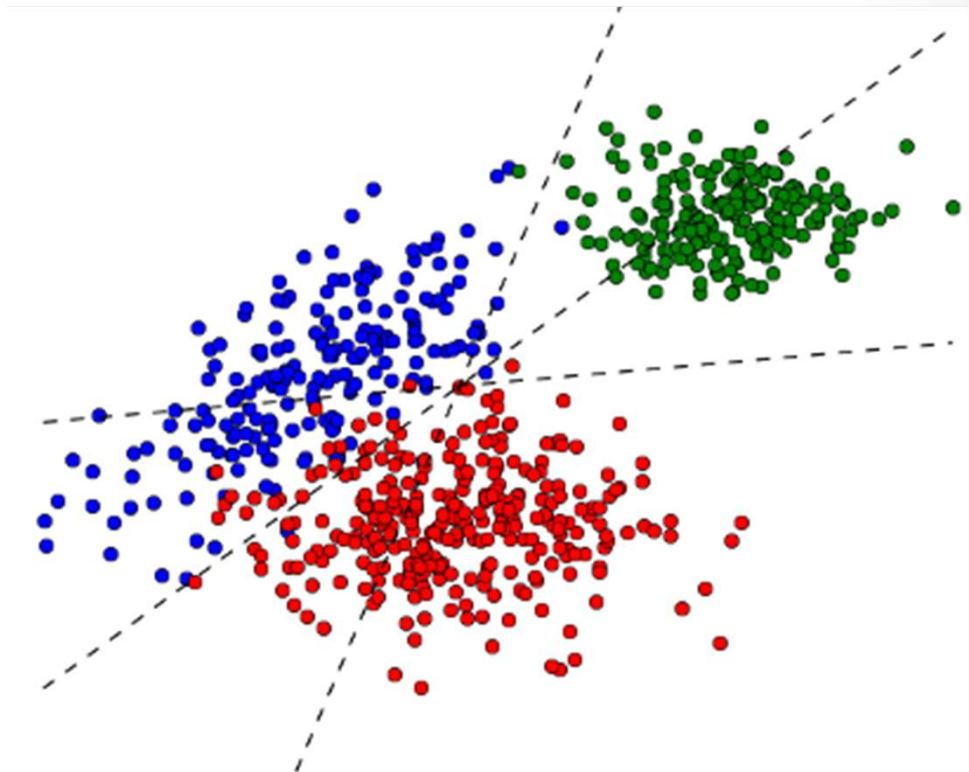


TOWARD REAL-WORLD DATA MULTICLASS TO SGD

許志仲 (Chih-Chung Hsu)

Assistant Professor
Institute of Data Science,
National Cheng Kung University



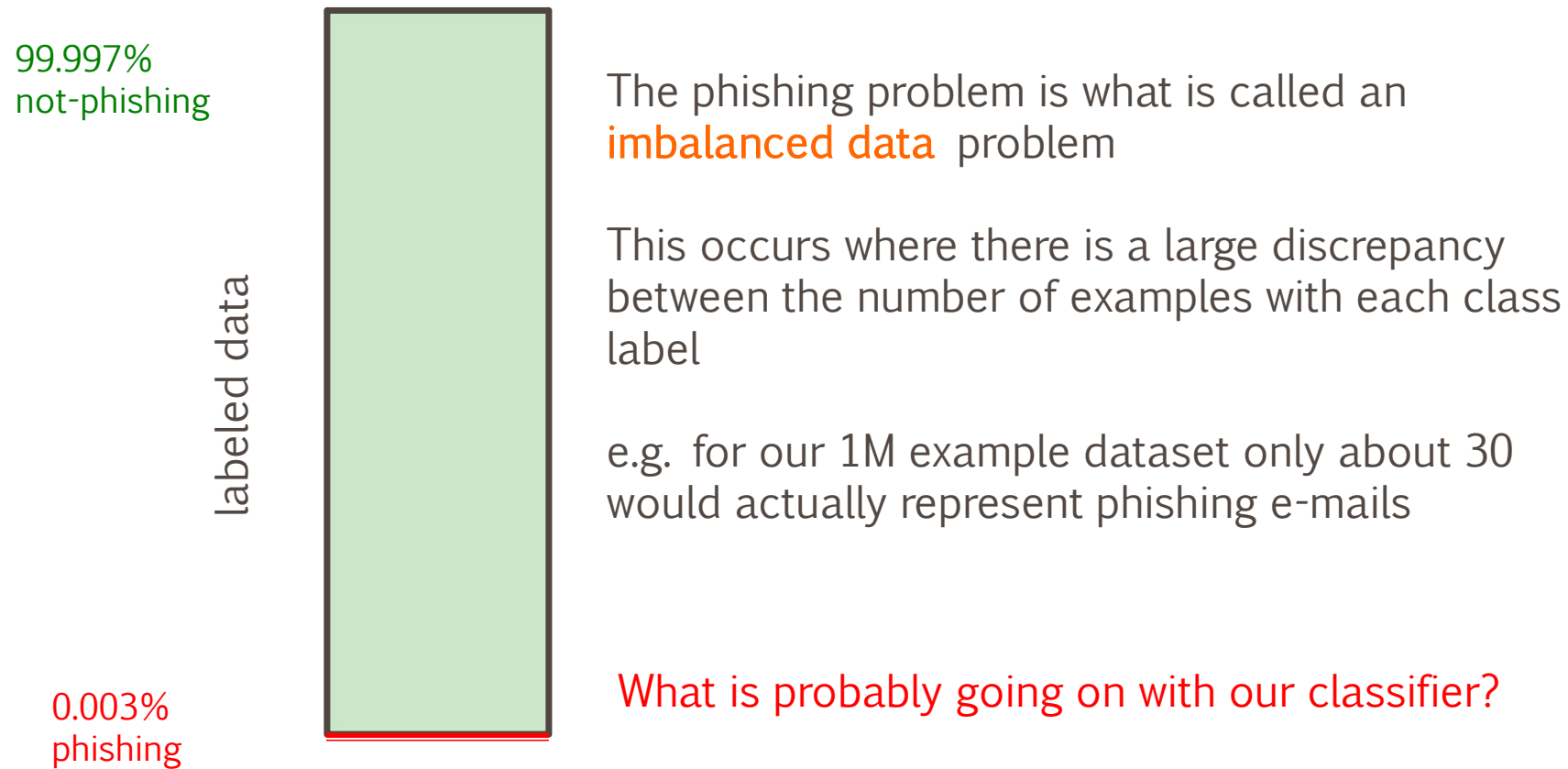
Final Project

- Online project (Kaggle)
 - <https://www.kaggle.com/competitions/optiver-trading-at-the-close>
 - Please register an account associate with your ID with prefix NCKU
- Final project presentation
 - TBD
- Oral presentation
 - Each team has 30min to present their work
 - Your solution, including how it work, how you analyze data and code demo!!
 - Team members: Max 3 & Min 1
 - Please hand up your team information before 10/30
 - Late submission will be disqualified
 - Report is necessary (Will give a referenced format)
- All code, slide, and report (PDF/WORD) should be uploaded to Moodle!

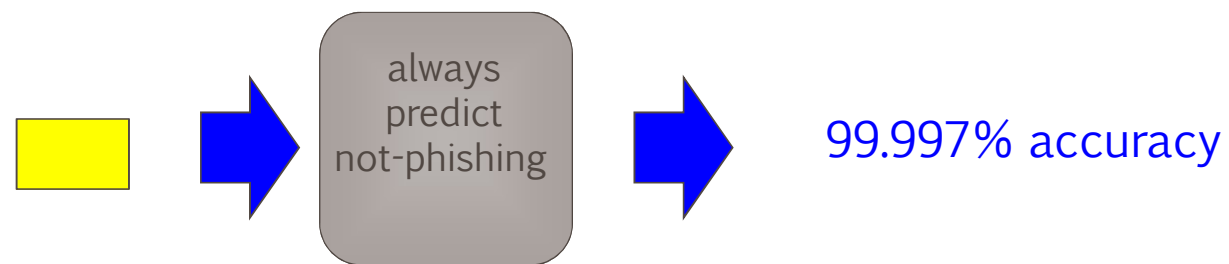


IMBALANCED DATA

Imbalanced data



Imbalanced data



Why does the classifier learn this?

Imbalanced data

- Many classifiers are designed to optimize error/accuracy
- This tends to bias performance towards the majority class
- *Anytime* there is an imbalance in the data this can happen
- It is particularly pronounced, though, when the imbalance is more pronounced

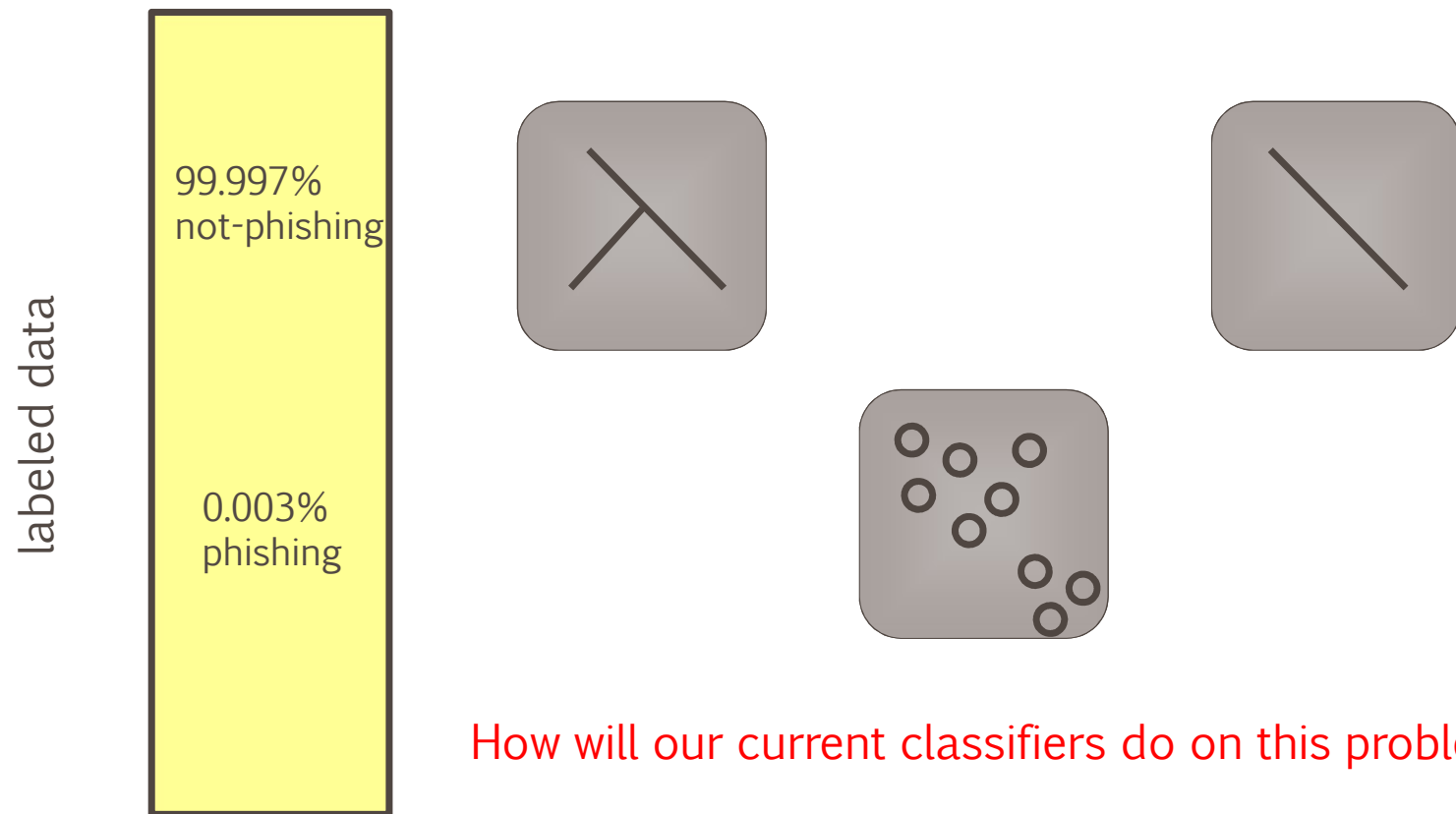
Imbalanced problem domains

Besides phishing (and spam) what are some other imbalanced problems domains?

Imbalanced problem domains

- Medical diagnosis
- Predicting faults/failures (e.g. hard-drive failures, mechanical failures, etc.)
- Predicting rare events (e.g. earthquakes)
- Detecting fraud (credit card transactions, internet traffic)

Imbalanced data: current classifiers



Imbalanced data: current classifiers

- All will do fine if the data can be easily separated/distinguished
- Decision trees:
 - explicitly minimizes training error
 - when pruning pick “majority” label at leaves
 - tend to do very poor at imbalanced problems
- k-NN:
 - even for small k, majority class will tend to overwhelm the vote
- perceptron:
 - can be reasonable since only updates when a mistake is made
 - can take a long time to learn

Part of the problem: evaluation

- Accuracy is not the right measure of classifier performance in these domains
- Other ideas for evaluation measures?

“identification” tasks

- View the task as trying to find/identify “positive” examples (i.e. the rare events)

Precision: proportion of test examples *predicted* as positive that are correct

$$\frac{\# \text{ correctly predicted as positive}}{\# \text{ examples predicted as positive}}$$

Recall: proportion of test examples *labeled* as positive that are correct

$$\frac{\# \text{ correctly predicted as positive}}{\# \text{ positive examples in test set}}$$

“identification” tasks

Precision: proportion of test examples *predicted* as positive that are correct

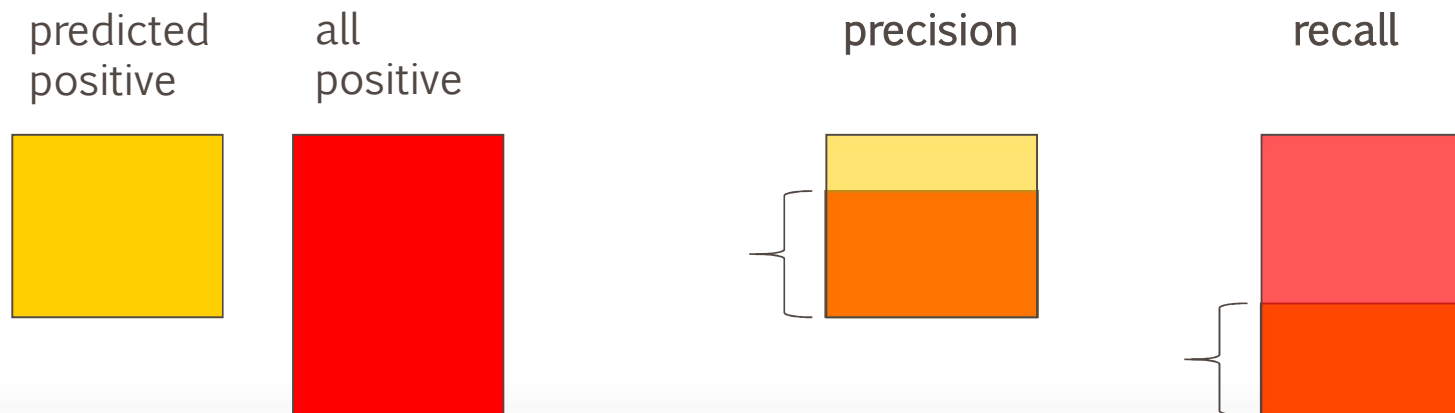
correctly predicted as positive

examples predicted as positive

Recall: proportion of test examples *labeled* as positive that are correct

correctly predicted as positive

positive examples in test set



precision and recall

data	label	predicted
------	-------	-----------



0

0



0

1



1

0



1

1



0

1



1

1










0

0

$$\text{precision} = \frac{\text{\# correctly predicted as positive}}{\text{\# examples predicted as positive}}$$

$$\text{recall} = \frac{\text{\# correctly predicted as positive}}{\text{\# positive examples in test set}}$$

precision and recall

data	label	predicted
	0	0
	0	1
	1	0
	1	1
	0	1
	1	1
	0	0








$$\text{precision} = \frac{\text{\# correctly predicted as positive}}{\text{\# examples predicted as positive}}$$

$$\text{recall} = \frac{\text{\# correctly predicted as positive}}{\text{\# positive examples in test set}}$$

$$\text{precision} = \frac{2}{4}$$

$$\text{recall} = \frac{2}{3}$$

precision and recall








data	label	predicted
	0	0
	0	1
	1	0
	1	1
	0	1
	1	1
	0	0

$$\text{precision} = \frac{\text{\# correctly predicted as positive}}{\text{\# examples predicted as positive}}$$

$$\text{recall} = \frac{\text{\# correctly predicted as positive}}{\text{\# positive examples in test set}}$$

Why do we have both measures?
How can we maximize precision?
How can we maximize recall?

Maximizing precision








data	label	predicted
	0	0
	0	0
	1	0
	1	0
	0	0
	1	0
	0	0

$$\text{precision} = \frac{\text{\# correctly predicted as positive}}{\text{\# examples predicted as positive}}$$

$$\text{recall} = \frac{\text{\# correctly predicted as positive}}{\text{\# positive examples in test set}}$$

Don't predict anything as positive!

