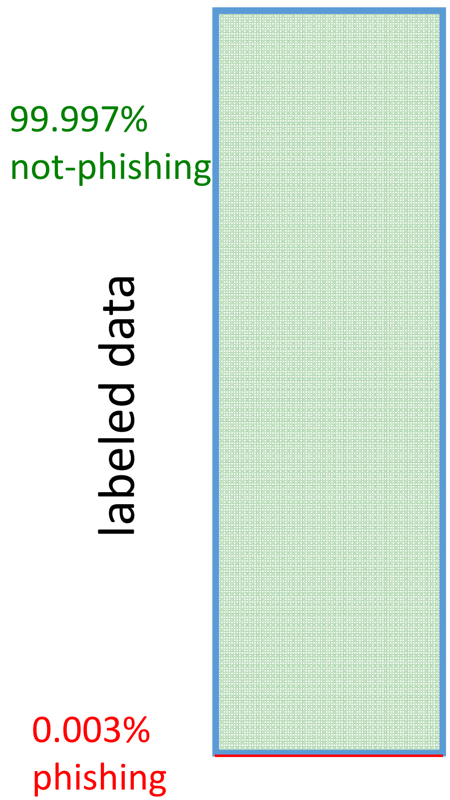
Example: Phishing

Example: Phishing

- You get 1M e-mails (randomly provided by Gmail)
  - they are labeled as “phishing” or “not-phishing”
- You have a fantastic feature representation for text / email
- You try out a few of your favorite classifiers
- You achieve an accuracy of 99.997%

- Should you be happy?

Imbalanced data



The phishing problem is an **imbalanced data** problem

Large discrepancy between the number of examples with each class label

e.g. for our 1M example dataset, maybe only about 30 are phishing e-mails

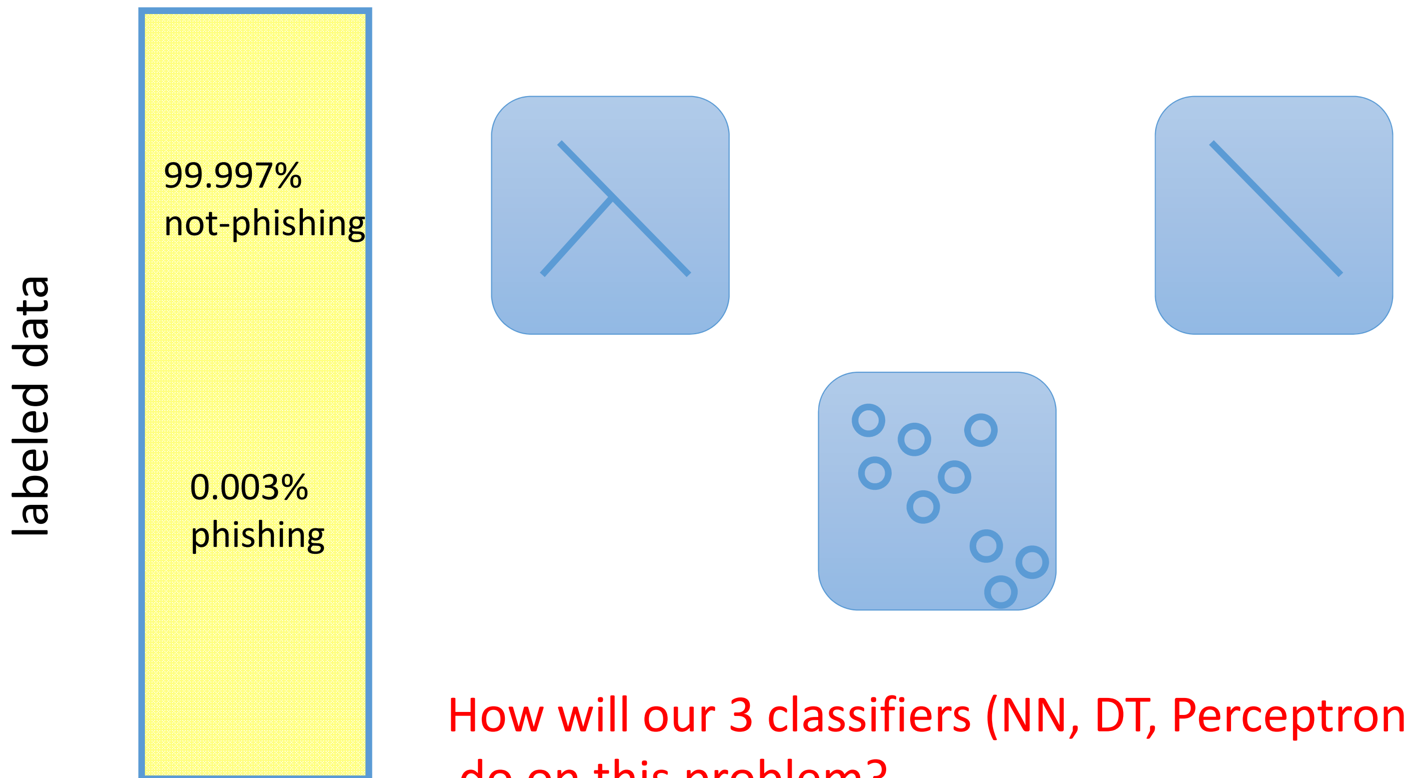
What is probably going on with our classifier?

