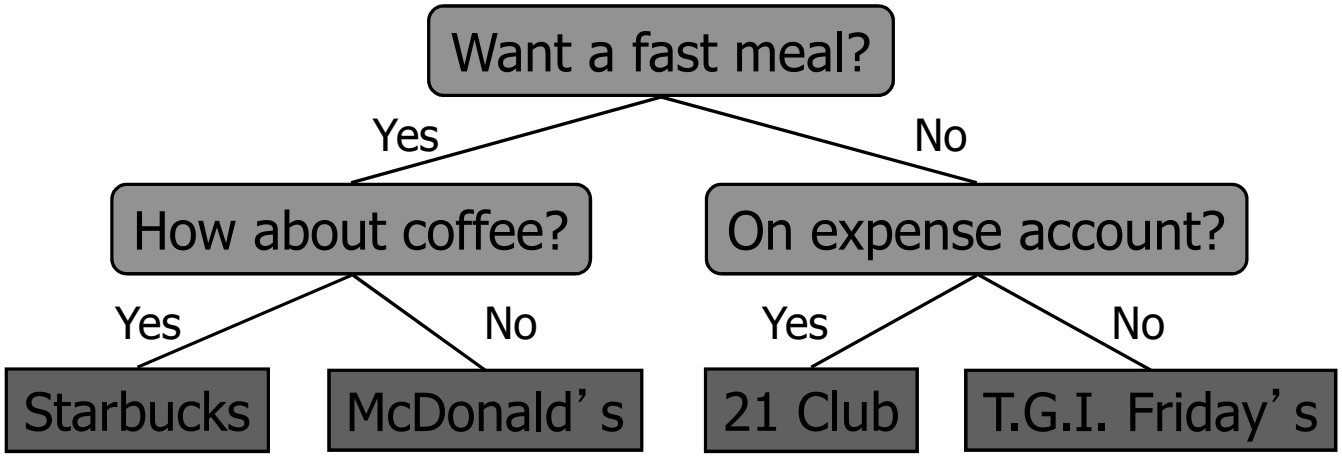
- Increasing dimensionality of features increases the data requirements **exponentially**.
- For example, if a single feature can be accurately approximated with 100 data points, to optimize the joint over two features requires 100*100 data points.

- Models should be small relative to the amount of available data
- Dimensionality reduction techniques – feature selection – can help.
  - L0-regularization is explicit feature selection
  - L1- and L2-regularizations approximate feature selection.

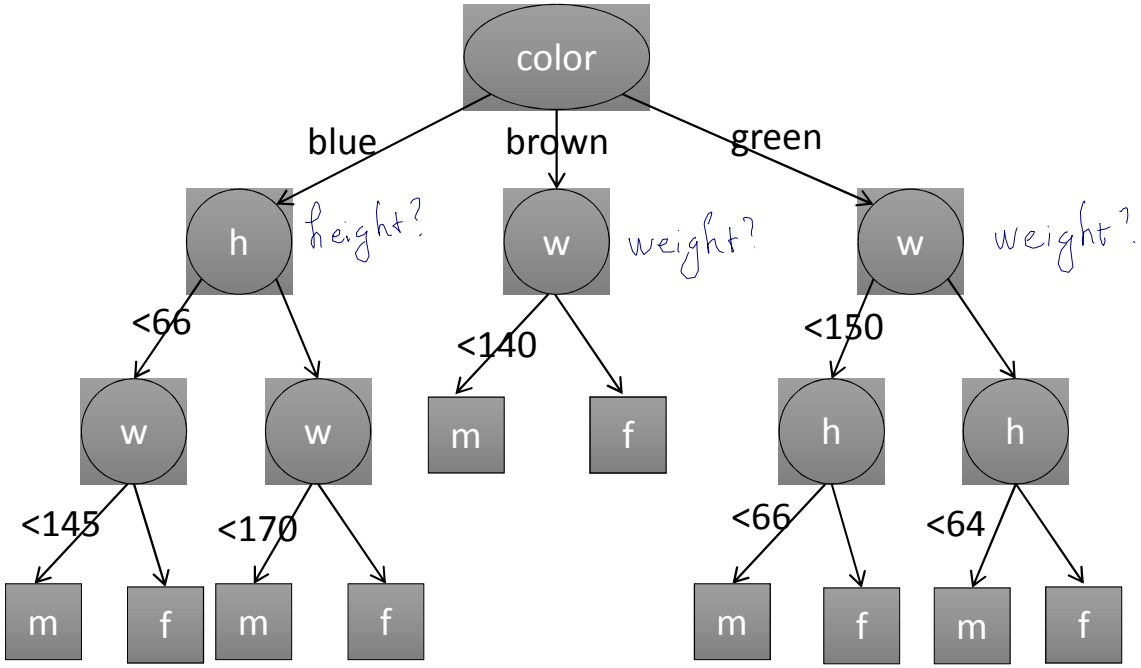
Decision Trees

Decision Trees

- Trees that define a decision process
  - Internal nodes: questions associated with a specific feature
  - Leaves: Decisions



Decision Trees



- Very easy to evaluate.
- Nested if statements

More formal Definition of a Decision Tree

- A **Tree** data structure
- Each **internal node** corresponds to a feature
- **Leaves** are associated with target values.
- Nodes with **nominal features** have *N* children, where *N* is the number of nominal values
- Nodes with **continuous features** have two children for values less than and greater than or equal to a **break point**.

Training a Decision Tree

- How do you decide what feature to use?
- For continuous features how do you decide what break point to use?

- Goal: Optimize Classification Accuracy.

Example Data Set

Height	Weight	Eye Color	Gender
66	170	Blue	Male
73	210	Brown	Male
72	165	Green	Male
70	180	Blue	Male
74	185	Brown	Male
68	155	Green	Male
65	150	Blue	Female
64	120	Brown	Female
63	125	Green	Female
67	140	Blue	Female
68	165	Brown	Female
66	130	Green	Female

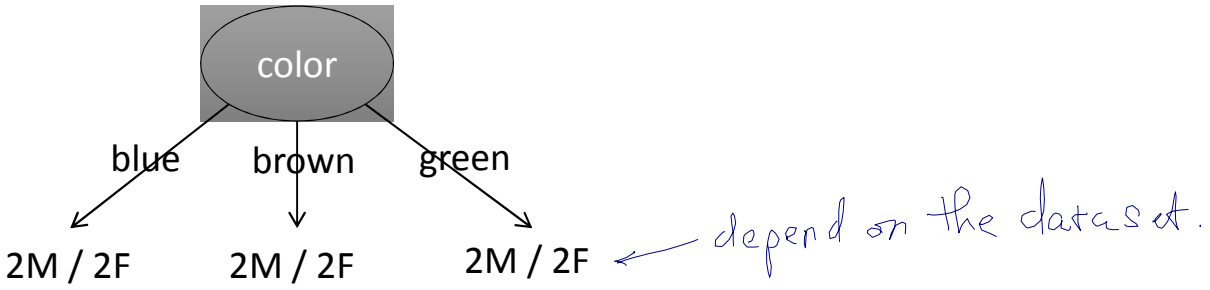
Baseline Classification Accuracy

- Select the majority class.
  - Here 6/12 Male, 6/12 Female.
  - Baseline Accuracy: 50%