Maximizing recall

data	label	predicted
	0	1
	0	1
	1	1
	1	1
	0	1
	1	1
	0	1

$$\text{precision} = \frac{\text{\# correctly predicted as positive}}{\text{\# examples predicted as positive}}$$








$$\text{recall} = \frac{\text{\# correctly predicted as positive}}{\text{\# positive examples in test set}}$$

Predict everything as positive!

precision vs. recall








- Often there is a tradeoff between precision and recall
- increasing one, tends to decrease the other
- For our algorithms, how might we increase/decrease precision/recall?

precision/recall tradeoff

data	label	predicted	confidence
	0	0	0.75
	0	1	0.60
	1	0	0.20
	1	1	0.80
	0	1	0.50
	1	1	0.55
	0	0	0.90

- For many classifiers we can get some notion of the prediction confidence
- Only predict positive if the confidence is above a given threshold
- By varying this threshold, we can vary precision and recall

precision/recall tradeoff

data	label	predicted	confidence
	1	1	0.80
	0	1	0.60
	1	1	0.55
	0	1	0.50
	1	0	0.20
	0	0	0.75
	0	0	0.90








- put most confident positive predictions at top

- put most confident negative predictions at bottom

- calculate precision/recall at each break point/threshold

- classify everything above threshold as positive and everything else negative








precision/recall tradeoff

data	label	predicted	confidence	precision	recall
	1	1	0.80		
	0	1	0.60	≥ 0.8	
	1	1	0.55		
	0	1	0.50		
	1	0	0.20		
	0	0	0.75		
	0	0	0.90		








$$1/2 = 0.5$$

$$1/3 = 0.33$$








precision/recall tradeoff

data	label	predicted	confidence	precision	recall
	1	1	0.80		
	0	1	0.60		
	1	1	0.55		
	0	1	0.50		
	1	0	0.20	≥ 0.55 $3/5 = 0.6$	$3/3 = 1.0$
	0	0	0.75		
	0	0	0.90		

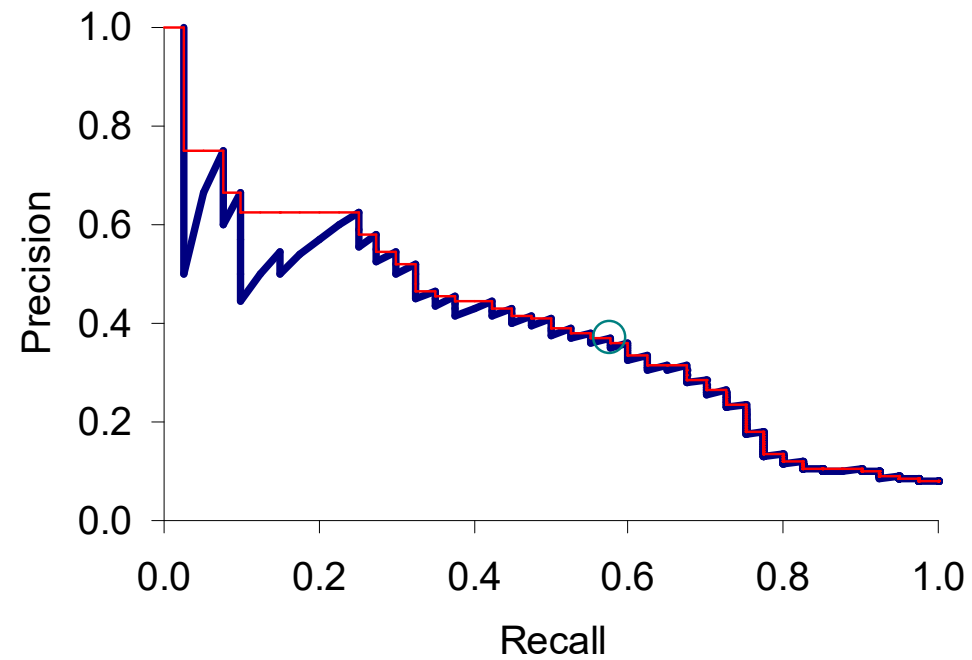
precision/recall tradeoff

data	label	predicted	confidence	precision	recall
	1	1	0.80		
	0	1	0.60		
	1	1	0.55		
	0	1	0.50		
	1	0	0.20		
	0	0	0.75		
	0	0	0.90	≥ 0	
				$3/7 = 0.43$	$3/3 = 1.0$

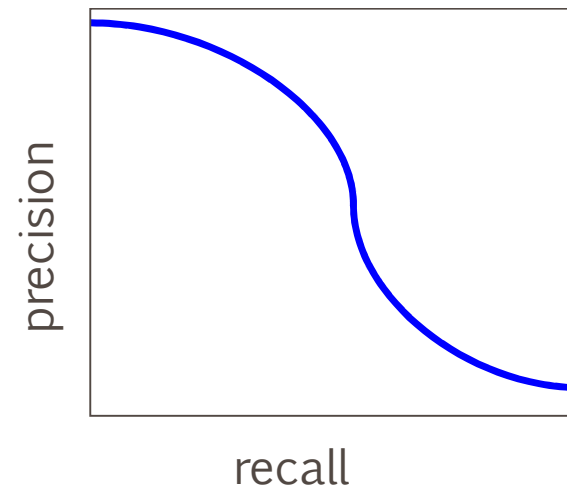
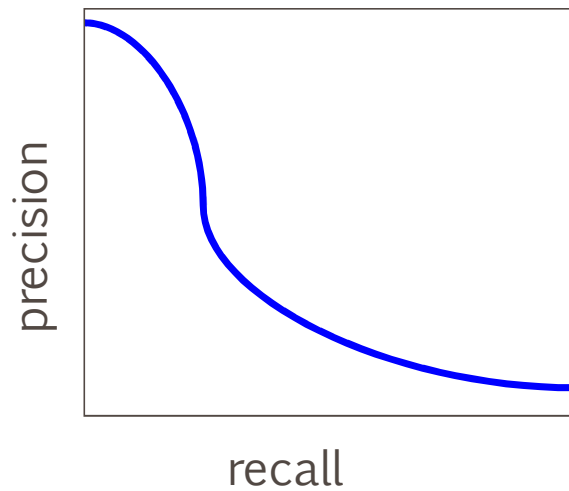
precision/recall tradeoff

data	label	predicted	confidence	precision	recall
	1	1	0.80	1.0	0.33
	0	1	0.60	0.5	0.33
	1	1	0.55	0.66	0.66
	0	1	0.50	0.5	0.66
	1	0	0.20	0.6	1.0
	0	0	0.75	0.5	1.0
	0	0	0.90	0.43	1.0

precision-recall curve



Which system is better?

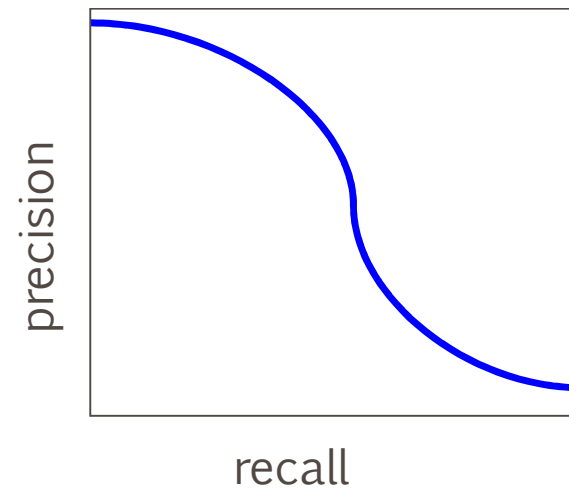
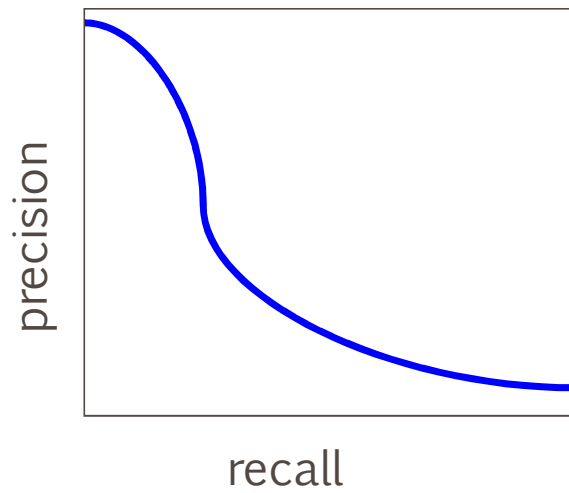


How can we quantify this?

Area under the curve

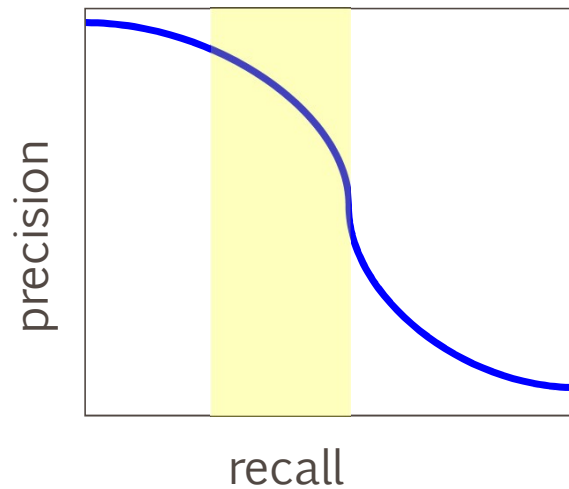
- Area under the curve (AUC) is one metric that encapsulates both precision and recall
- calculate the precision/recall values for all thresholding of the test set (like we did before)
- then calculate the area under the curve
- can also be calculated as the average precision for all the recall points

Area under the curve?

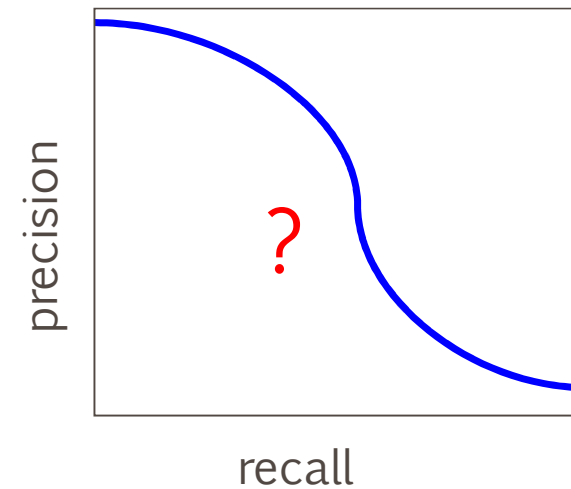


Any concerns/problems?

Area under the curve?

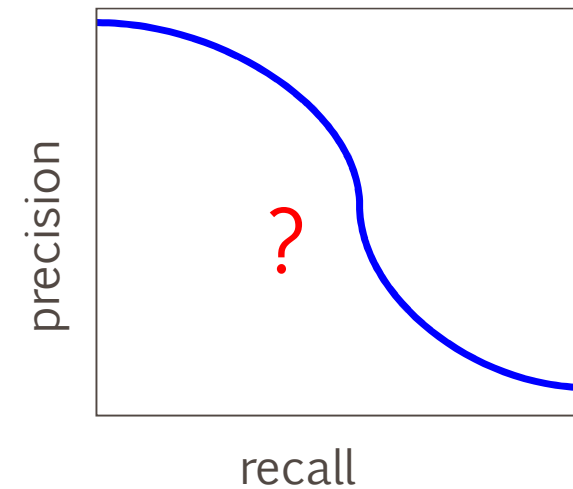
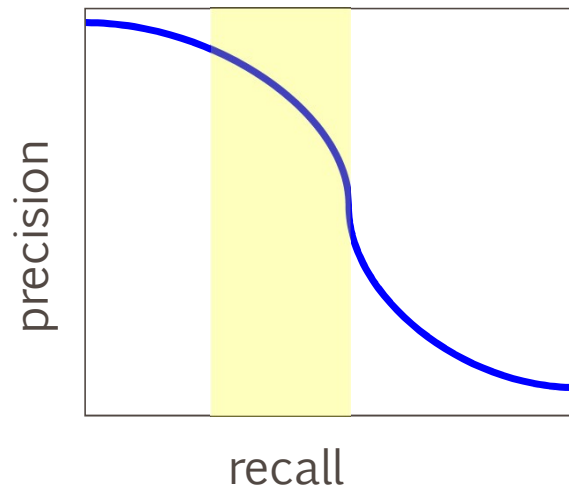


For real use, often only interested in performance in a particular range



Eventually, need to deploy. How do we decide what threshold to use?

Area under the curve?



Ideas? We'd like a compromise between precision and recall

A combined measure: F

- Combined measure that assesses precision/recall tradeoff is **F measure** (weighted harmonic mean):

$$F = \frac{1}{\alpha \frac{1}{P} + (1 - \alpha) \frac{1}{R}} = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

F1-measure

Most common $\alpha=0.5$: equal balance/weighting between precision and recall:

$$F = \frac{1}{\alpha \frac{1}{P} + (1-\alpha) \frac{1}{R}} = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

$$F1 = \frac{1}{0.5 \frac{1}{P} + 0.5 \frac{1}{R}} = \frac{2PR}{P + R}$$

A combined measure: F

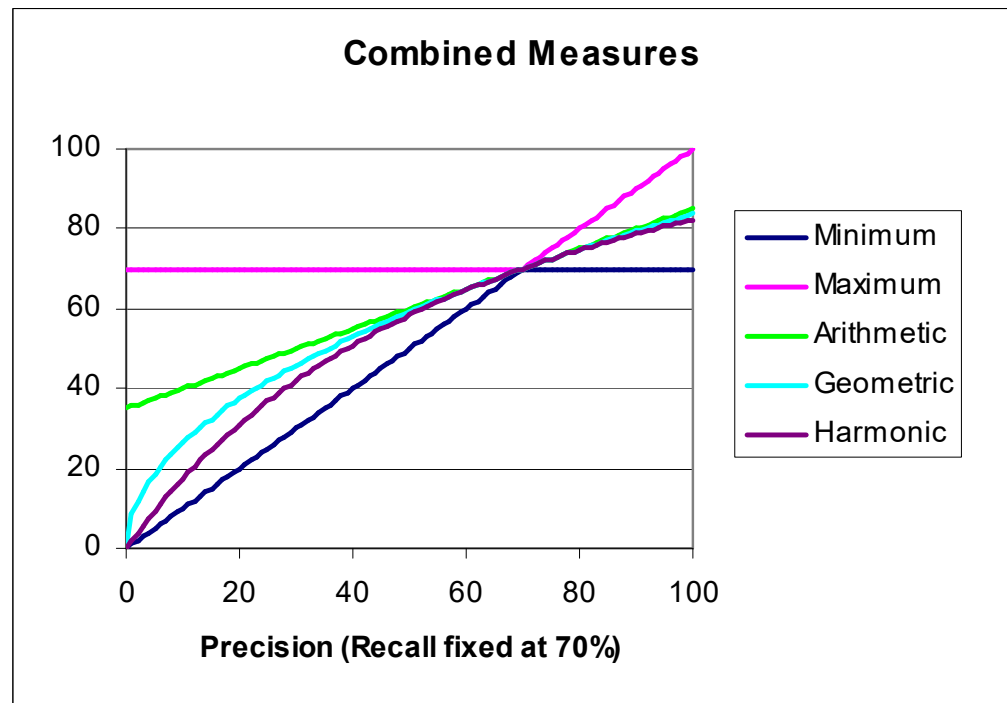
Combined measure that assesses precision/recall tradeoff is **F measure** (weighted harmonic mean):

$$F = \frac{1}{\alpha \frac{1}{P} + (1 - \alpha) \frac{1}{R}} = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

Why harmonic mean?

Why not normal mean (i.e. average)?

F_1 and other averages



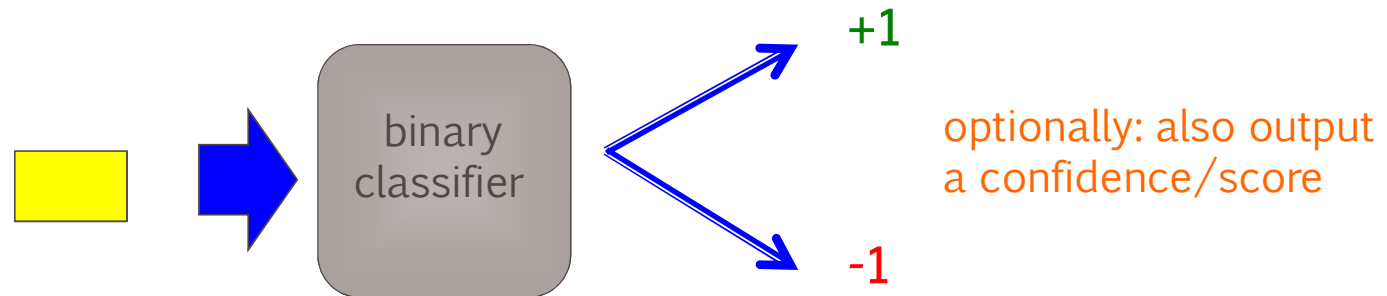
Harmonic mean encourages precision/recall values that are similar!

Evaluation summarized

- Accuracy is often NOT an appropriate evaluation metric for imbalanced data problems
- precision/recall capture different characteristics of our classifier
- AUC and F1 can be used as a single metric to compare algorithm variations (and to tune hyperparameters)

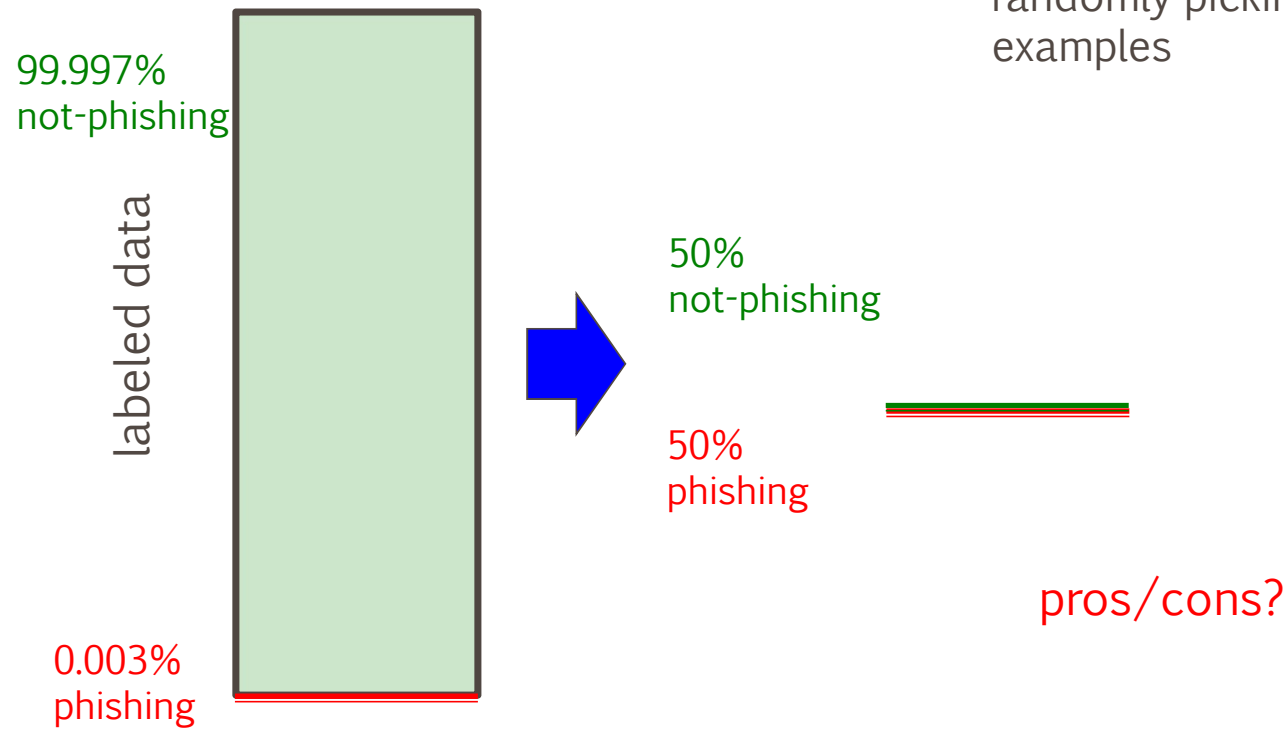
Black box approach

Abstraction: we have a generic binary classifier, how can we use it to solve our new problem



Can we do some pre-processing/post-processing of our data to allow us to still use our binary classifiers?

Idea 1: subsampling



- Create a new training data set by:
- including all k “positive” examples
 - randomly picking k “negative” examples

Subsampling

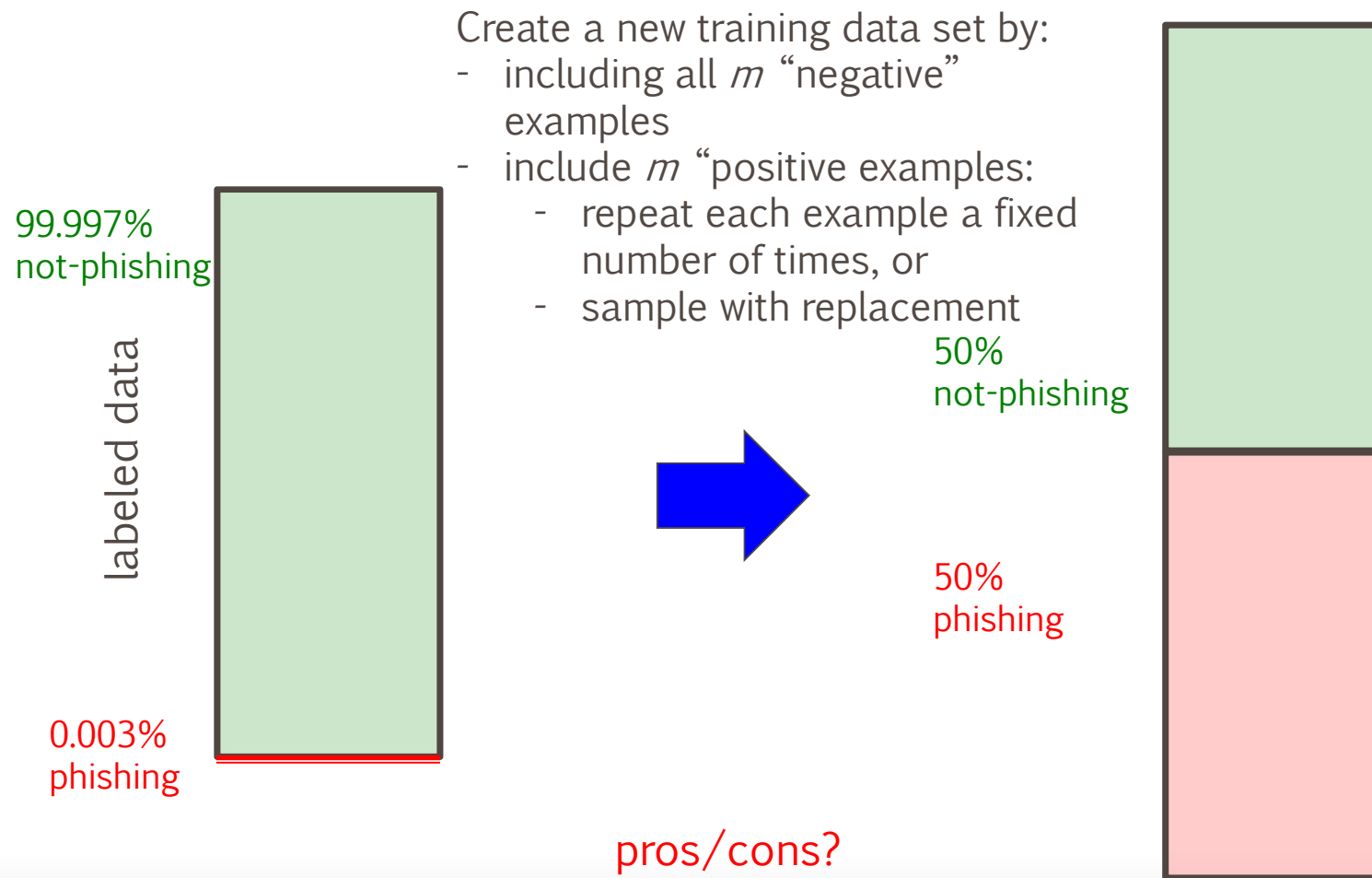
Pros:

- Easy to implement
- Training becomes much more efficient (smaller training set)
- For some domains, can work very well

Cons:

- Throwing away a lot of data/information

Idea 2: oversampling



oversampling

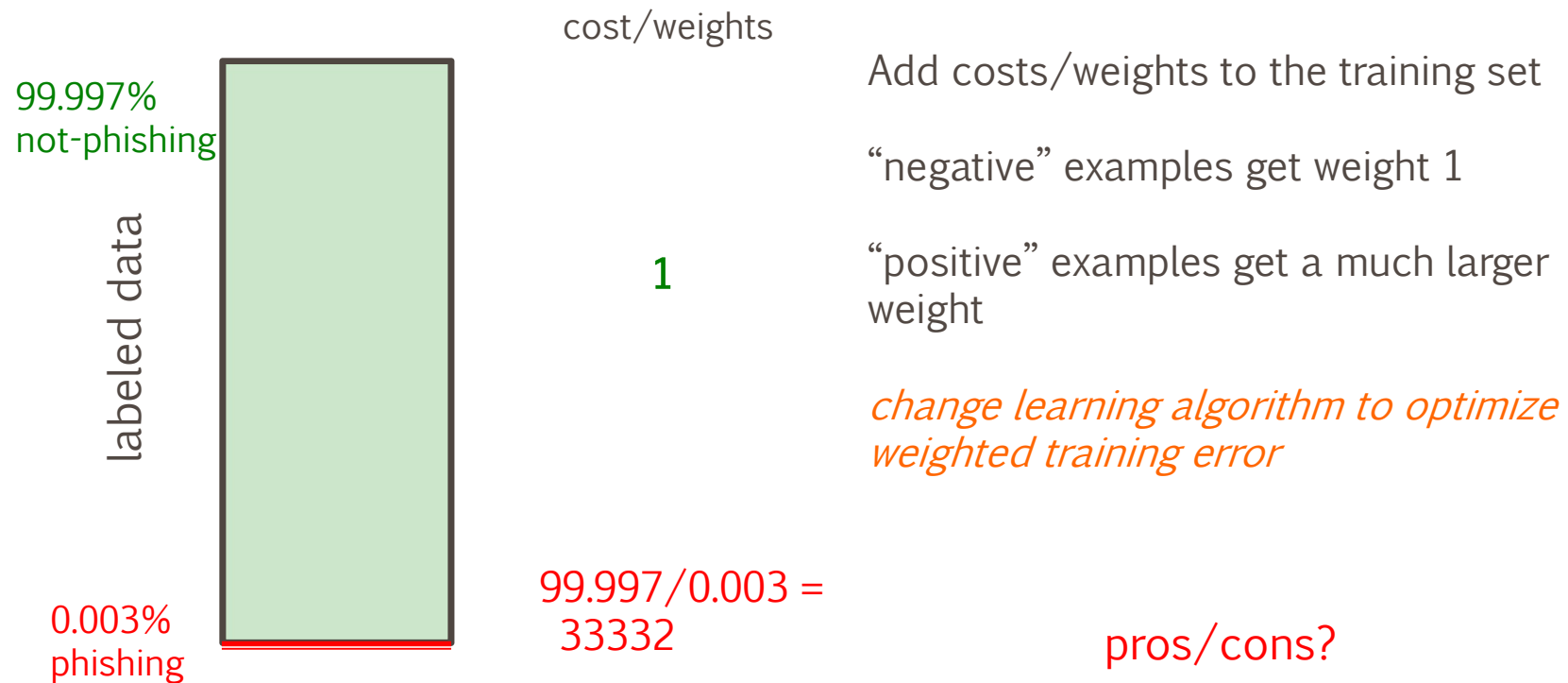
Pros:

- Easy to implement
- Utilizes all of the training data
- Tends to perform well in a broader set of circumstances than subsampling

Cons:

- Computationally expensive to train classifier

Idea 2b: weighted examples



weighted examples

- Pros:
 - Achieves the effect of oversampling without the computational cost
 - Utilizes all of the training data
 - Tends to perform well in a broader set circumstances
- Cons:
 - Requires a classifier that can deal with weights

Of our three classifiers, can all be modified to handle weights?

Building decision trees

- Otherwise:
 - calculate the “score” for each feature if we used it to split the data
 - pick the feature with the highest score, partition the data based on that data value and call recursively
-

We used the training error to decide on which feature to choose:

use the *weighted* training error

In general, any time we do a count, use the weighted count (e.g. in calculating the majority label at a leaf)

Idea 3: optimize a different error metric

- Train classifiers that try and optimize F1 measure or AUC or ...
- or, come up with another learning algorithm designed specifically for imbalanced problems
- pros/cons?

Idea 3: optimize a different error metric

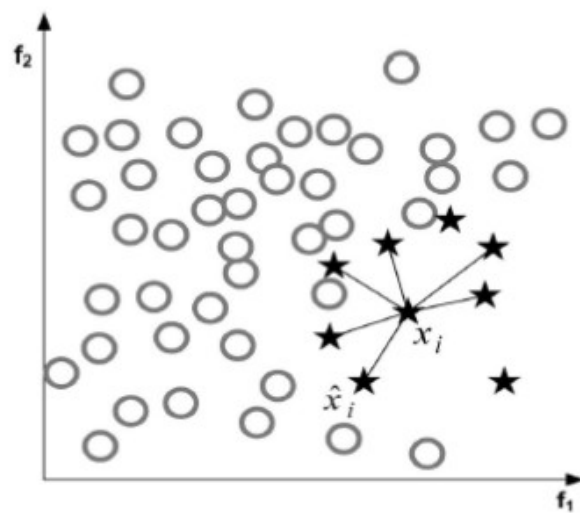
- Train classifiers that try and optimize F1 measure or AUC or ...
- Challenge: not all classifiers are amenable to this
- or, come up with another learning algorithm designed specifically for imbalanced problems
- Don't want to reinvent the wheel!
- That said, there are a number of approaches that have been developed to specifically handle imbalanced problems

Oversampling-SMOTE

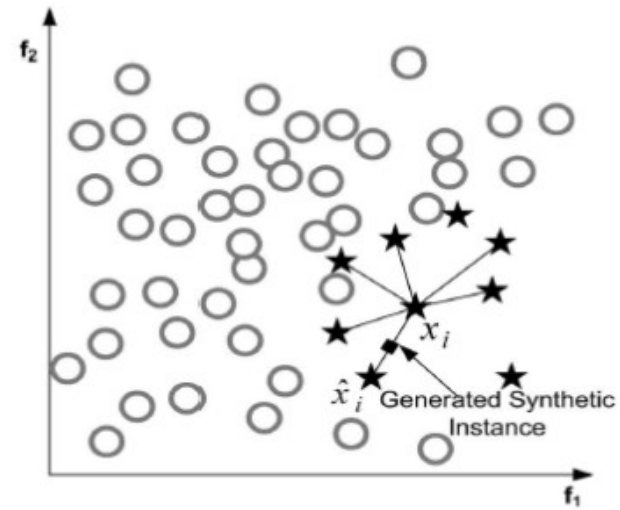
- SMOTE: A State-of-the-Art Resampling Approach
- SMOTE stands for Synthetic Minority Oversampling Technique.
- For each minority Sample
 - Find its k-nearest minority neighbors
 - Randomly select j of these neighbors
 - Randomly generate synthetic samples along the lines joining the minority sample and its j selected neighbors
 - (j depends on the amount of oversampling desired)

Oversampling-SMOTE

$$x_{new} = x + rand(0,1) \times (\tilde{x} - x)$$

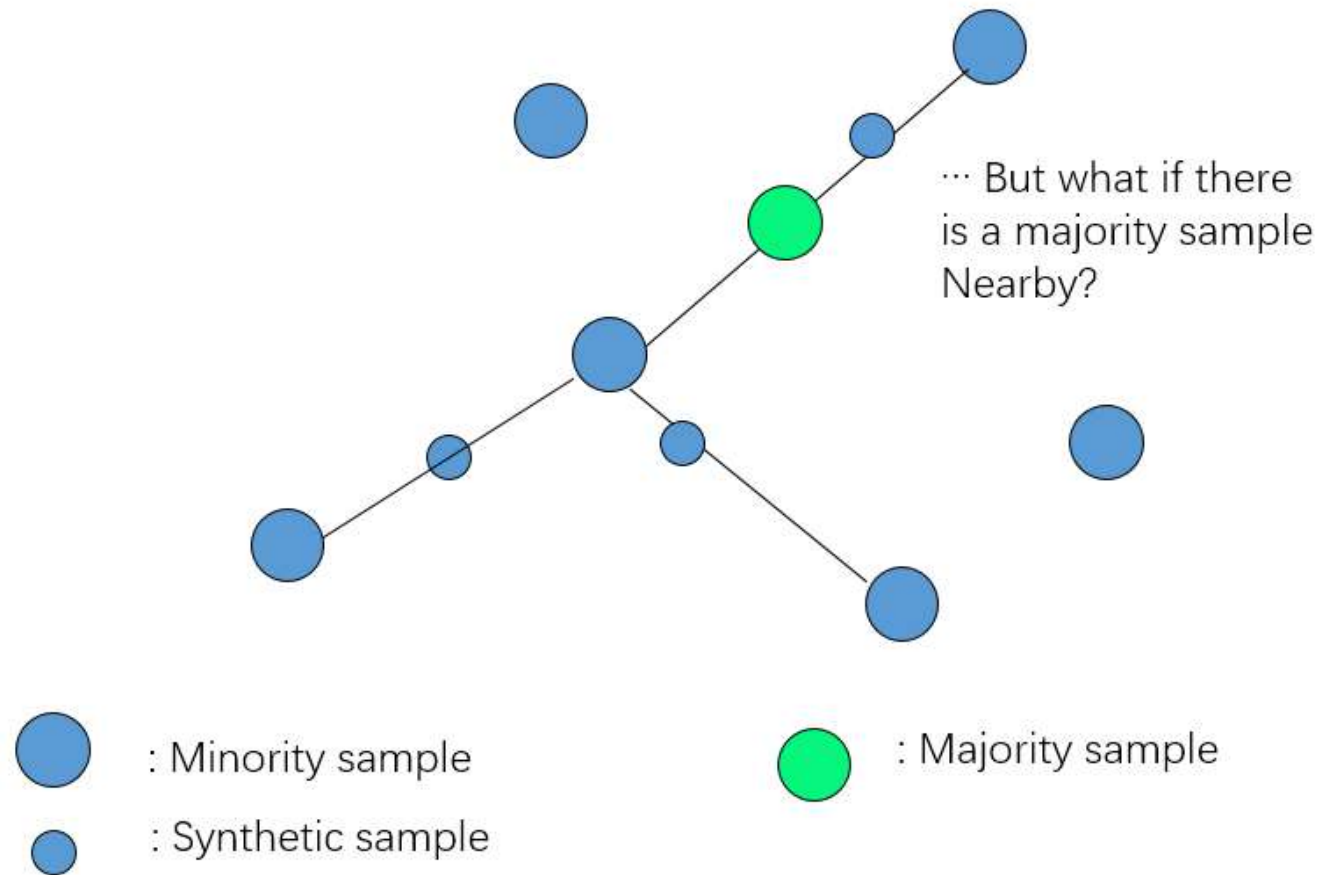


(a)

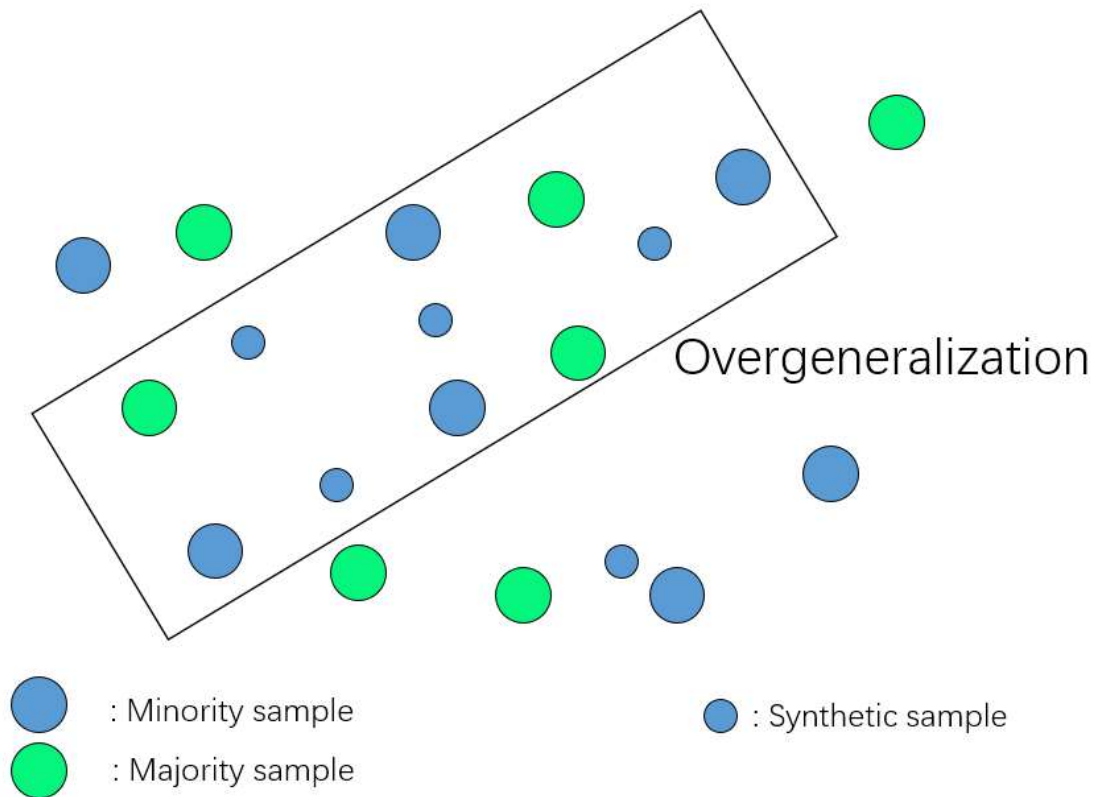


(b)

Oversampling-SMOTE



Overgeneralization



Oversampling-SMOTE

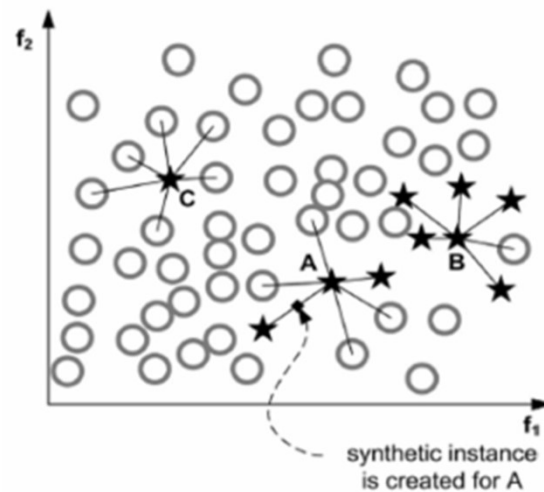
- Overgeneralization
 - SMOTE's procedure is inherently dangerous since it blindly generalizes the minority area without regard to the majority class.
 - This strategy is particularly problematic in the case of highly skewed class distributions since, in such cases, the minority class is very sparse with respect to the majority class, thus resulting in a greater chance of class mixture.
- Lack of Flexibility
 - The number of synthetic samples generated by SMOTE is fixed in advance, thus not allowing for any flexibility in the re-balancing rate.