- How good is each branch?
  - The improvement to classification accuracy

Training Example

- Possible branches



50% Accuracy before Branch 6M / 6F

50% Accuracy after Branch 2M / 2F for each branch

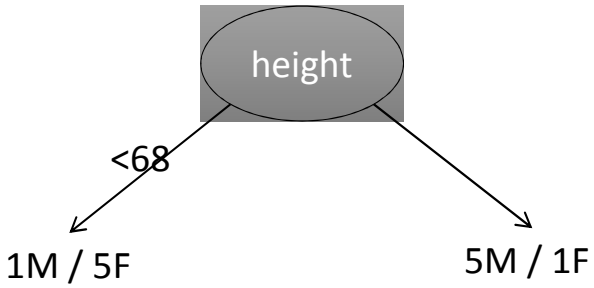
0% Accuracy Improvement

Example Data Set

Height	Weight	Eye Color	Gender
63	125	Green	Female
64	120	Brown	Female
65	150	Blue	Female
66	170	Blue	Male
66	130	Green	Female
67	140	Blue	Female
68	145	Brown	Female
6	155	Green	Male
70	180	Blue	Male
72	165	Green	Male
73	210	Brown	Male
74	185	Brown	Male

Training Example

- Possible branches



50% Accuracy before Branch

5/6 = 83.3% Accuracy after Branch

33.3% Accuracy Improvement

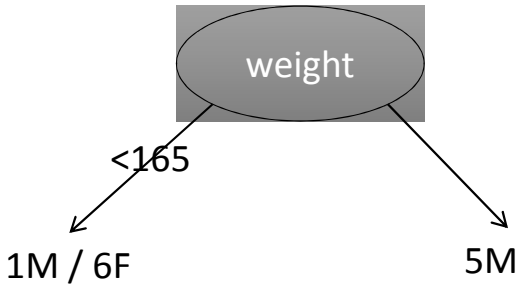
83.3 − 50

Example Data Set

Height	Weight	Eye Color	Gender
64	120	Brown	Female
63	125	Green	Female
66	130	Green	Female
67	140	Blue	Female
68	145	Brown	Female
65	150	Blue	Female
68	155	Green	Male
72	165	Green	Male
66	170	Blue	Male
70	180	Blue	Male
74	185	Brown	Male
73	210	Brown	Male

Training Example

- Possible branches



50% Accuracy before Branch

91.7% Accuracy after Branch $= \frac{\frac{5}{6}+1}{2} = \frac{11}{12}$

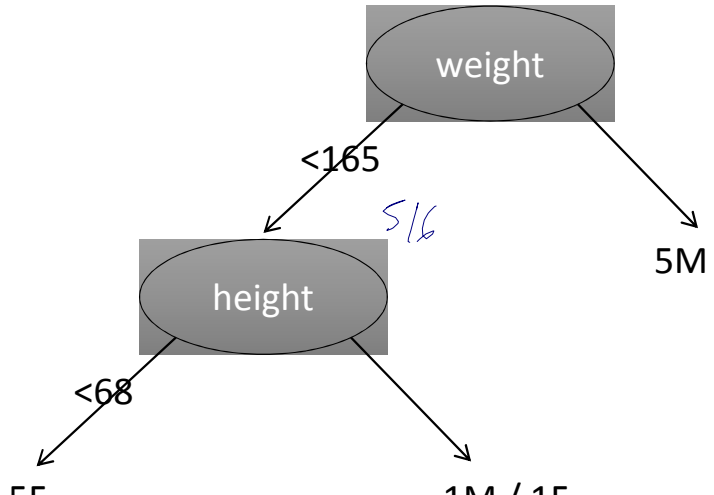
41.7% Accuracy Improvement

↑

91.7 - 50

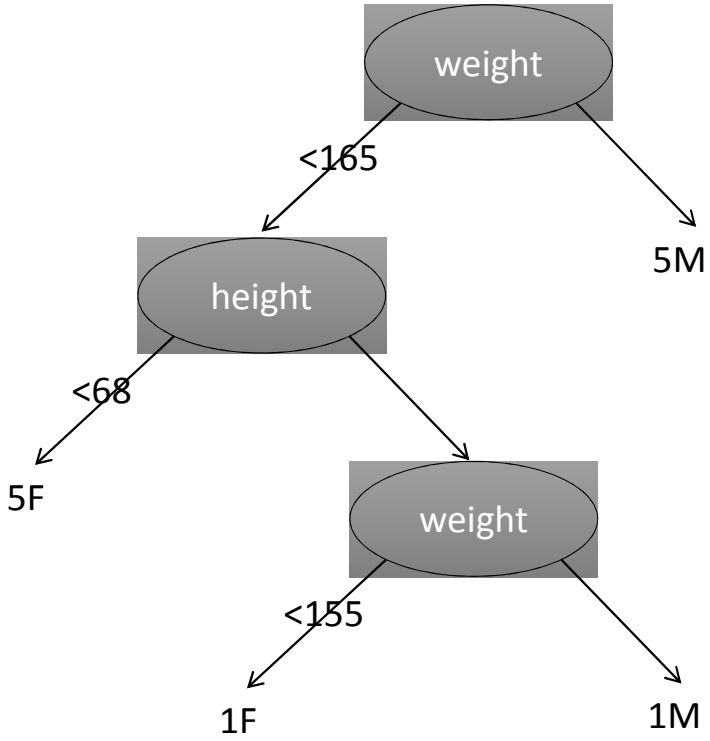
Training Example

- Recursively train child nodes.



Training Example

- Finished Tree



Generalization

- What is the performance of the tree on the training data?
  - Is there any way we could get less than 100% accuracy?
- What performance can we expect on unseen data?

Evaluation

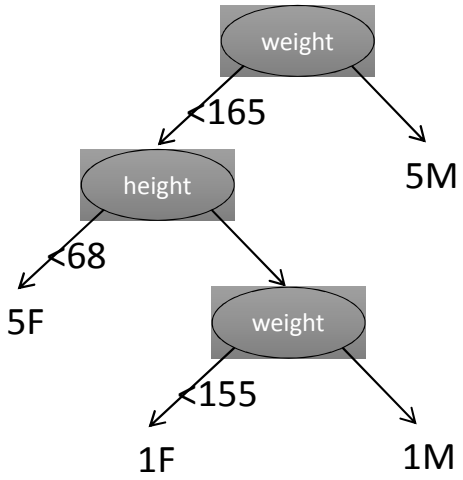
- Evaluate performance on data that was not used in training.
- Isolate a subset of data points to be used for evaluation.
- Evaluate generalization performance.

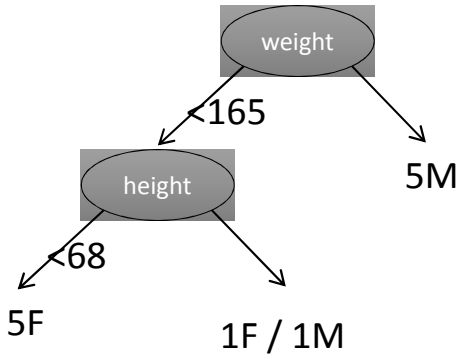
Evaluation of our Decision Tree

- What is the Training performance?
- What is the Evaluation performance?
  - Never classify female over 165
  - Never classify male under 165, and under 68.
  - The middle section is trickier.
- What are some ways to make these similar?

Pruning (when the tree is deep)

- There are many pruning techniques.
- A simple approach is to have a minimum membership size in each node.