Oversampling-Borderline SMOTE1

For each point p in S :

1. Compute its m nearest neighbors in T . Call this set M_p and let $m' = |M_p \cap L|$.
2. If $m' = m$, p is a noisy example. Ignore p and continue to the next point.
3. If $0 \leq m' \leq m/2$, p is safe. Ignore p and continue to the next point.
4. If $m/2 \leq m' \leq m$, add p to the set DANGER.

For each point d in DANGER, apply the SMOTE algorithm to generate synthetic examples.



Consider 6-nearest neighbor: $m=6$

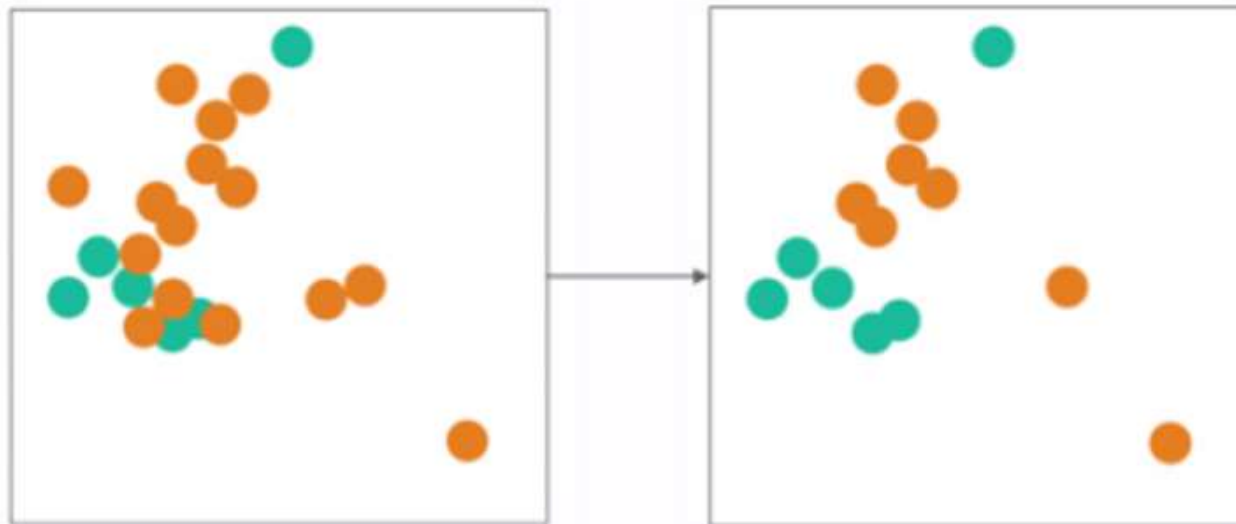
- | | | |
|---|---|-------------------|
| For A: number of minority instance: 2
number of majority instance: 4 | → | "DANGER" instance |
| For B: number of minority instance: 5
number of majority instance: 1 | → | "SAFE" instance |
| For C: number of minority instance: 0
number of majority instance: 6 | → | "NOISE" instance |

Oversampling-Borderline SMOTE2

- Similar with Borderline SMOTE2
- FOR p in DANGER:
 - Find k -NN samples S_k and L_k in S and L
 - Generate samples using SMOTE selecting α rate in S_k
 - Generate samples using SMOTE selecting $1 - \alpha$ rate in L_k

Undersampling-Edited Nearest Neighbor

- Delete those majority samples most of whose K neighbor is minor
- Repeated Edited Nearest Neighbor





HOW ABOUT MULTICLASS?

Multiclass classification

examples



label

Same setup where we have a set of features for each example



apple



orange



apple



banana



banana



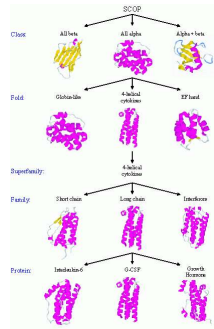
pineapple

real-world examples?

Real world multiclass classification



document classification



protein classification

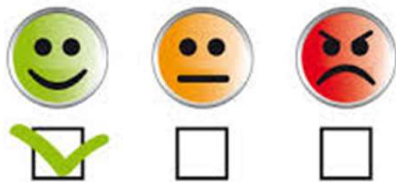


handwriting recognition



face recognition

most real-world applications tend to be multiclass



sentiment analysis

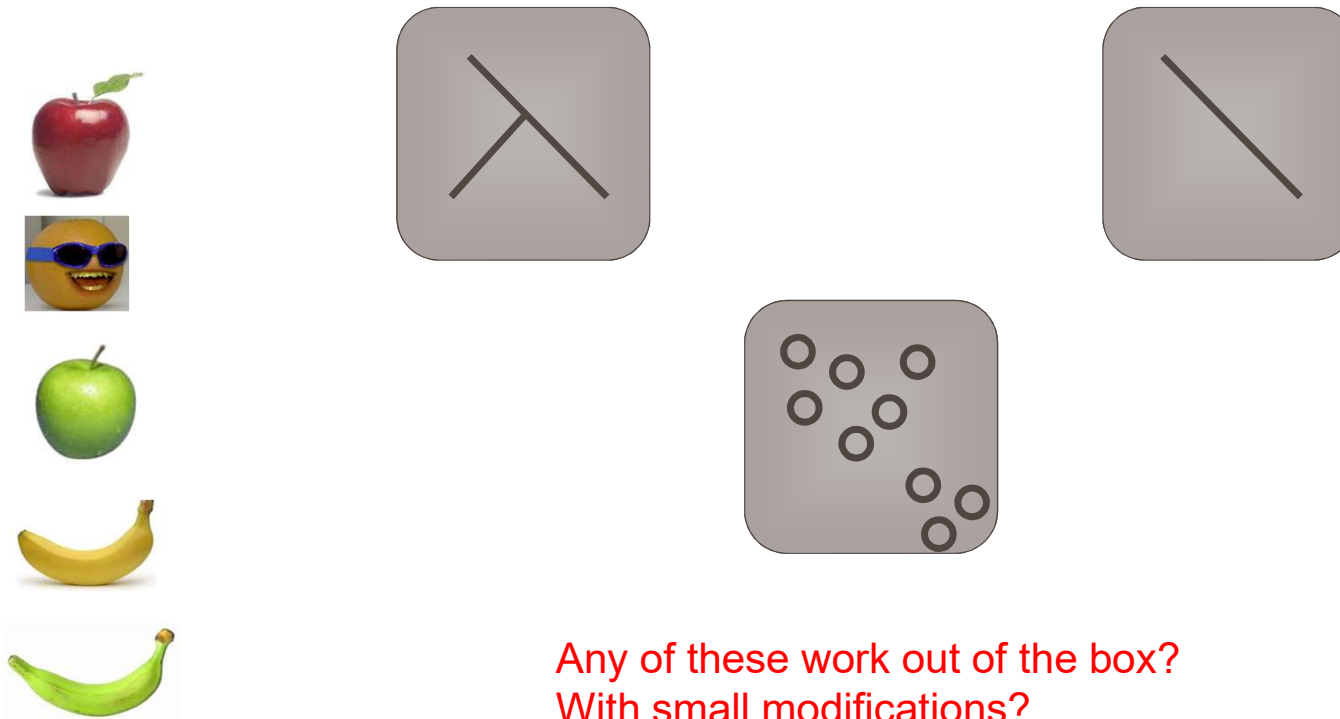


autonomous vehicles



emotion recognition

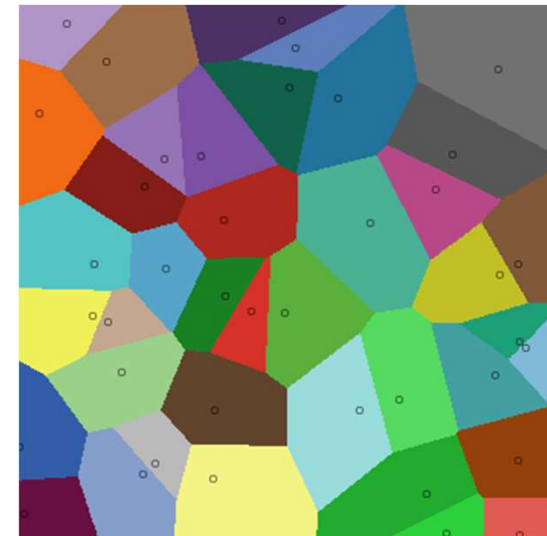
Multiclass: current classifiers



k-Nearest Neighbor (k-NN)

- To classify an example d :
 - Find k nearest neighbors of d
 - Choose as the label the majority label within the k nearest neighbors

No algorithmic changes!

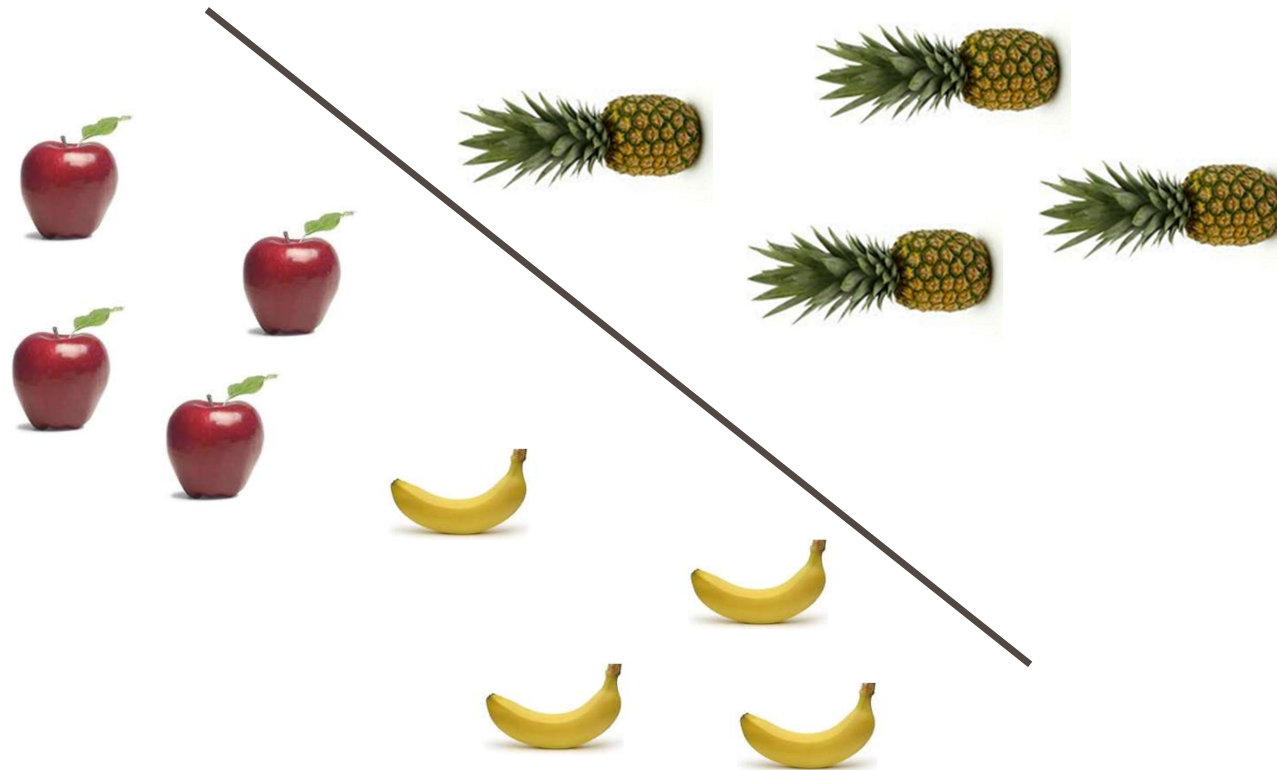


Decision Tree learning

- Base cases:
 - If all data belong to the same class, pick that label
 - If all the data have the same feature values, pick majority label
 - If we're out of features to examine, pick majority label
 - If the we don't have any data left, pick majority label of parent
 - If some other stopping criteria exists to avoid overfitting, pick majority label
- Otherwise:
 - calculate the “score” for each feature if we used it to split the data
 - pick the feature with the highest score, partition the data based on that data value and call recursively

No algorithmic changes!

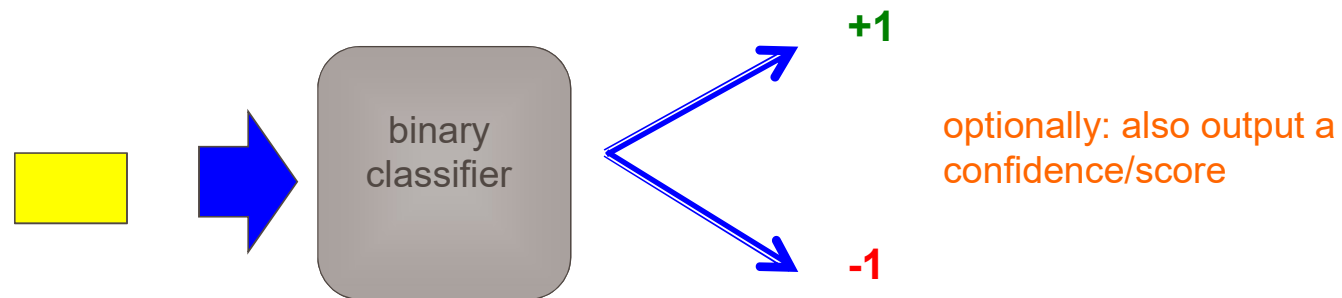
Perceptron learning



Hard to separate three classes with just one line ☹️

Black box approach to multiclass






















- Abstraction: we have a generic binary classifier, how can we use it to solve our new problem



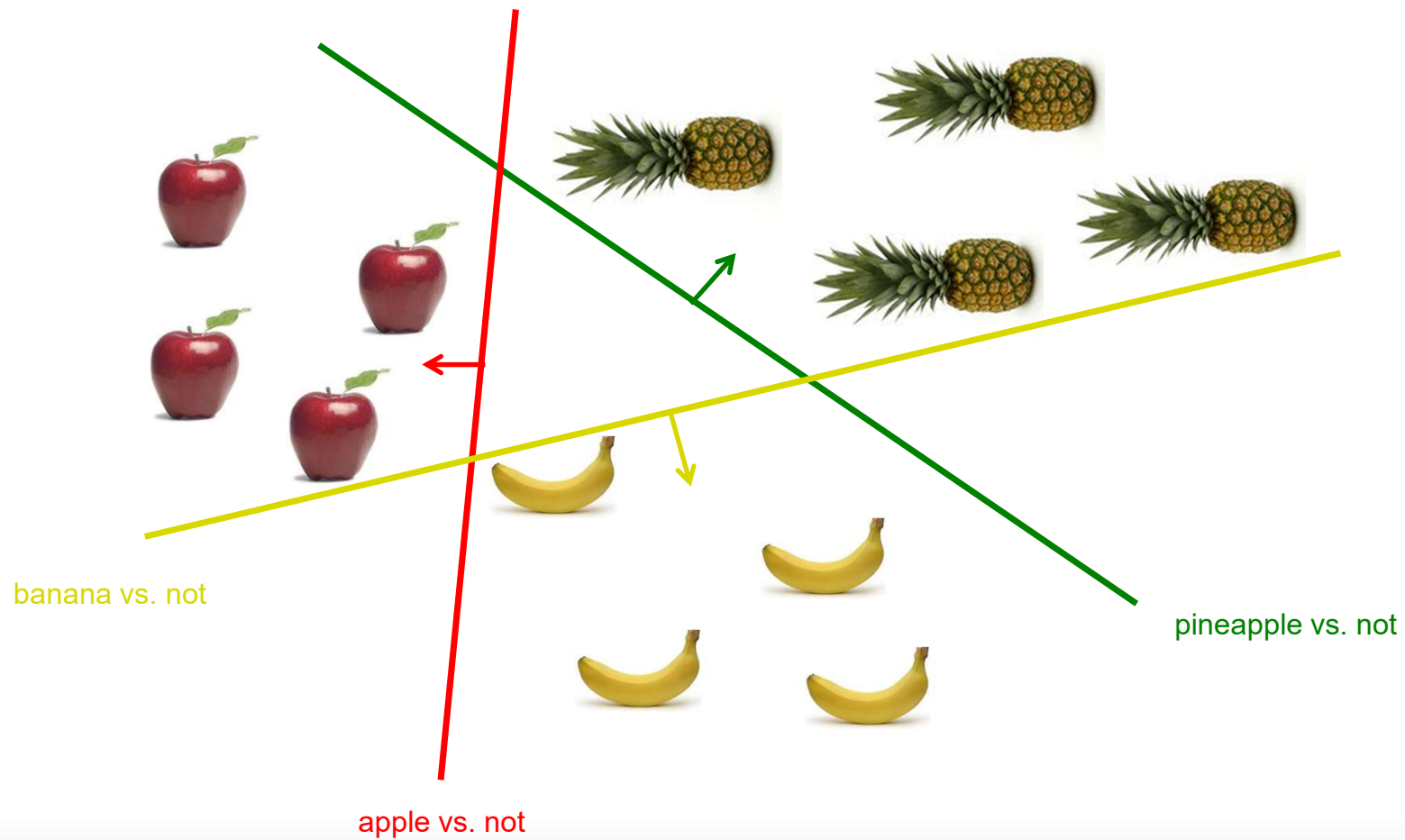
Can we solve our multiclass problem with this?

Approach 1: One vs. all (OVA)

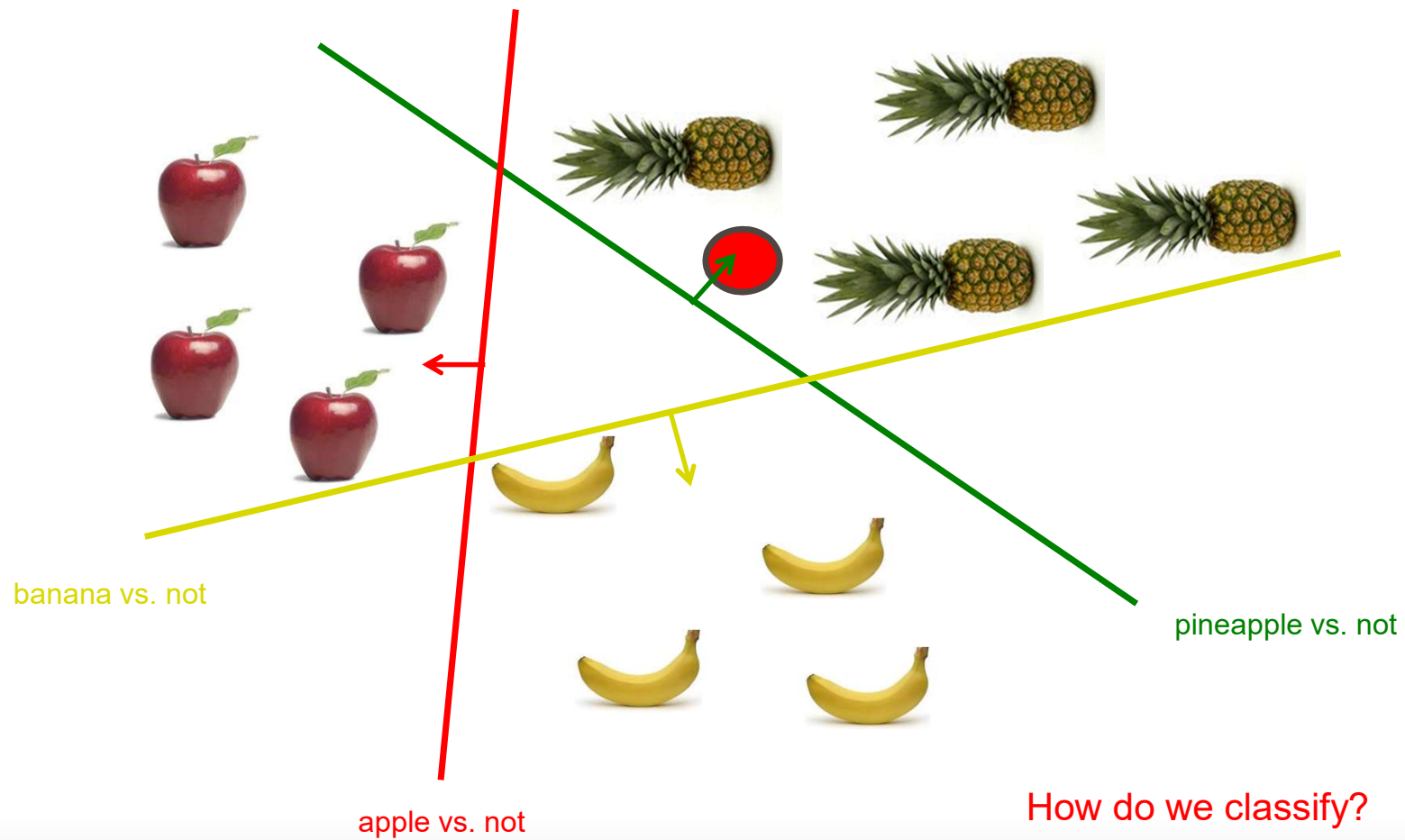
- Training: for each label L , pose as a binary problem
 - all examples with label L are positive
 - all other examples are negative

			apple vs. not		orange vs. not		banana vs. not
	apple		 +1		 -1		 -1
	orange		 -1		 +1		 -1
	apple		 +1		 -1		 -1
	banana		 -1		 -1		 +1
	banana		 -1		 -1		 +1

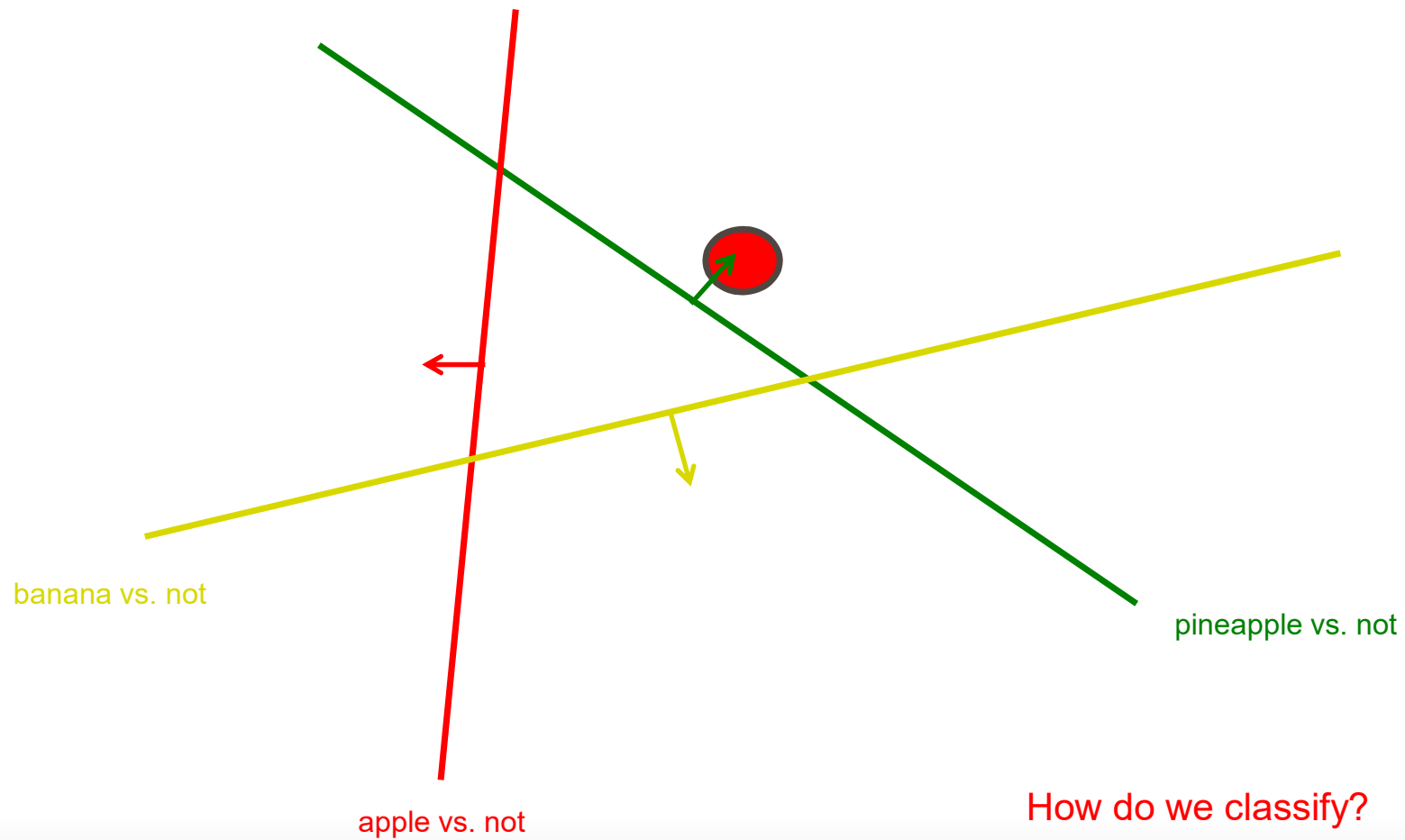
OVA: linear classifiers (e.g. perceptron)



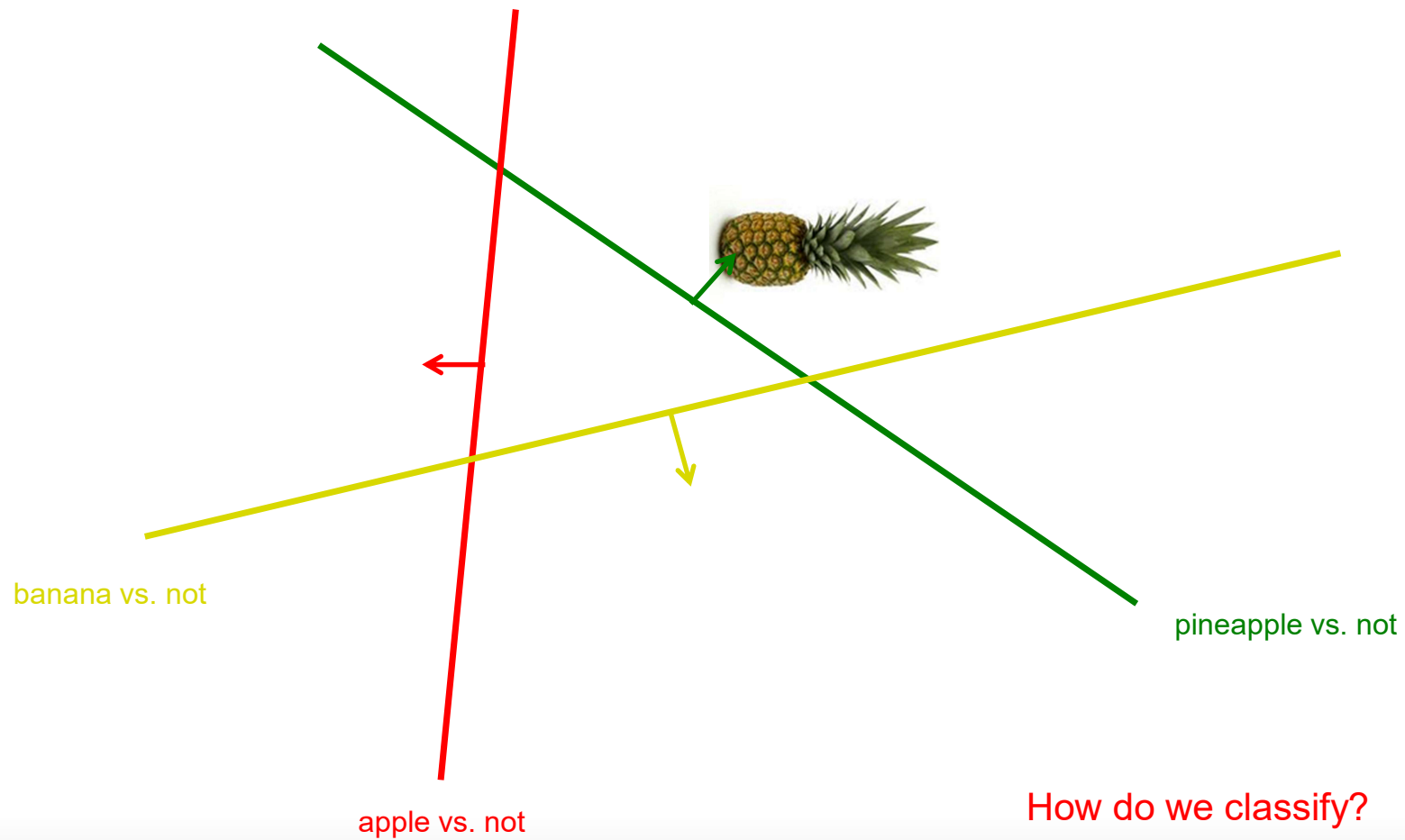
OVA: linear classifiers (e.g. perceptron)



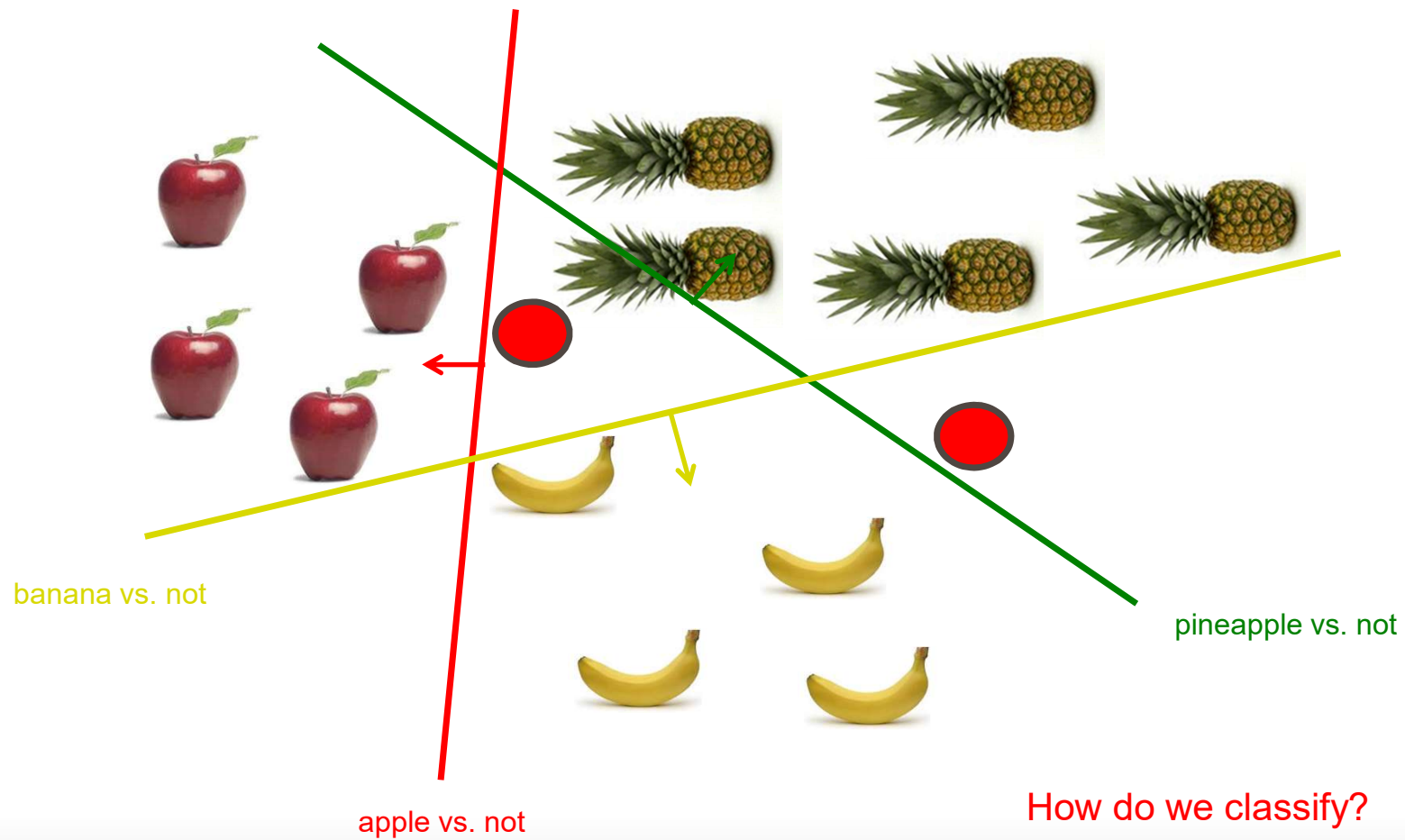
OVA: linear classifiers (e.g. perceptron)



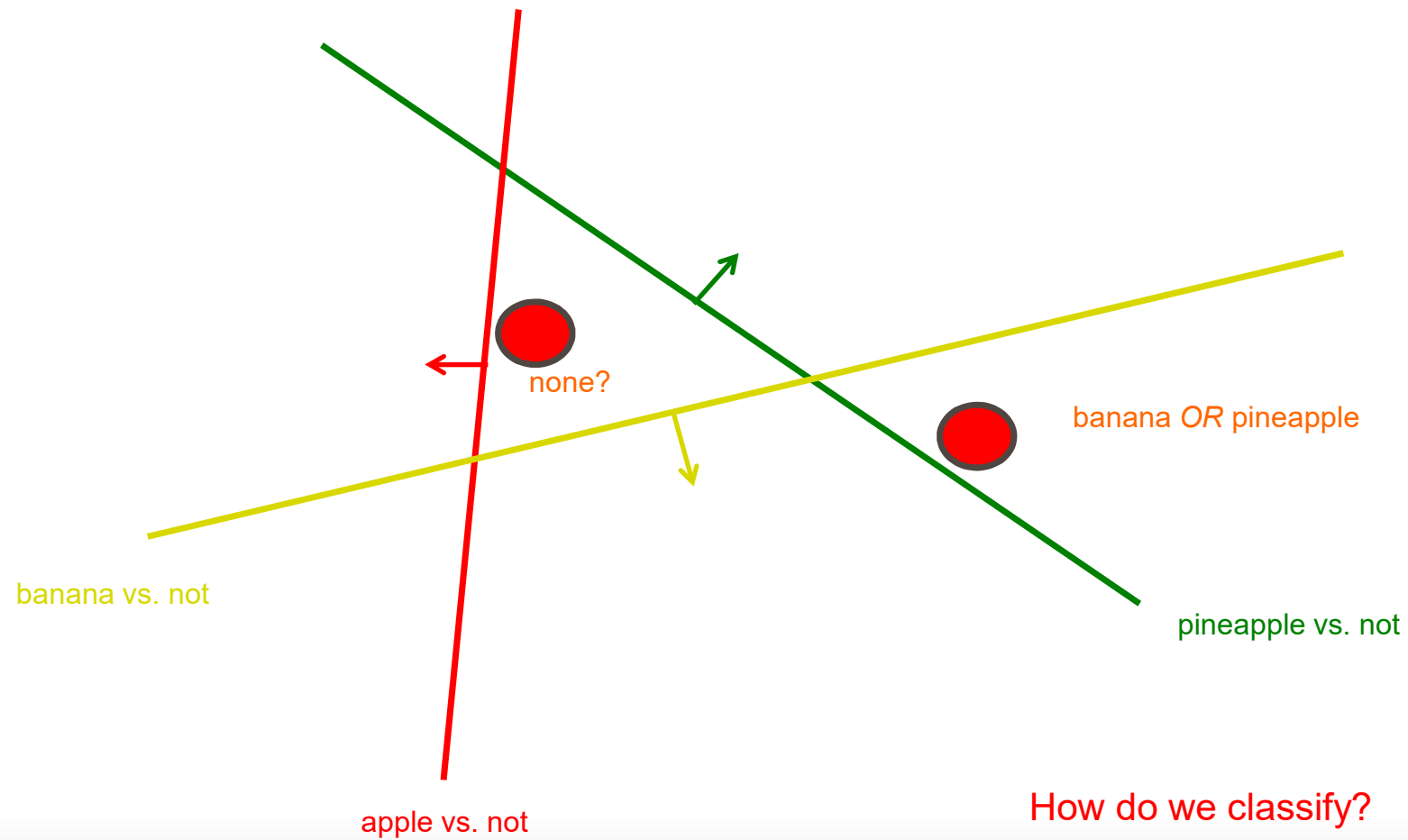
OVA: linear classifiers (e.g. perceptron)