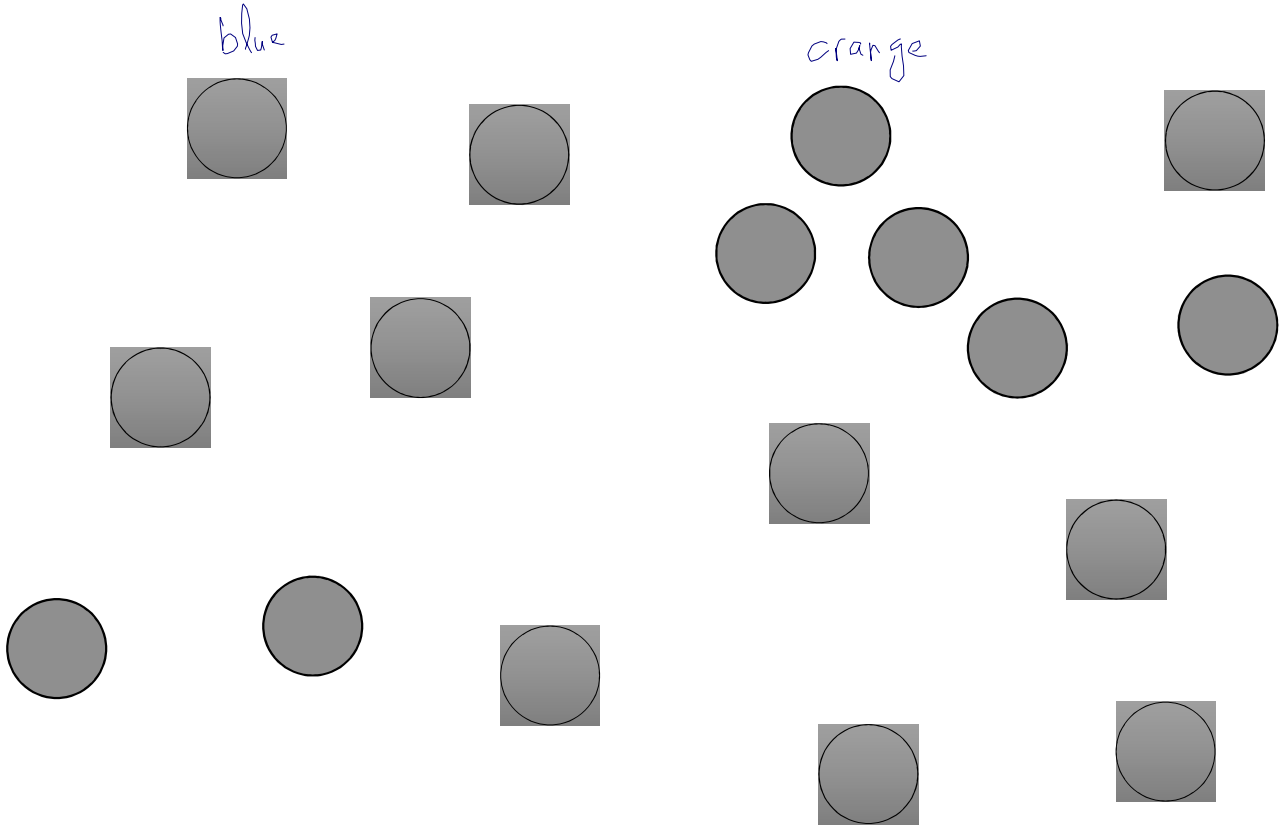




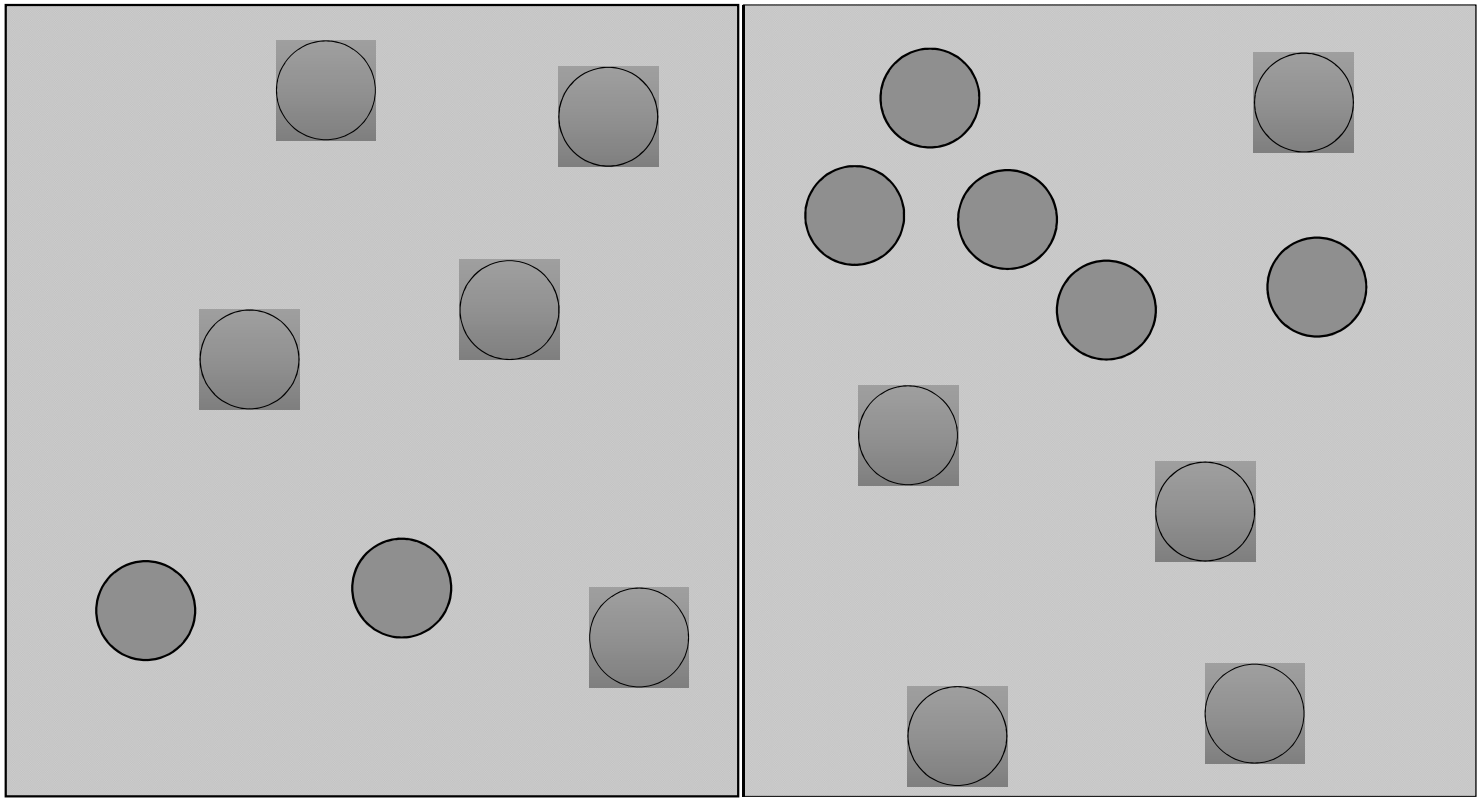
Decision Trees

- Training via **Recursive Partitioning.**
- Simple, interpretable models.
- Different node selection criteria can be used.
  - Information theory is a common choice.
- Pruning techniques can be used to make the model more robust to unseen data.