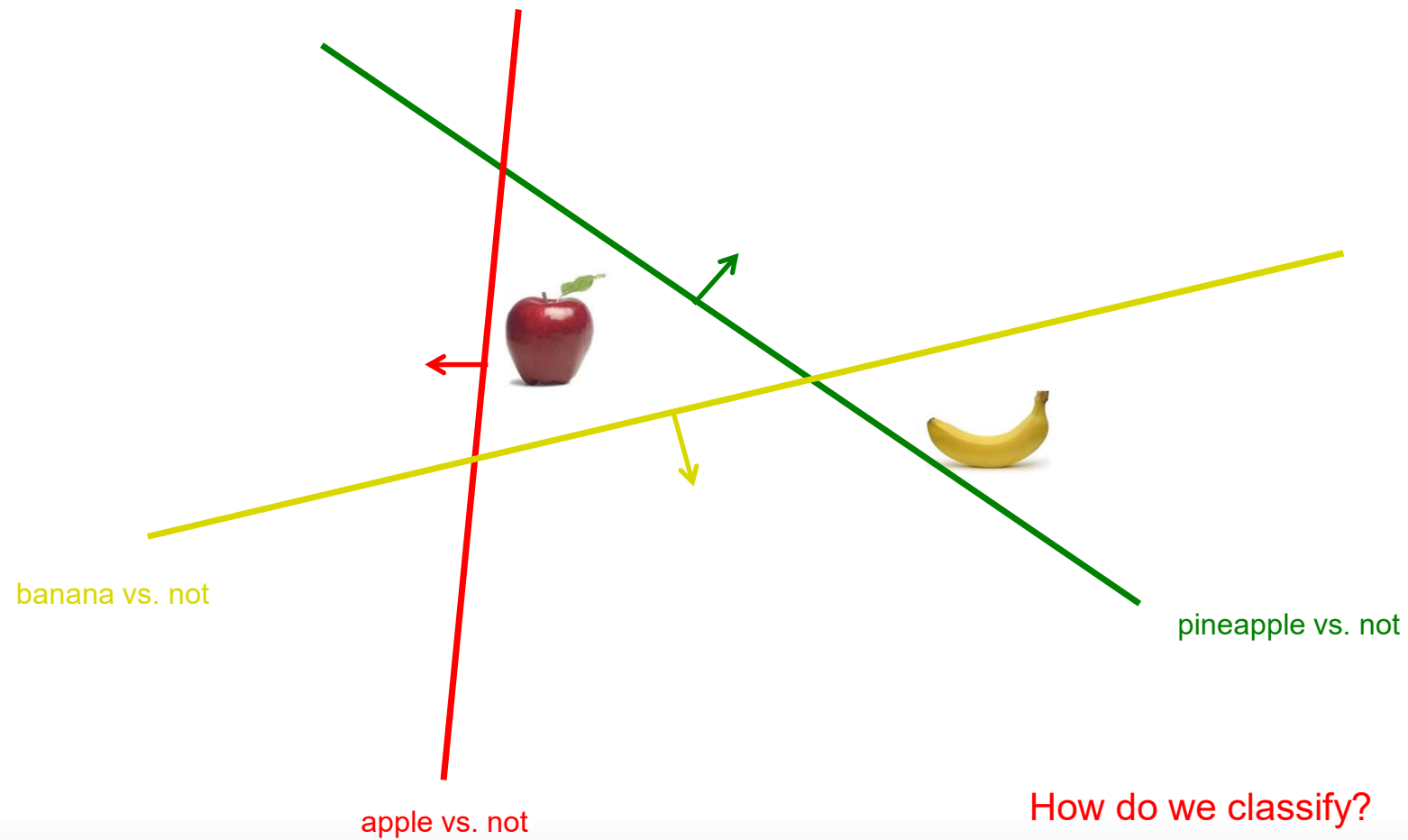
OVA: linear classifiers (e.g. perceptron)



OVA: linear classifiers (e.g. perceptron)



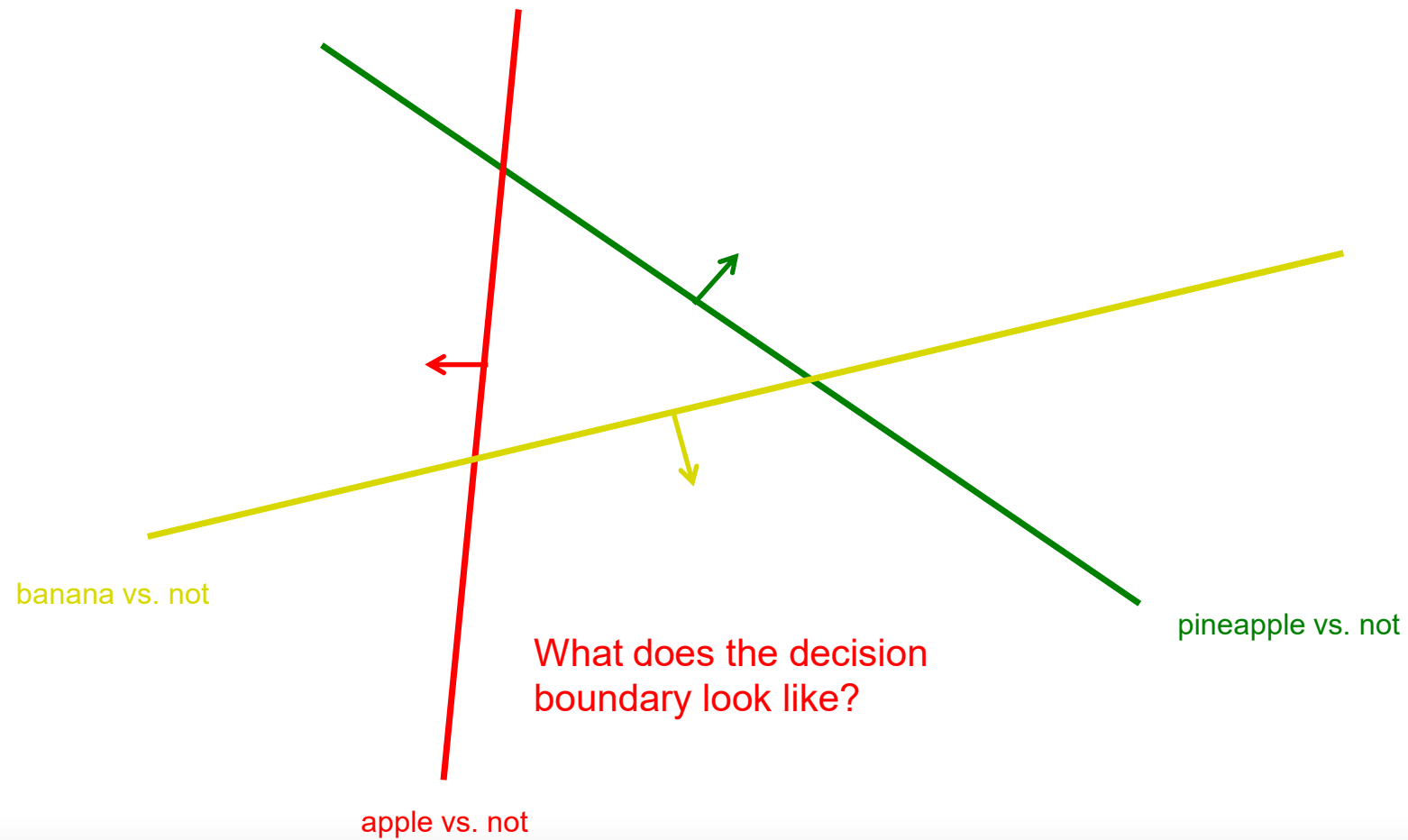
OVA: linear classifiers (e.g. perceptron)



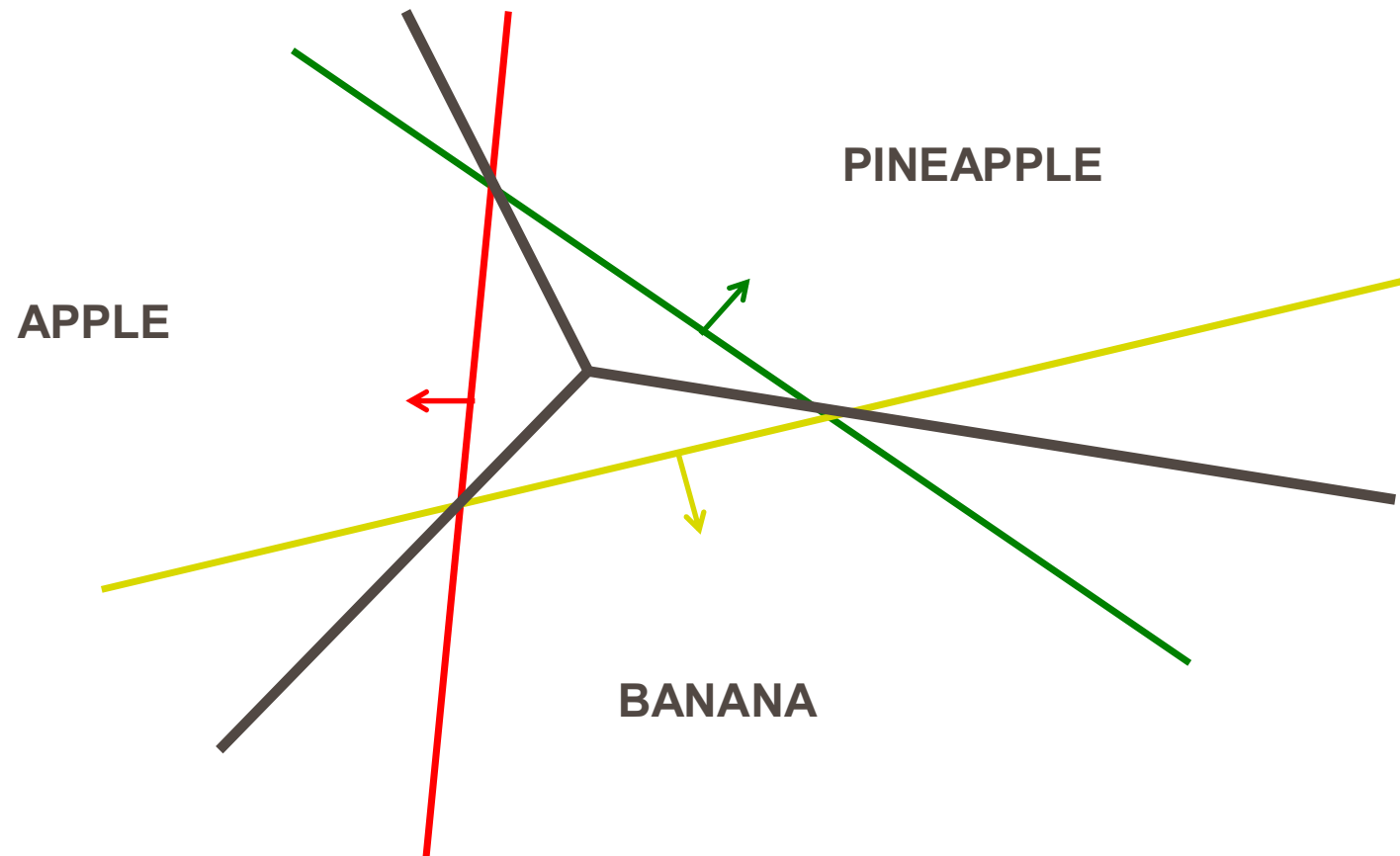
OVA: classify

- Classify:
 - If classifier doesn't provide confidence (this is rare) and there is ambiguity, pick one of the ones in conflict
 - Otherwise:
 - pick the most confident positive
 - if none vote positive, pick least confident negative

OVA: linear classifiers (e.g. perceptron)



OVA: linear classifiers (e.g. perceptron)



OVA: classify, perceptron

- Classify:
 - If classifier doesn't provide confidence (this is rare) and there is ambiguity, pick majority in conflict
 - Otherwise:
 - pick the most confident positive
 - if none vote positive, pick least confident negative

How do we calculate this for the perceptron?

OVA: classify, perceptron

- Classify:
 - If classifier doesn't provide confidence (this is rare) and there is ambiguity, pick majority in conflict
 - Otherwise:
 - pick the most confident positive
 - if none vote positive, pick least confident negative

$$\text{prediction} = b + \sum_{i=1}^n w_i f_i$$


Distance from the hyperplane

Approach 2: All vs. all (AVA)

- Training:
- For each pair of labels, train a classifier to distinguish between them

```
for i = 1 to number of labels:  
  for k = i+1 to number of labels:  
    train a classifier to distinguish between labelj and labelk:  
      - create a dataset with all examples with labelj labeled positive  
        and all examples with labelk labeled negative  
      - train classifier on this subset of the data
```

AVA training visualized

	apple
	orange
	apple
	banana
	banana

apple vs orange

	+1
	+1
	-1

orange vs banana

	+1
	-1
	-1

apple vs banana

	+1
	+1
	-1
	-1

AVA classify

apple vs orange



apple vs banana



orange vs banana



What class?

AVA classify

apple vs orange



orange

apple vs banana



apple

orange vs banana



orange



orange

In general?

AVA classify

To classify example e , classify with each classifier f_{jk}

We have a few options to choose the final class:

- Take a majority vote
- Take a weighted vote based on confidence
 - $y = f_{jk}(e)$
 - $\text{score}_j += y$
 - $\text{score}_k -= y$

How does this work?

*Here we're assuming that y encompasses both the prediction (+1,-1) and the confidence, i.e. $y = \text{prediction} * \text{confidence}$.*

AVA classify

- Take a weighted vote based on confidence
 - $y = f_{jk}(e)$
 - $\text{score}_j += y$
 - $\text{score}_k -= y$

If y is positive, classifier thought it was of type j :

- raise the score for j
- lower the score for k

if y is negative, classifier thought it was of type k :

- lower the score for j
- raise the score for k

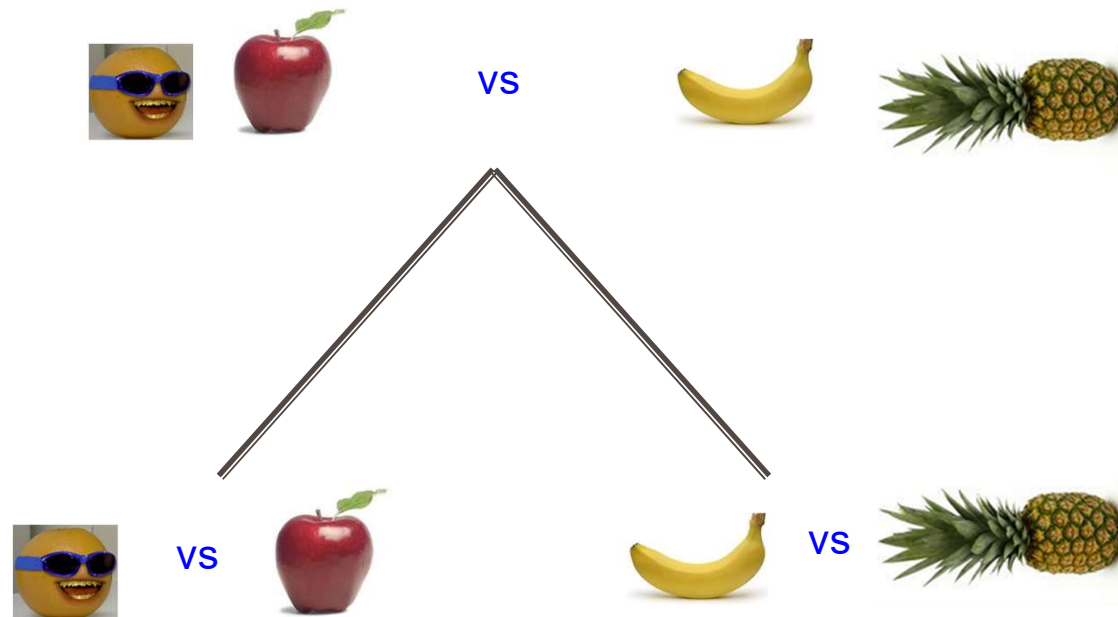
OVA vs. AVA

- Train/classify runtime?
- Error? Assume each binary classifier makes an error with probability ε

OVA vs. AVA

- Train time:
 - AVA learns more classifiers, however, they're trained on much smaller data this tends to make it faster if the labels are equally balanced
- Test time:
 - AVA has more classifiers
- Error (see the book for more justification):
 - AVA trains on more balanced data sets
 - AVA tests with more classifiers and therefore has more chances for errors
 - Theoretically:
 - OVA: ϵ (number of labels -1)
 - AVA: 2ϵ (number of labels -1)

Approach 3: Divide and conquer









Pros/cons vs. AVA?

Multiclass summary







- If using a binary classifier, the most common thing to do is OVA
- Otherwise, use a classifier that allows for multiple labels:
 - DT and k-NN work reasonably well
 - We'll see a few more in the coming weeks that will often work better

Multiclass evaluation

	label	prediction
	apple	orange
	orange	orange
	apple	apple
	banana	pineapple
	banana	banana
	pineapple	pineapple






How should we evaluate?

Multiclass evaluation

	label	prediction
	apple	orange
	orange	orange
	apple	apple
	banana	pineapple
	banana	banana
	pineapple	pineapple

Accuracy: 4/6

Multiclass evaluation imbalanced data

	label	prediction
	apple	orange
	...	
	apple	apple
	banana	pineapple
	banana	banana
	pineapple	pineapple

Any problems?

Data imbalance!







Macroaveraging vs. microaveraging

- microaveraging: average over examples (this is the “normal” way of calculating)
- macroaveraging: calculate evaluation score (e.g. accuracy) for each label, then average over labels

What effect does this have?
Why include it?