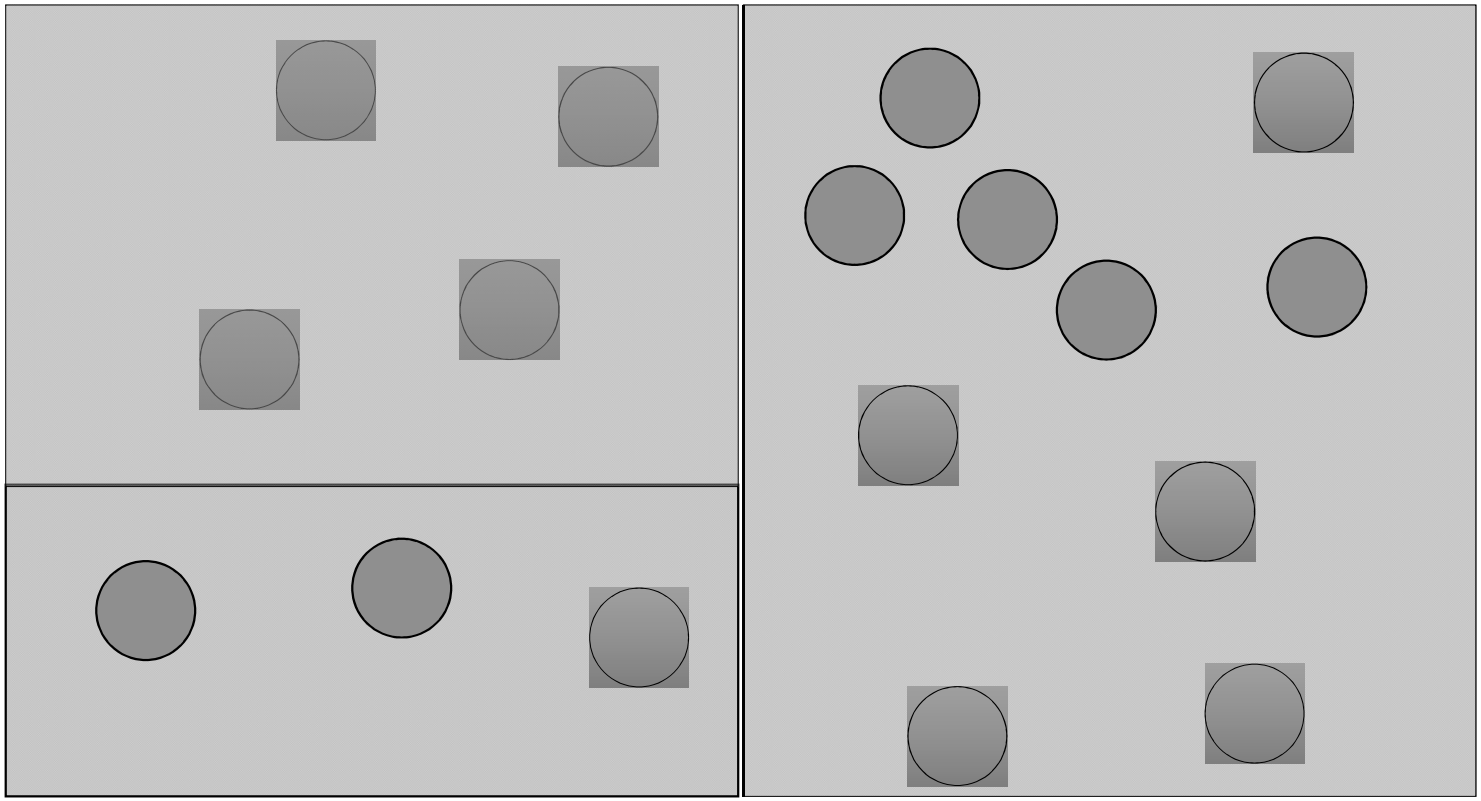
Visualization of Decision Tree Training



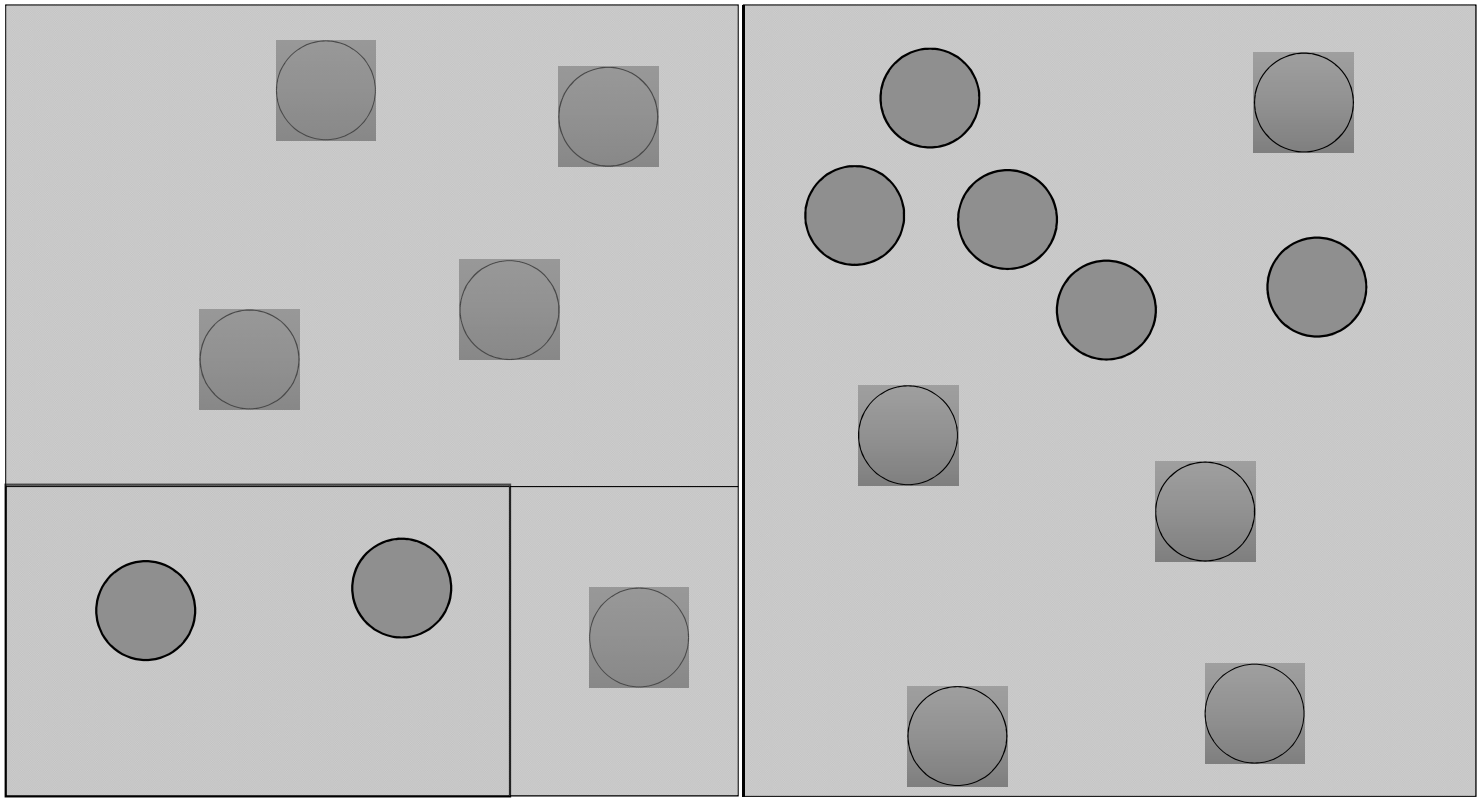
Visualization of Decision Tree Training



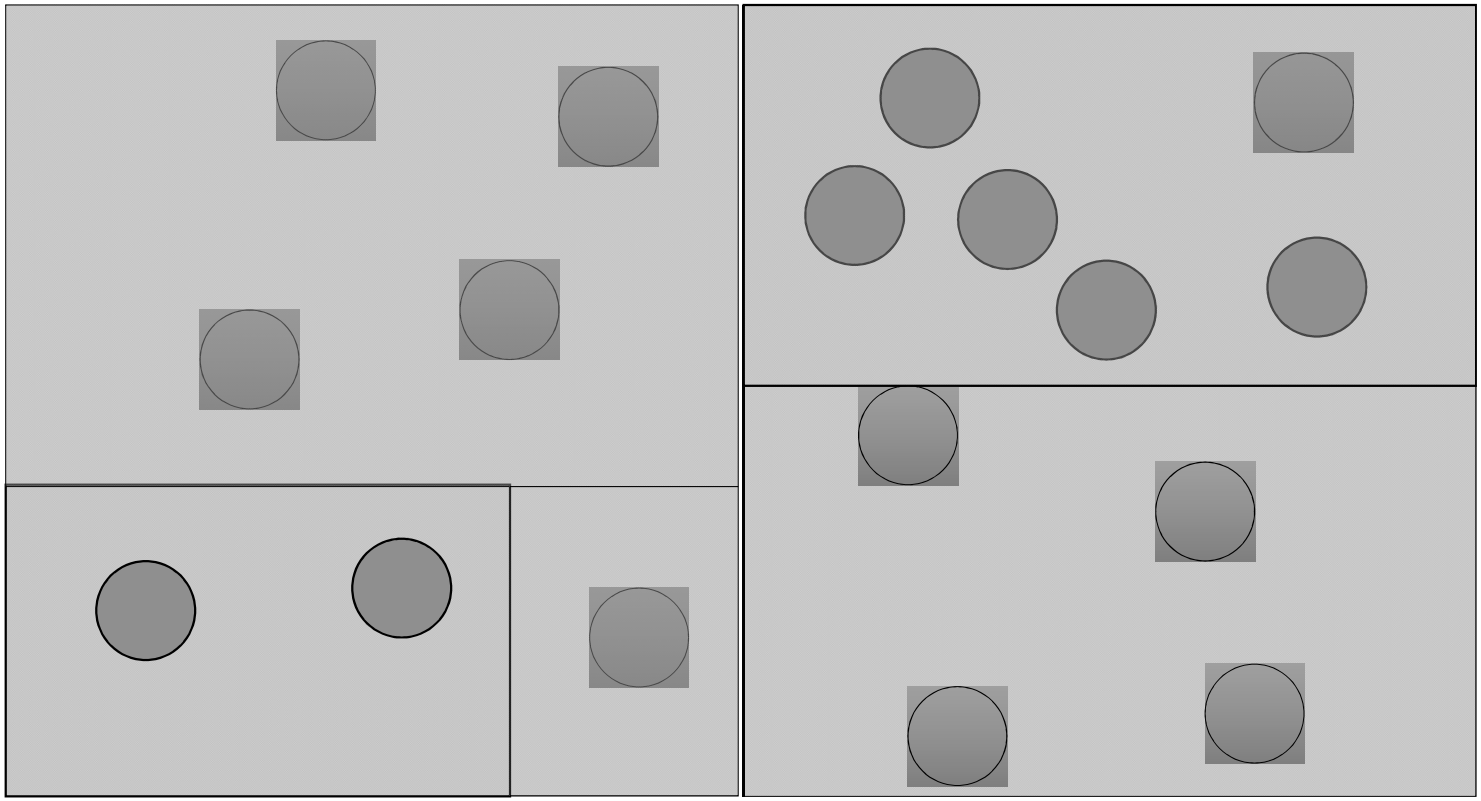
Visualization of Decision Tree Training



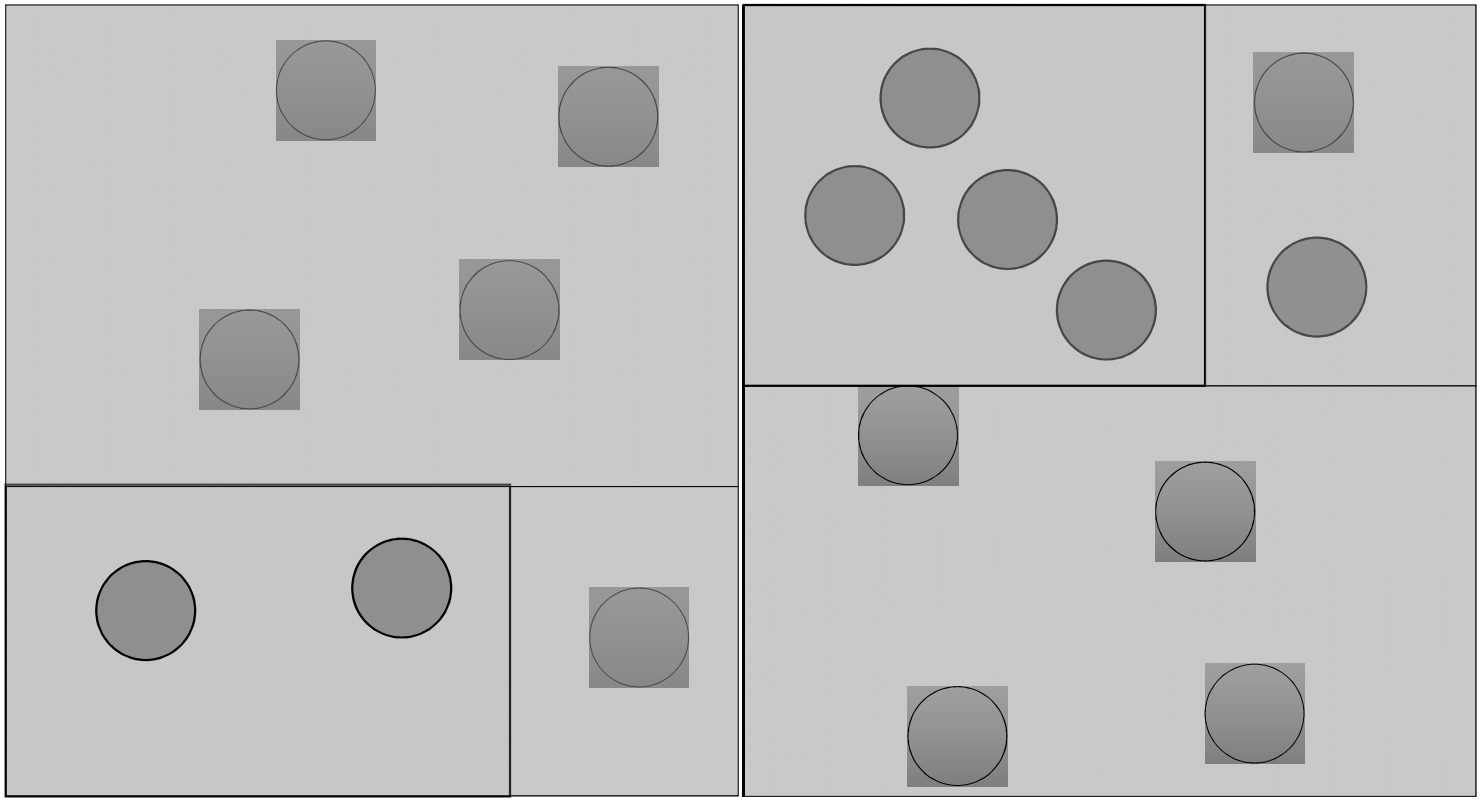
Visualization of Decision Tree Training



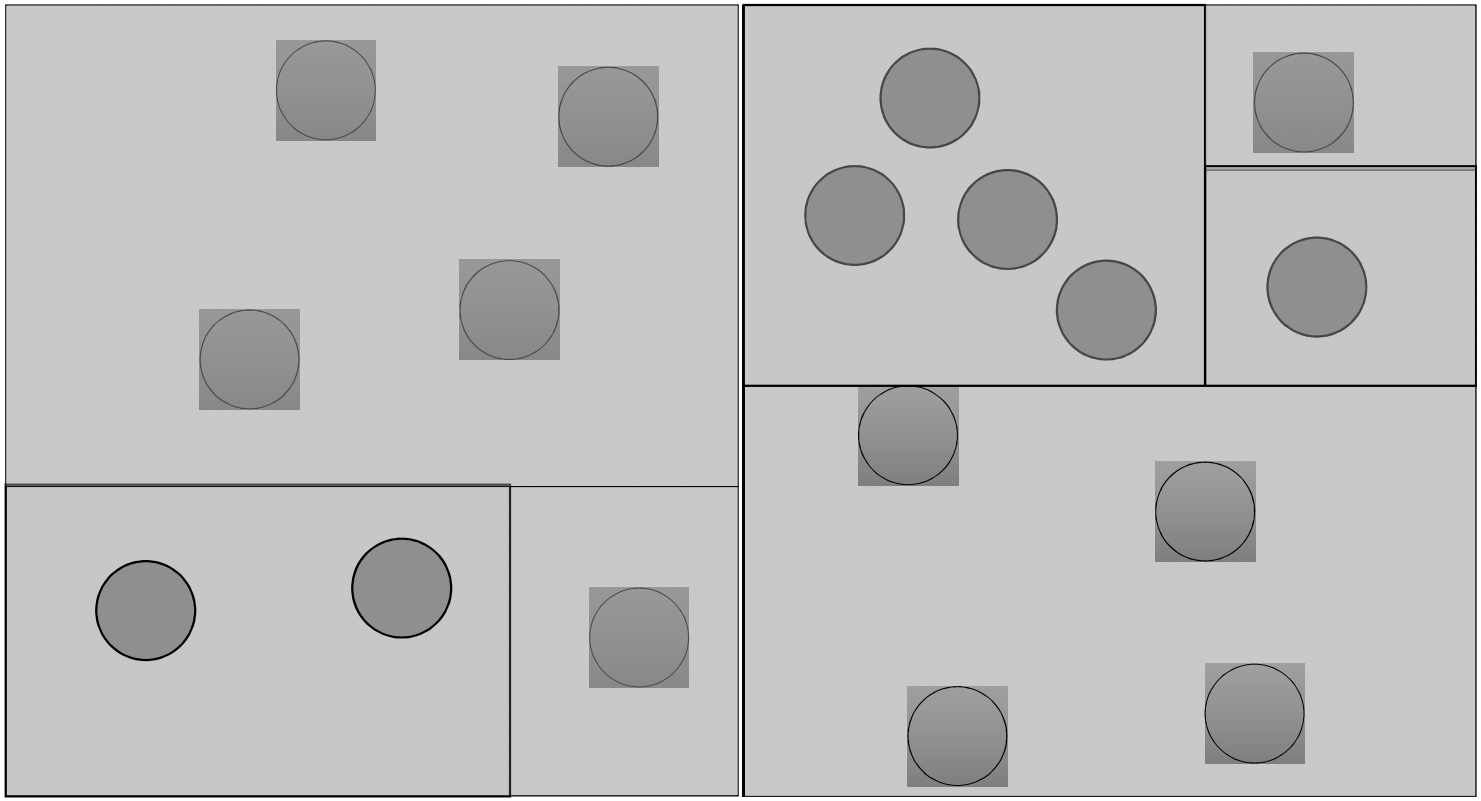
Visualization of Decision Tree Training



Visualization of Decision Tree Training



Visualization of Decision Tree Training



Logistic Regression

Logistic Regression

- Linear model applied to **classification**
- **Supervised**: target information is available
  - Each data point $\mathbf{x}_i$ has a corresponding target t_i.
- Goal: Identify a function