- Puts more weight/emphasis on rarer labels
- Allows another dimension of analysis

Macroaveraging vs. microaveraging

	label	prediction
	apple	orange
	orange	orange
	apple	apple
	banana	pineapple
	banana	banana
	pineapple	pineapple

- **microaveraging**: average over examples
- **macroaveraging**: calculate evaluation score (e.g. accuracy) for each label, then average over labels

Macroaveraging vs. microaveraging

	label	prediction
	apple	orange
	orange	orange
	apple	apple
	banana	pineapple
	banana	banana
	pineapple	pineapple

▪ **microaveraging:** 4/6

▪ **macroaveraging:**

- apple = 1/2
- orange = 1/1
- banana = 1/2
- pineapple = 1/1
- total = $(1/2 + 1 + 1/2 + 1)/4 = 3/4$

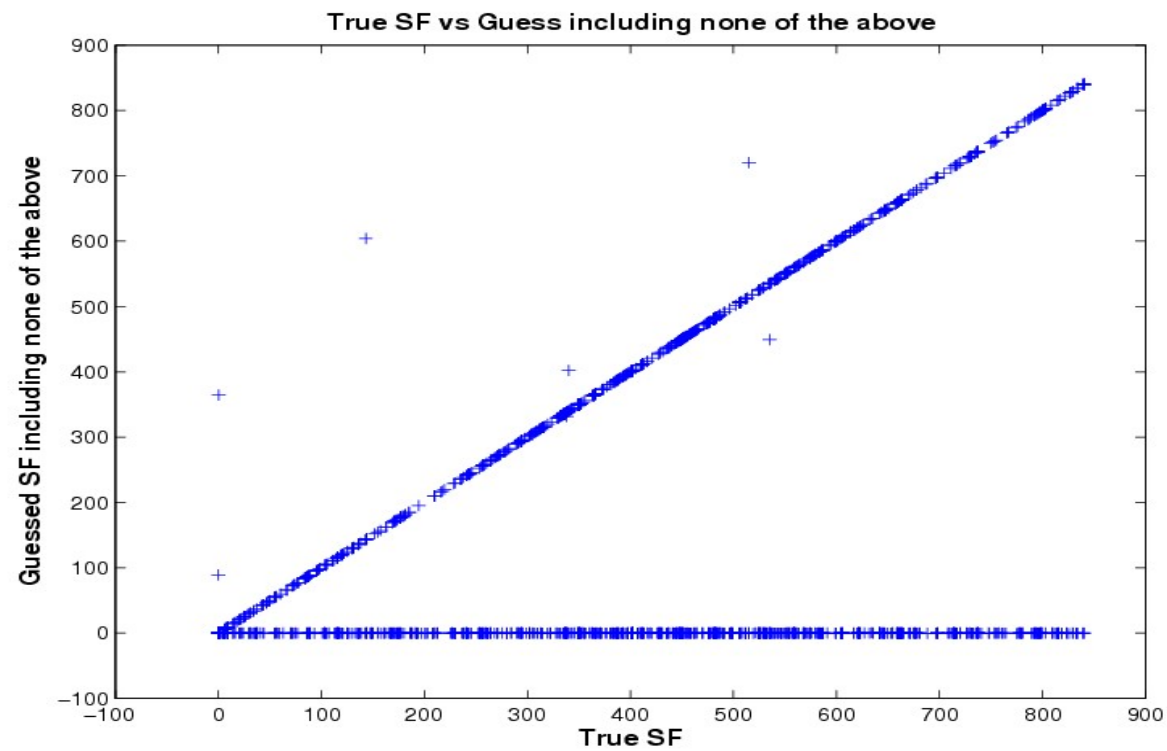
Confusion matrix

entry (i, j) represents the number of examples with label i that were predicted to have label j

another way to understand both the data and the classifier

	Classic	Country	Disco	Hiphop	Jazz	Rock
Classic	86	2	0	4	18	1
Country	1	57	5	1	12	13
Disco	0	6	55	4	0	5
Hiphop	0	15	28	90	4	18
Jazz	7	1	0	0	37	12
Rock	6	19	11	0	27	48

Confusion matrix



BLAST classification of proteins in 850 superfamilies

Multilabel vs. multiclass classification

- Is it edible?
- Is it sweet?
- Is it a fruit?
- Is it a banana?

Is it a banana?
Is it an apple?
Is it an orange?
Is it a pineapple?

Is it a banana?
Is it yellow?
Is it sweet?
Is it round?

Any difference in these labels/categories?

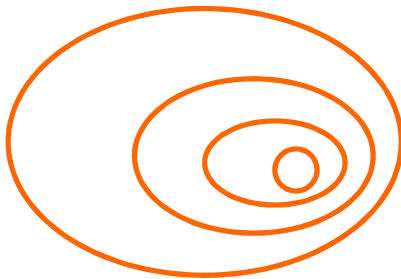
Multilabel vs. multiclass classification

- Is it edible?
- Is it sweet?
- Is it a fruit?
- Is it a banana?

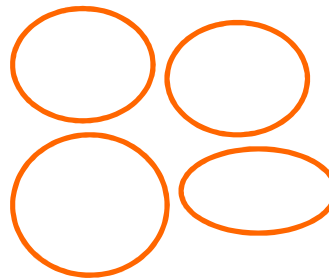
Is it a banana?
Is it an apple?
Is it an orange?
Is it a pineapple?

Is it a banana?
Is it yellow?
Is it sweet?
Is it round?

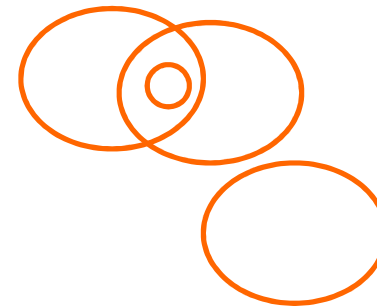
Different structures



Nested/ Hierarchical



Exclusive/ Multiclass



General/Structured

Multiclass vs. multilabel

- Multiclass: each example has one label and exactly one label
- Multilabel: each example has zero or more labels. Also called annotation

Multilabel applications?

Multilabel

- Image annotation
- Document topics
- Labeling people in a picture
- Medical diagnosis

Multiclass vs. multilabel

- Multiclass: each example has one label and exactly one label
- Multilabel: each example has zero or more labels. Also called annotation

Which of our approaches work for multilabel?



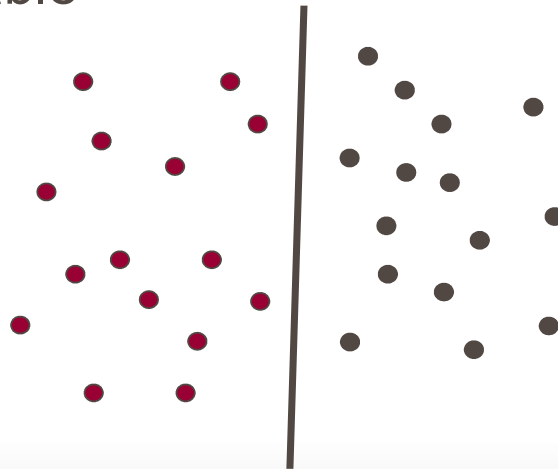
GRADIENT DESCENT

Linear models

A strong high-bias assumption is *linear separability*:

- in 2 dimensions, can separate classes by a line
- in higher dimensions, need hyperplanes

A *linear model* is a model that assumes the data is linearly separable



Linear models

A linear model in n -dimensional space (i.e. n features) is define by $n+1$ weights:

In two dimensions, a line: $0 = w_1 f_1 + w_2 f_2 + b$ (where $b = -a$)

In three dimensions, a plane: $0 = w_1 f_1 + w_2 f_2 + w_3 f_3 + b$

In m -dimensions, a *hyperplane* $0 = b + \sum_{j=1}^m w_j f_j$



Perceptron learning algorithm

repeat until convergence (or for some # of iterations):

for each training example $(f_1, f_2, \dots, f_m, \text{label})$:

$$\text{prediction} = b + \sum_{j=1}^m w_j f_j$$

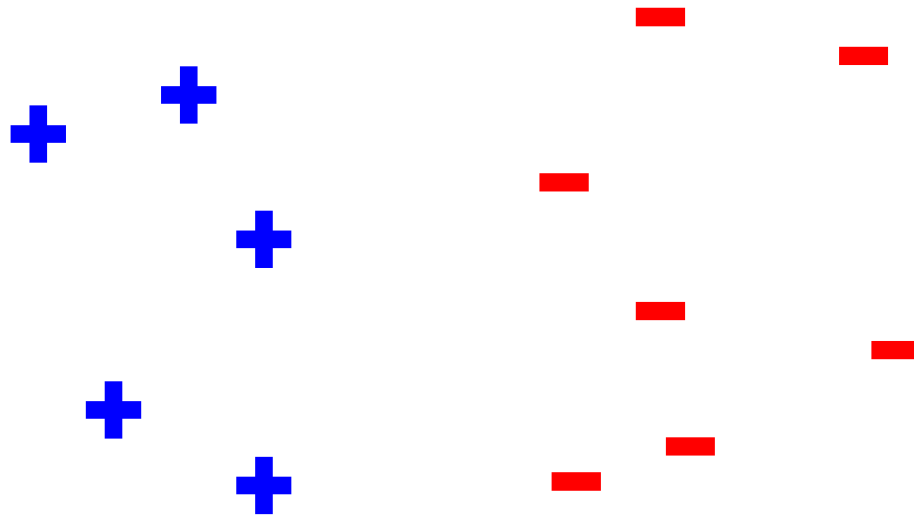
if $\text{prediction} * \text{label} \leq 0$: // they don't agree

for each w_j :

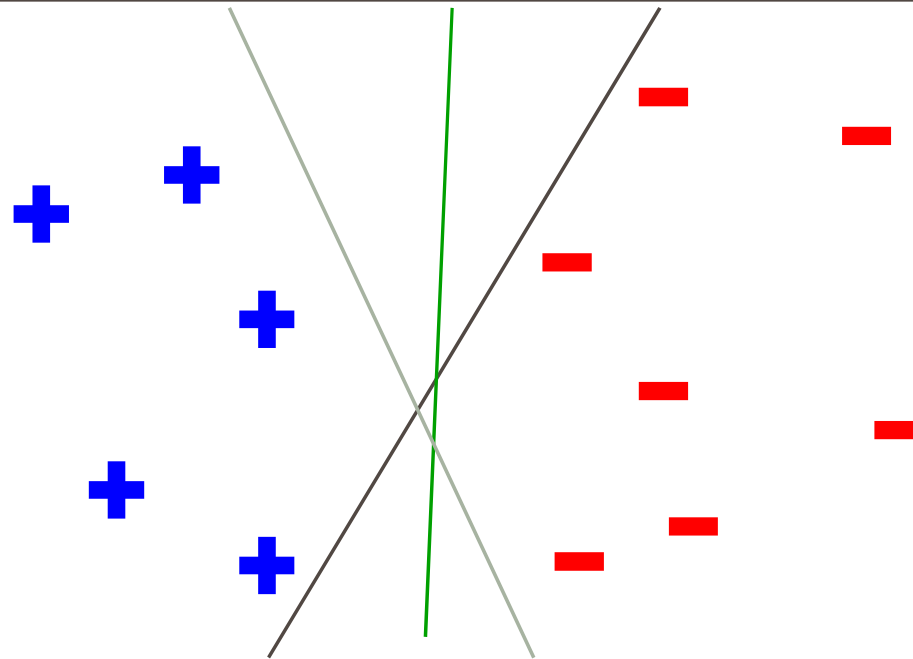
$$w_j = w_j + f_j * \text{label}$$

$$b = b + \text{label}$$

Which line will it find?



Which line will it find?



Only guaranteed to find
some line that separates
the data

Linear models

- Perceptron algorithm is one example of a linear classifier
- Many, many other algorithms that learn a line (i.e. a setting of a linear combination of weights)
- Goals:
 - Explore a number of linear training algorithms
 - Understand **why these algorithms work**

Perceptron learning algorithm

repeat until convergence (or for some # of iterations):

for each training example $(f_1, f_2, \dots, f_m, \text{label})$:

$$\text{prediction} = b + \sum_{j=1}^m w_j f_j$$

if $\text{prediction} * \text{label} \leq 0$: // they don't agree

for each w_i :

$$w_i = w_i + f_i * \text{label}$$

$$b = b + \text{label}$$

A closer look at why we got it wrong

w_1 w_2

$(-1, -1, \text{positive})$

$$0 * f_1 + 1 * f_2 =$$

$$0 * -1 + 1 * -1 = -1$$

We'd like this value to be positive
since it's a positive value

didn't contribute,
but could have

contributed in the
wrong direction

decrease

$0 \rightarrow -1$

decrease

$1 \rightarrow 0$

Intuitively these make sense
Why change by 1?
Any other way of doing it?

Model-based machine learning

1. pick a model
 - e.g. a hyperplane, a decision tree,...
 - A model is defined by a collection of parameters



What are the parameters for DT? Perceptron?

Model-based machine learning

1. pick a model
 - e.g. a hyperplane, a decision tree,...
 - A model is defined by a collection of parameters
2. pick a criteria to optimize (aka objective function)

What criterion do decision tree learning and perceptron learning optimize?

Model-based machine learning

1. pick a model
 - e.g. a hyperplane, a decision tree,...
 - A model is defined by a collection of parameters
2. pick a criteria to optimize (aka objective function)
 - e.g. training error
3. develop a learning algorithm
 - the algorithm should try and minimize the criteria
 - sometimes in a heuristic way (i.e. non-optimally)
 - sometimes explicitly

Linear models in general

1. pick a model

$$0 = \textcircled{b} + \sum_{j=1}^m \textcircled{w_j} f_j$$

These are the parameters we want to learn

2. pick a criteria to optimize (aka objective function)

Some notation: indicator function

$$1[x] = \begin{cases} 1 & \text{if } x = \text{True} \\ 0 & \text{if } x = \text{False} \end{cases}$$

Convenient notation for turning T/F answers into numbers/counts:

$$\text{drinks_to_bring_for_class} = \sum_{x \in \text{class}} 1[x \geq 21]$$

Some notation: dot-product

Sometimes it is convenient to use **vector notation**

We represent an example f_1, f_2, \dots, f_m as a single vector, x

Similarly, we can represent the weight vector w_1, w_2, \dots, w_m as a single vector, w

The **dot-product** between two vectors a and b is defined as:

$$a \cdot b = \sum_{j=1}^m a_j b_j$$

Linear models

1. pick a model

$$0 = \textcircled{b} + \sum_{j=1}^n \textcircled{w_j} f_j$$

These are the parameters we want to learn

2. pick a criteria to optimize (aka objective function)

$$\sum_{i=1}^n 1[y_i(w \cdot x_i + b) \leq 0]$$

What does this equation say?

0/1 loss function

$$\sum_{i=1}^n 1[y_i(w \cdot x_i + b) \leq 0]$$

$$\text{distance} = b + \sum_{j=1}^m w_j x_j = w \cdot x + b$$

distance from hyperplane

$$\text{incorrect} = y_i(w \cdot x_i + b) \leq 0$$

whether or not the prediction
and label agree

$$0/1 \text{ loss} = \sum_{i=1}^n 1[y_i(w \cdot x_i + b) \leq 0]$$

total number of mistakes, aka
0/1 loss

Model-based machine learning

1. pick a model $0 = b + \sum_{j=1}^m w_j f_j$

2. pick a criteria to optimize (aka objective function)

$$\sum_{i=1}^n 1[y_i(w \cdot x_i + b) \leq 0]$$

3. develop a learning algorithm

$$\operatorname{argmin}_{w,b} \sum_{i=1}^n 1[y_i(w \cdot x_i + b) \leq 0]$$

Find w and b that minimize the 0/1 loss

Minimizing 0/1 loss

$$\operatorname{argmin}_{w,b} \sum_{i=1}^n 1[y_i(w \cdot x_i + b) \leq 0]$$

Find w and b that
minimize the 0/1 loss

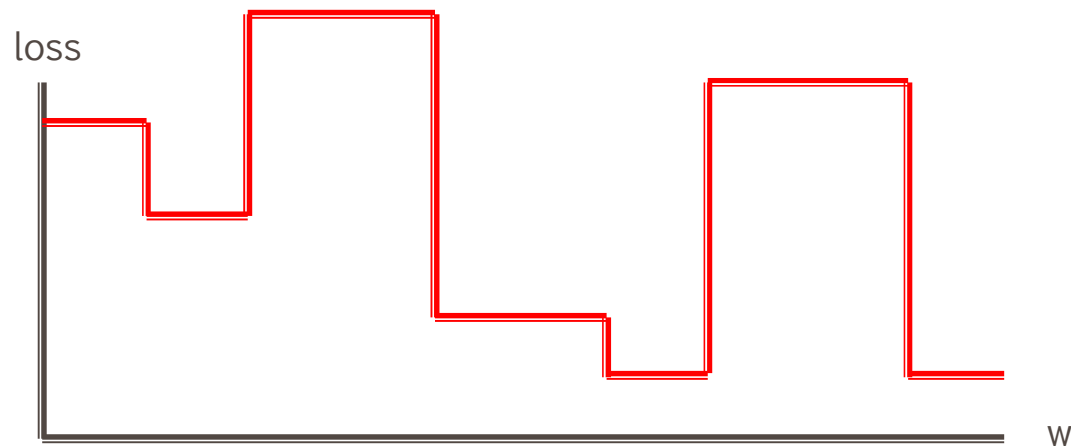
How do we do this?

How do we *minimize* a function?

Why is it hard for this function?

Minimizing 0/1 in one dimension

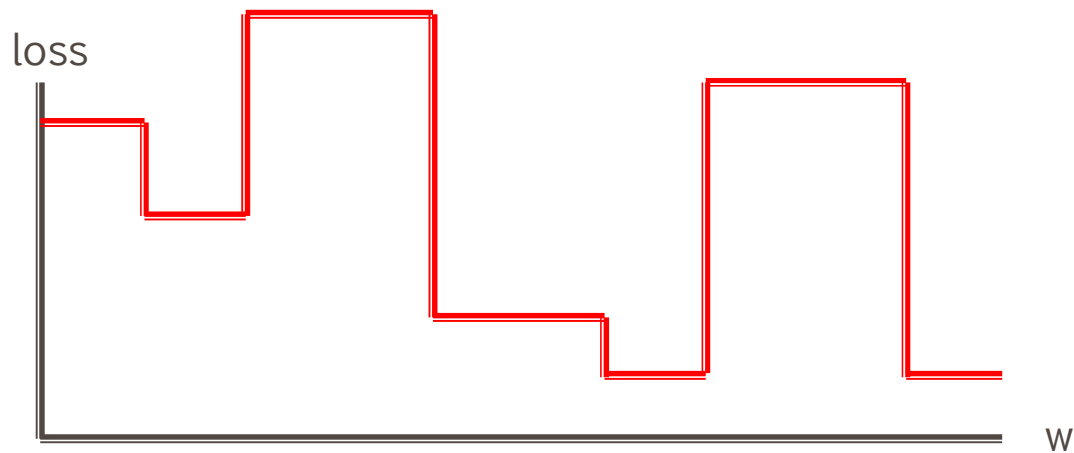
$$\sum_{i=1}^n 1[y_i(w \cdot x_i + b) \leq 0]$$



Each time we change w such that the example is right/wrong the loss will increase/decrease

Minimizing 0/1 over all w

$$\sum_{i=1}^n 1[y_i(w \cdot x_i + b) \leq 0]$$



Each new feature we add (i.e. weights) adds another dimension to this space!

Minimizing 0/1 loss

$$\operatorname{argmin}_{w,b} \sum_{i=1}^n 1[y_i(w \cdot x_i + b) \leq 0]$$

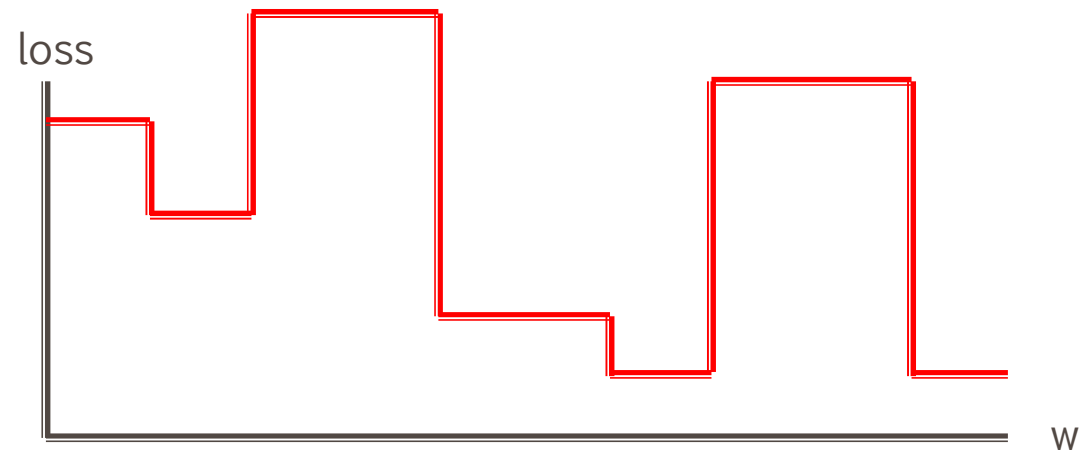
Find w and b that
minimize the 0/1 loss

This turns out to be hard (in fact, NP-HARD ☹)

Challenge:

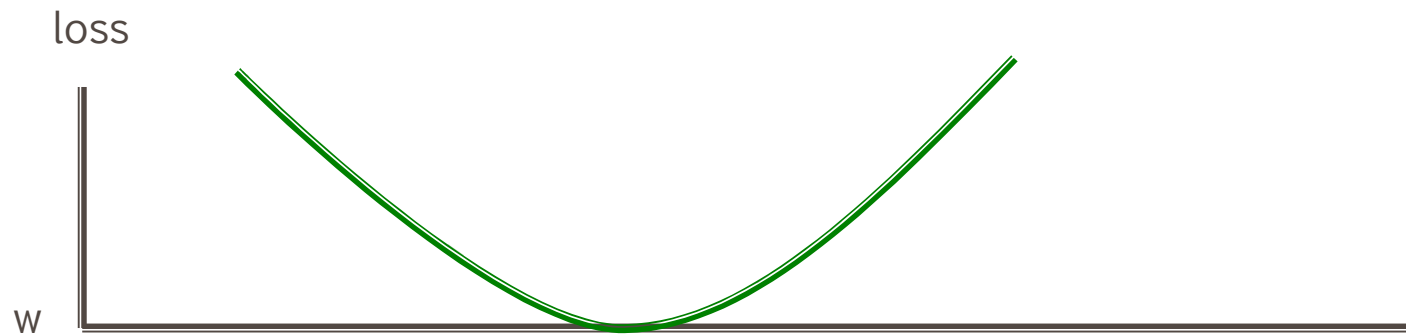
- small changes in any w can have large changes in the loss (the change isn't continuous)
- there can be many, many local minima
- at any give point, we don't have much information to direct us towards any minima

More manageable loss functions



What property/properties do we want from our loss function?

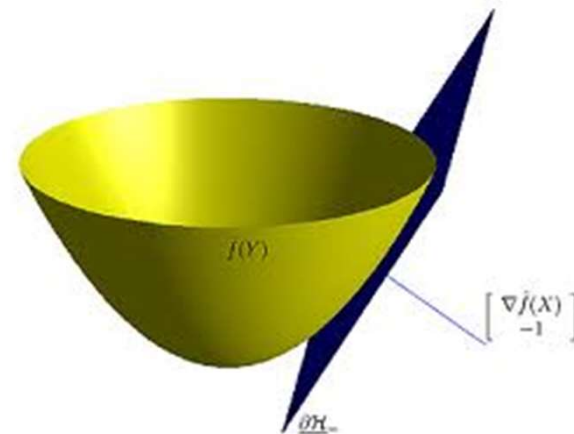
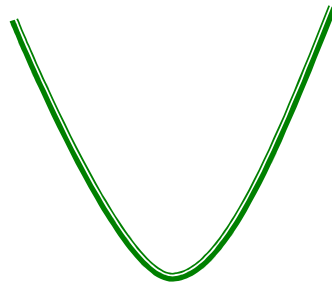
More manageable loss functions



- Ideally, continuous (i.e. differentiable) so we get an indication of direction of minimization
- Only one minima

Convex functions

Convex functions look something like:



One definition: The line segment between any two points on the function is *above* the function

Surrogate loss functions

- For many applications, we really would like to minimize the 0/1 loss
- A **surrogate/auxiliary loss function** is a loss function that provides an upper bound on the actual loss function (in this case, 0/1)
- We'd like to identify convex surrogate loss functions to make them easier to minimize
- Key to a loss function is how it scores the difference between the actual label y and the predicted label y'

Surrogate loss functions

0/1 loss: $l(y, y') = 1[y y' \leq 0]$

Ideas?

Some function that is a proxy for error,
but is continuous and convex

Surrogate loss functions

0/1 loss: $l(y, y') = 1[y y' \leq 0]$

Hinge: $l(y, y') = \max(0, 1 - y y')$

Exponential: $l(y, y') = \exp(-y y')$

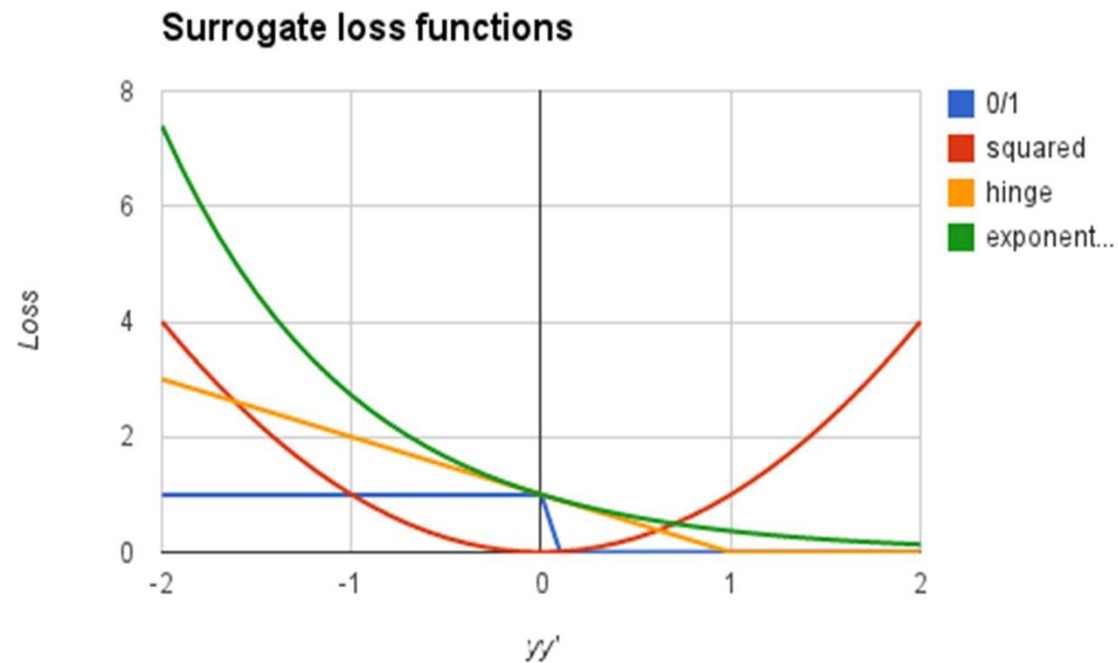
Squared loss: $l(y, y') = (y - y')^2$

Why do these work? What do they penalize?

Surrogate loss functions

0/1 loss: $l(y, y') = 1[y y' \leq 0]$ Hinge: $l(y, y') = \max(0, 1 - y y')$

Squared loss: $l(y, y') = (y - y')^2$ Exponential: $l(y, y') = \exp(-y y')$



Model-based machine learning

1. pick a model

$$0 = b + \sum_{j=1}^m w_j f_j$$

2. pick a criteria to optimize (aka objective function)

$$\sum_{i=1}^n \exp(-y_i(w \cdot x_i + b))$$

use a convex surrogate
loss function

3. develop a learning algorithm

$$\operatorname{argmin}_{w,b} \sum_{i=1}^n \exp(-y_i(w \cdot x_i + b))$$

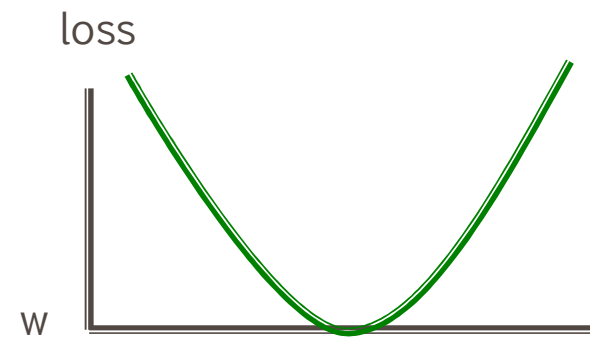
Find w and b that minimize the
surrogate loss

Finding the minimum



You're blindfolded, but you can see out of the bottom of the blindfold to the ground right by your feet. I drop you off somewhere and tell you that you're in a convex shaped valley and escape is at the bottom/minimum. How do you get out?

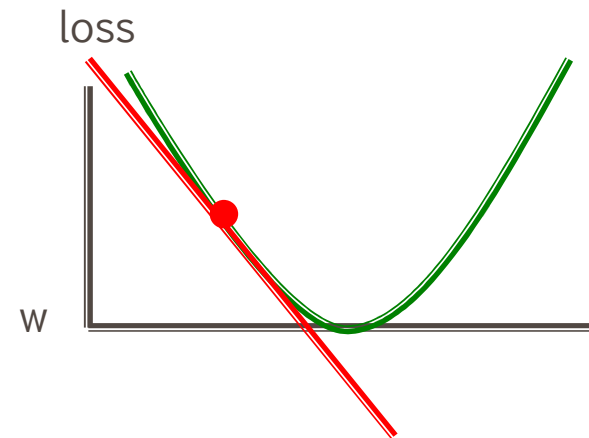
Finding the minimum



How do we do this for a function?

One approach: gradient descent

Partial derivatives give us the slope (i.e. direction to move) in that dimension

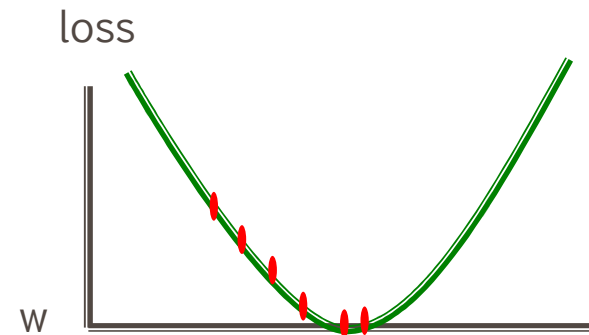


One approach: gradient descent

Partial derivatives give us the slope (i.e. direction to move) in that dimension

Approach:

- pick a starting point (w)
- repeat:
 - pick a dimension
 - move a small amount in that dimension towards decreasing loss (using the derivative)

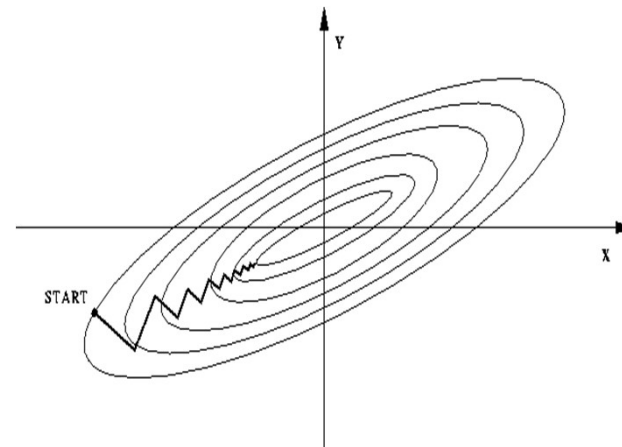


One approach: gradient descent

Partial derivatives give us the slope (i.e. direction to move) in that dimension


Approach:

- pick a starting point (w)
- repeat:
 - pick a dimension
 - move a small amount in that dimension towards decreasing loss (using the derivative)



Gradient descent


- pick a starting point (w)
- repeat until loss doesn't decrease in all dimensions:
 - pick a dimension
 - move a small amount in that dimension towards decreasing loss (using the derivative)

$$w_j = w_j - \eta \frac{d}{dw_j} \text{loss}(w)$$


What does this do?

Gradient descent

- pick a starting point (w)
- repeat until loss doesn't decrease in all dimensions:
 - pick a dimension
 - move a small amount in that dimension towards decreasing loss (using the derivative)

$$w_j = w_j - \eta \frac{d}{dw_j} \text{loss}(w)$$


learning rate (how much we want to move in the error direction, often this will change over time)

Some maths

$$\begin{aligned}\frac{d}{dw_j} loss &= \frac{d}{dw_j} \sum_{i=1}^n \exp(-y_i(w \cdot x_i + b)) \\ &= \sum_{i=1}^n \exp(-y_i(w \cdot x_i + b)) \frac{d}{dw_j} (-y_i(w \cdot x_i + b)) \\ &= \sum_{i=1}^n -y_i x_{ij} \exp(-y_i(w \cdot x_i + b))\end{aligned}$$

Gradient descent

- pick a starting point (w)
- repeat until loss doesn't decrease in all dimensions:
 - pick a dimension
 - move a small amount in that dimension towards decreasing loss (using the derivative)

$$w_j = w_j + \eta \sum_{i=1}^n y_i x_{ij} \exp(-y_i (w \cdot x_i + b))$$

What is this doing?

Exponential update rule

$$w_j = w_j + \eta \sum_{i=1}^n y_i x_{ij} \exp(-y_i (w \cdot x_i + b))$$

for each example x_i :

$$w_j = w_j + \eta y_i x_{ij} \exp(-y_i (w \cdot x_i + b))$$

Does this look familiar?

Perceptron learning algorithm!

repeat until convergence (or for some # of iterations):

for each training example (f_1, f_2, \dots, f_m , label):

$$\text{prediction} = b + \sum_{j=1}^m w_j f_j$$

if $\text{prediction} * \text{label} \leq 0$: // they don't agree

for each w_j :

$$w_j = w_j + f_j * \text{label}$$

$$b = b + \text{label}$$

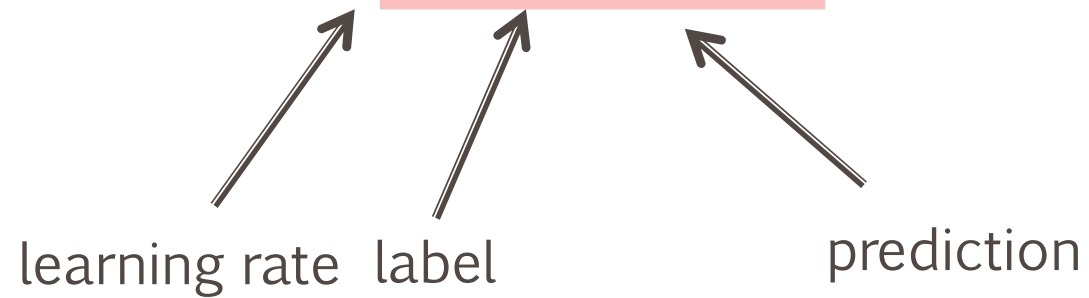
$$w_j = w_j + \eta y_i x_{ij} \exp(-y_i (w \cdot x_i + b))$$

or

$$w_j = w_j + x_{ij} y_i c \quad \text{where } c = \eta \exp(-y_i (w \cdot x_i + b))$$

The constant

$$c = \eta \exp(-y_i(w \cdot x_i + b))$$



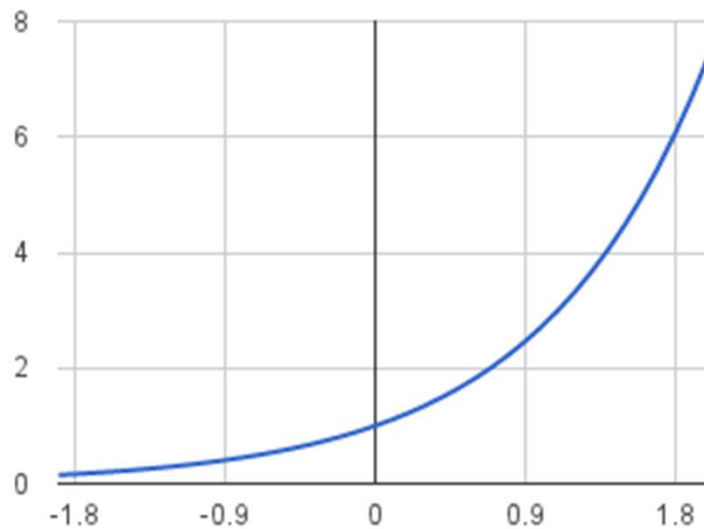
When is this large/small?

The constant

$$c = \eta \exp(-y_i(w \cdot x_i + b))$$

↑
label

↑
prediction



If they're the same sign, as the predicted gets larger there update gets smaller

If they're different, the more different they are, the bigger the update

Perceptron learning algorithm!

repeat until convergence (or for some # of iterations):

for each training example (f_1, f_2, \dots, f_m , label):

$$prediction = b + \sum_{j=1}^m w_j f_j$$

~~if $prediction * label \leq 0$: // they don't agree~~

for each w_j :

$$w_j = w_j + f_j * label$$

$$b = b + label$$

Note: for gradient descent, we always update

$$w_j = w_j + \eta y_i x_{ij} \exp(-y_i (w \cdot x_i + b))$$

or

$$w_j = w_j + x_{ij} y_i c \quad \text{where } c = \eta \exp(-y_i (w \cdot x_i + b))$$

Summary

- Model-based machine learning:
 - define a model, objective function (i.e. loss function), minimization algorithm

- Gradient descent minimization algorithm
 - require that our loss function is convex
 - make small updates towards lower losses

- Perceptron learning algorithm:
 - gradient descent
 - exponential loss function (modulo a learning rate)