$$y : \mathbb{R}^D \rightarrow C$$

$$\text{where } t_i \in C = \{c_0, \ldots, c_{K-1}\}$$

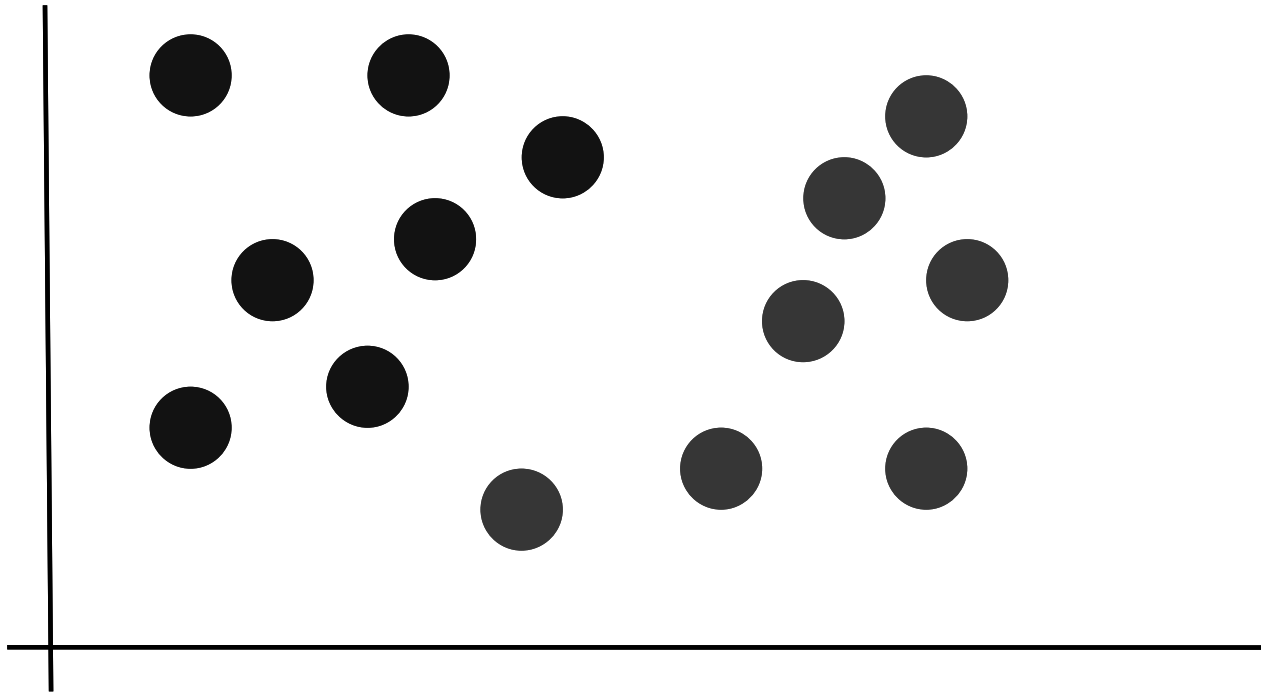
Target Variables

$$y : \mathbb{R}^D \rightarrow C \text{ where } t_i \in C$$

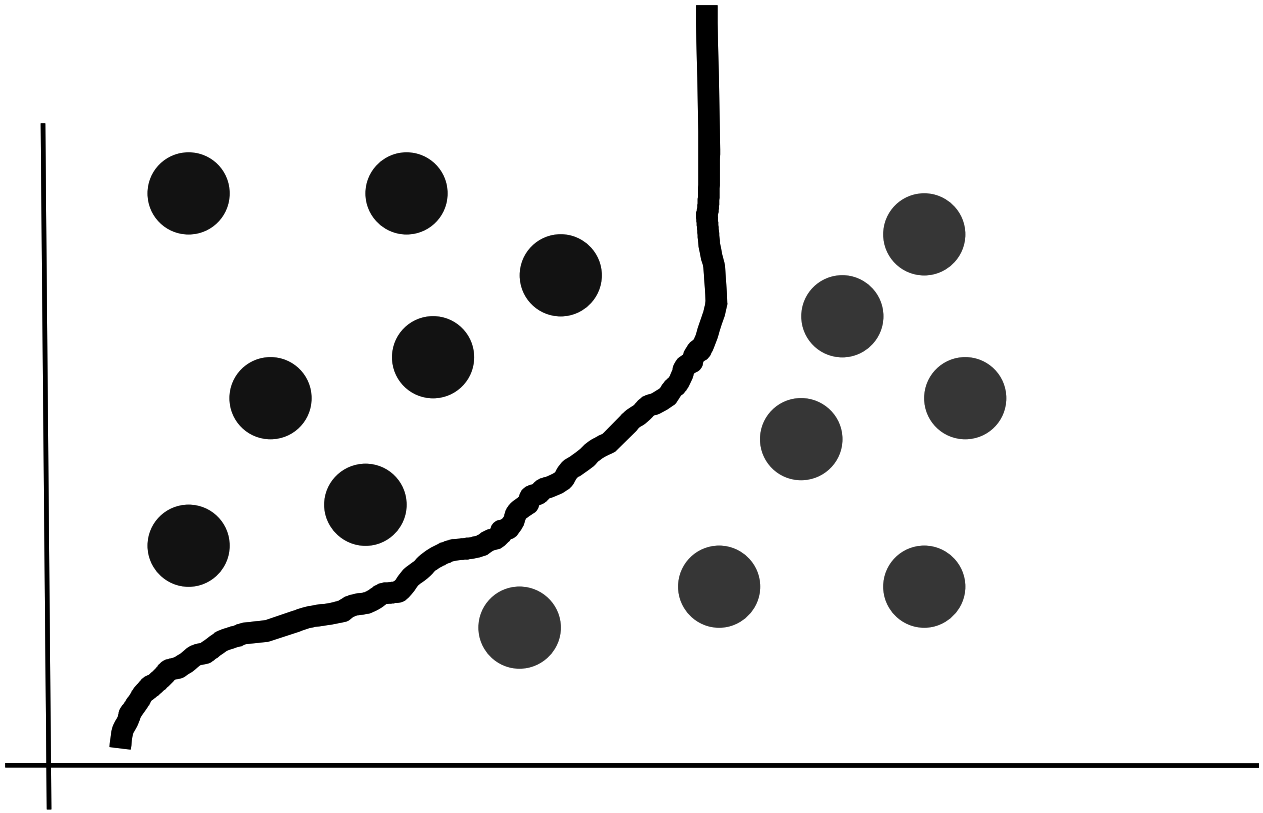
- In **binary** classification, it is convenient to represent t_i as a scalar with a range of [0,1]
  - Interpretation of t_i as the likelihood that $\mathbf{x}_i$ is the member of the positive class
  - Used to represent the **confidence** of a prediction.
- For L > 2 classes, $\mathbf{t}_i$ is often represented as a K element vector.
  - t_{ij} represents the degree of membership in class j.
  - $|\mathbf{t}_i| = 1$
  - E.g. 5-way classification vector:

$$\vec{t} = (0, 0, 1, 0, 0)^T$$

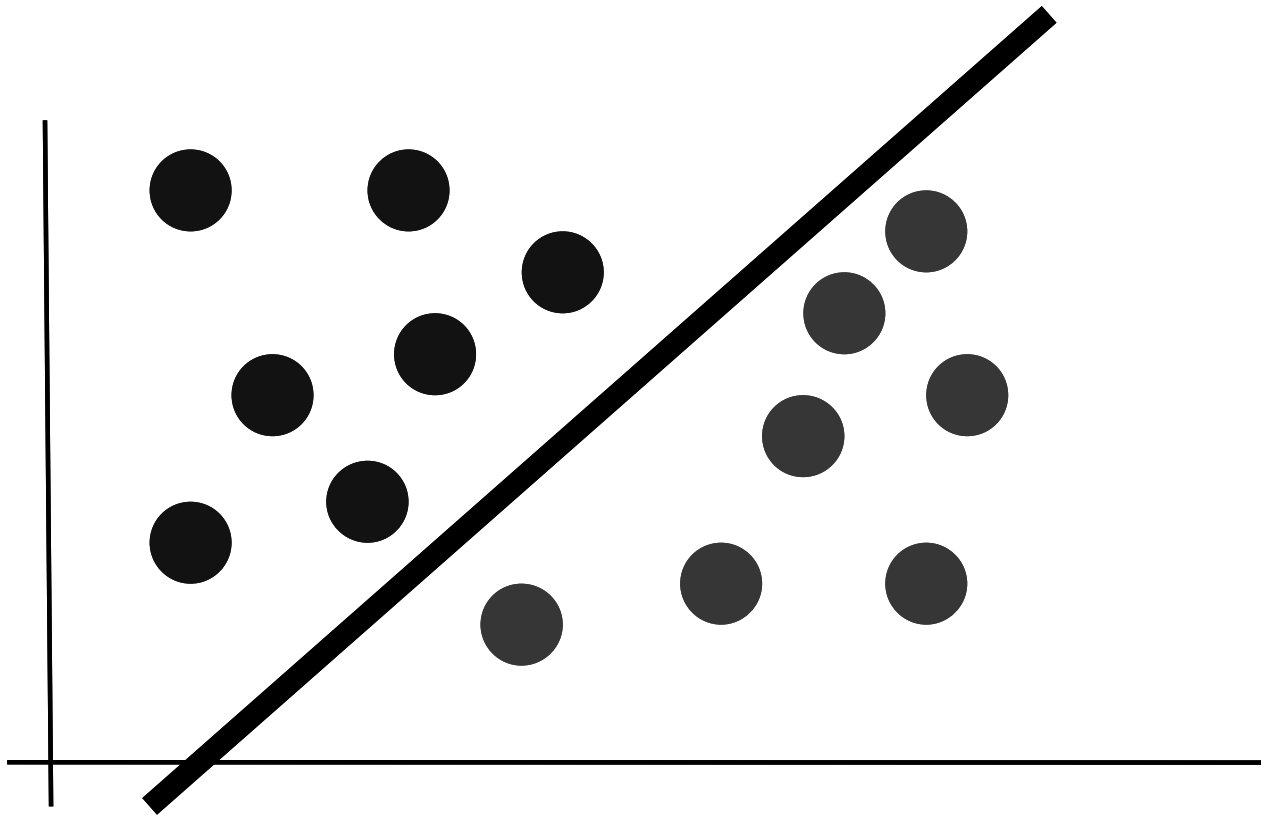
Graphical Example of Classification



Decision Boundaries



Graphical Example of Classification



Classification approaches

- **Generative**
  - Models the joint distribution between c and x
  - Highest data requirements

$$p(c_j|\vec{x}) = \frac{p(\vec{x}|c_j)p(c_j)}{p(\vec{x})}$$

- **Discriminative**
  - Fewer parameters to approximate

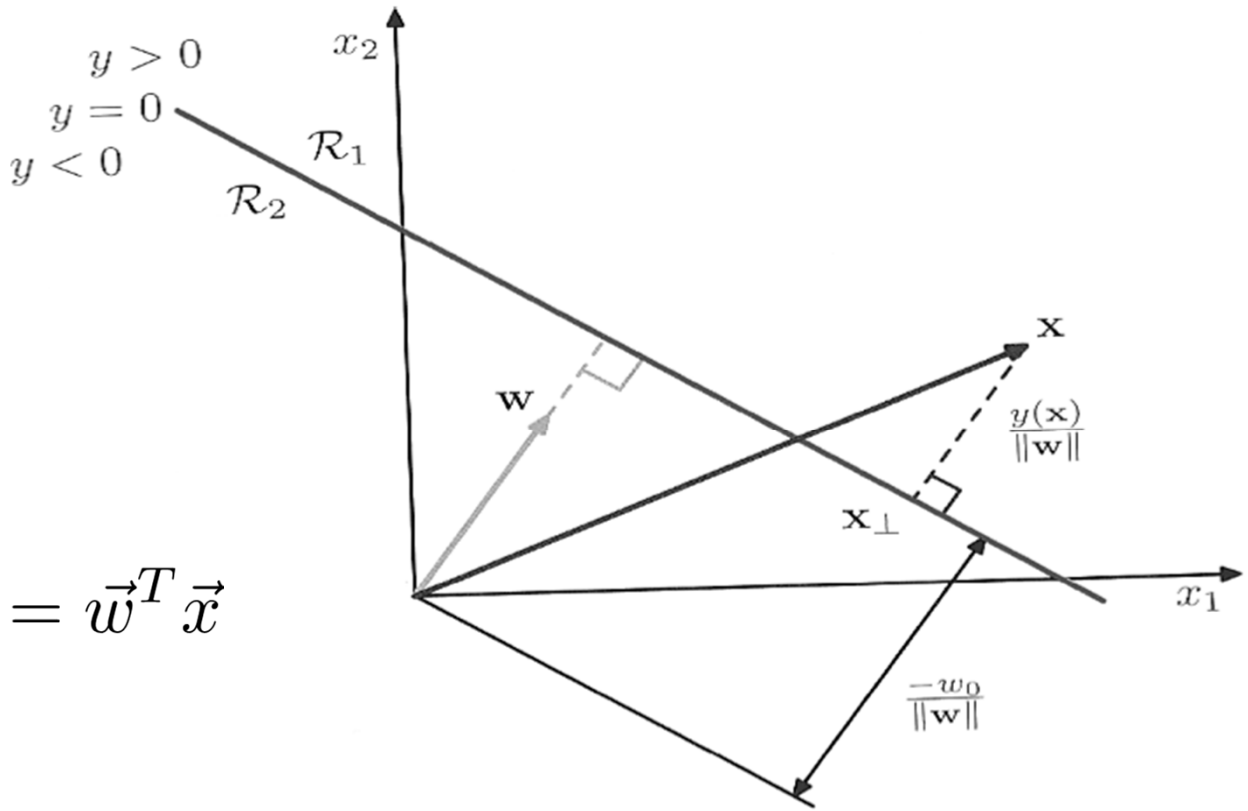
$$p(c_j|\vec{x})$$

- **Discriminant** Function
  - May still be trained probabilistically, but not **necessarily** modeling a likelihood.

$$f(\vec{x}) = c_j$$

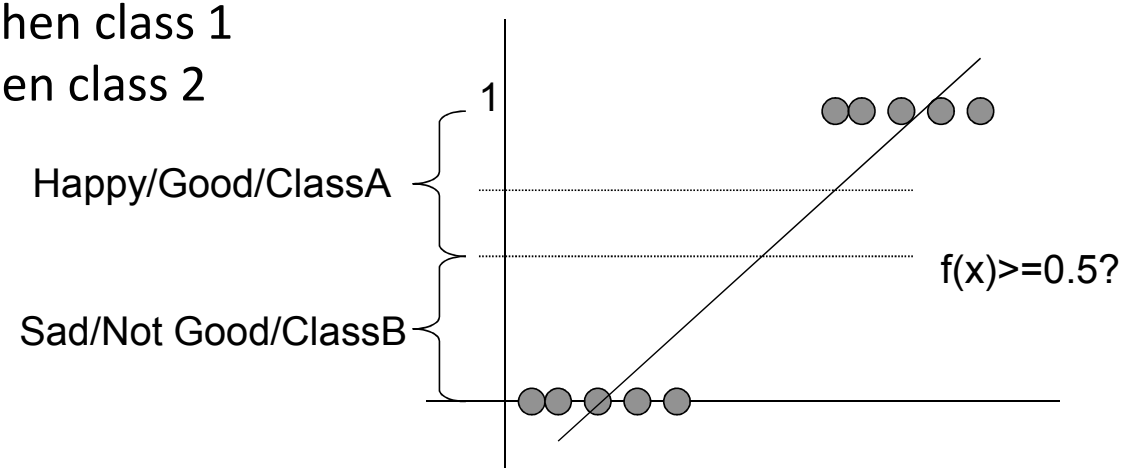
Treating Classification as a Linear model

$$y = \vec{w}^T \vec{x}$$



Relationship between Regression and Classification

- Since we're classifying two classes, why not set one class to '0' and the other to '1' then use linear regression.
  - Regression: -infinity to infinity, while class labels are 0, 1
- Can use a threshold, e.g.
  - y >= 0.5 then class 1
  - y < 0.5 then class 2



Odds-ratio

- Rather than thresholding, we'll relate the regression to the **class-conditional** probability.
- Ratio of the odd of prediction y = 1 or y = 0
  - If p(y=1|x) = 0.8 and p(y=0|x) = 0.2
  - Odds ratio = 0.8/0.2 = 4
- Use a linear model to predict **odds** rather than a class label.

$$\frac{p(y=1|\vec{x})}{p(y=0|\vec{x})} = \vec{w}^T\vec{x}$$

Logit – Log odds ratio function

$$\frac{p(y=1|\vec{x})}{p(y=0|\vec{x})} = \vec{w}^T\vec{x}$$

- LHS: 0 to infinity
- RHS: -infinity to infinity
- Use a log function.
  - Has the added bonus of disolving the division leading to easy manipulation

$$\log \frac{p(y=1|\vec{x})}{p(y=0|\vec{x})}$$

$$\log \frac{p(y=1|\vec{x})}{1-p(y=1|\vec{x})}$$

$$logit(p(x)) = \log \frac{p(x)}{1-p(x)}$$

Logistic Regression

- A **linear model** used to predict log-odds ratio of two classes

linear func

$$\log \frac{p(y=1|\vec{x})}{1-p(y=1|\vec{x})} = \vec{w}^T \vec{x}$$

Logit to probability

$$\log \frac{p(y=1|\vec{x})}{1-p(y=1|\vec{x})} = a$$

$$\frac{p(y=1|\vec{x})}{1-p(y=1|\vec{x})} = \exp(a)$$

$$p(y=1|\vec{x}) = \exp(a)\,(1-p(y=1|\vec{x}))$$

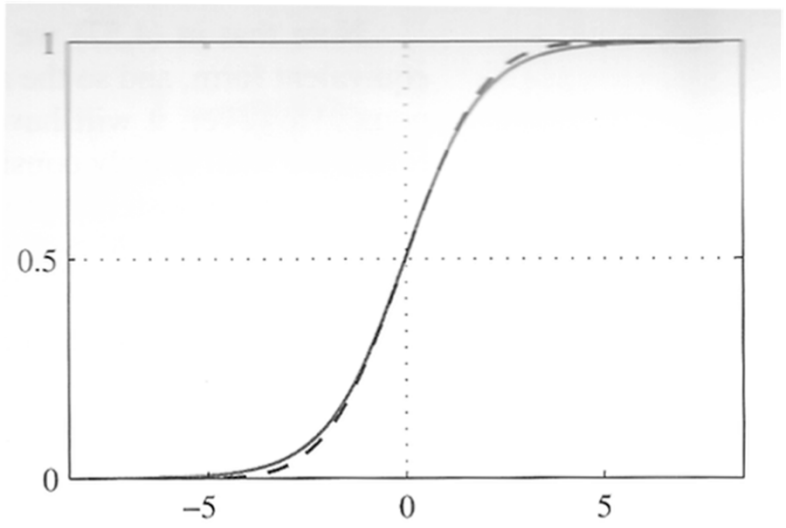
$$p(y=1|\vec{x}) + \exp(a)p(y=1|\vec{x}) = \exp(a)$$

$$p(y=1|\vec{x}) = \frac{\exp(a)}{1+\exp(a)}$$

$$p(y=1|\vec{x}) = \frac{1}{1+\exp(-a)}$$

Sigmoid function

$$\sigma(x) = \frac{1}{1 + exp(-x)}$$



- Squashing function to map the reals to a finite domain.

$$\sigma : \mathbb{R} \rightarrow (0, 1)$$