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
  - Problem is worse when the imbalance is more pronounced

- Common in certain domains:
  - Medical diagnosis
  - Predicting faults/failures (e.g. hard-drive failures, mechanical failures, etc.)
  - Predicting rare events (e.g. earthquakes, credit card fraud)

Imbalanced data: current classifiers



How will our 3 classifiers (NN, DT, Perceptron) do on this problem?

Imbalanced data: current classifiers

- 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

“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

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

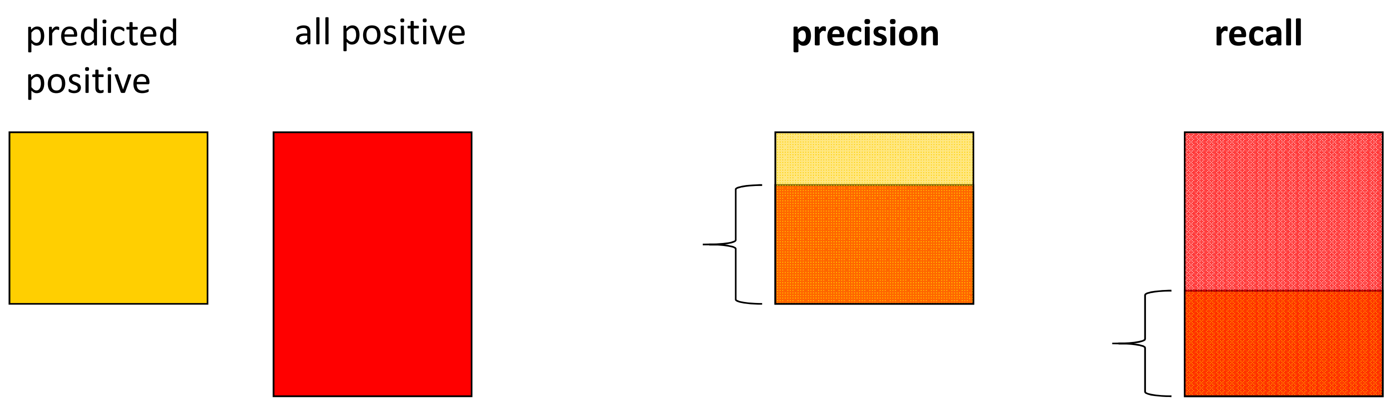
“identification” tasks

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

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

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

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



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-Recall tradeoff: Easy to maximize one and ignore the other.

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	1/2 = 0.5	1/3 = 0.33
	1	1	0.55		
	0	1	0.50		
	1	0	0.20		
	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	3/5 = 0.6	3/3 = 1.0
	0	0	0.75		
	0	0	0.90		

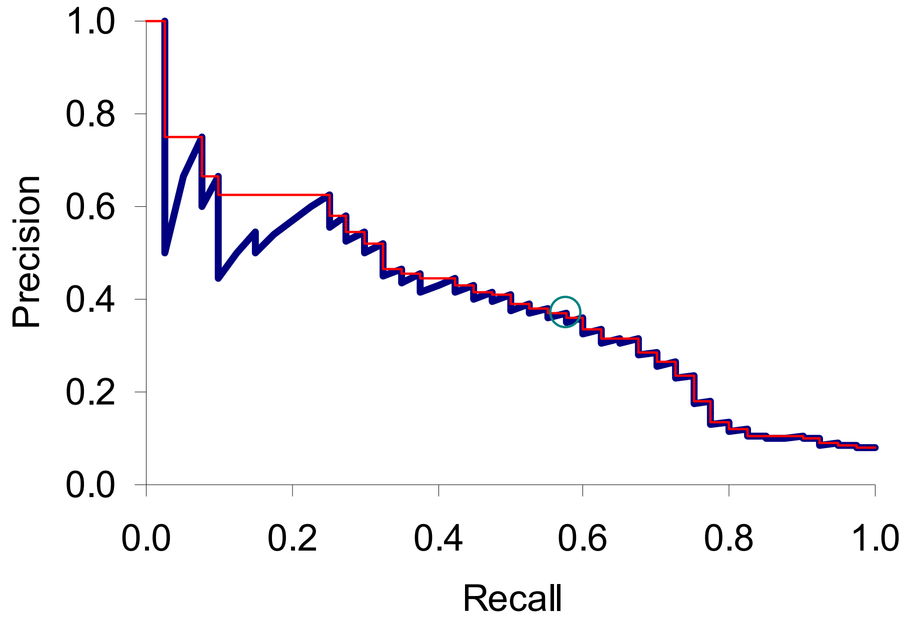
precision/recall tradeoff

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	3/7 = 0.43	3/3 = 1.0

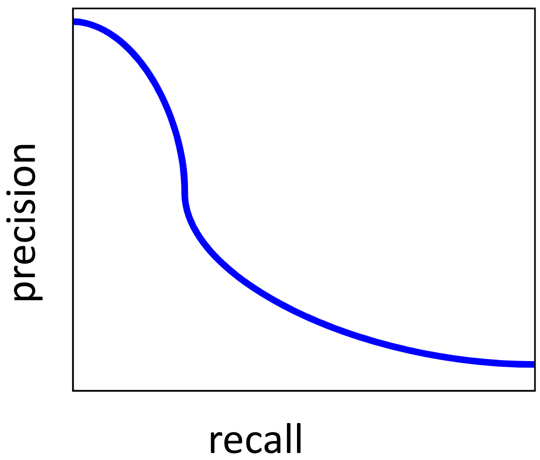
precision/recall tradeoff

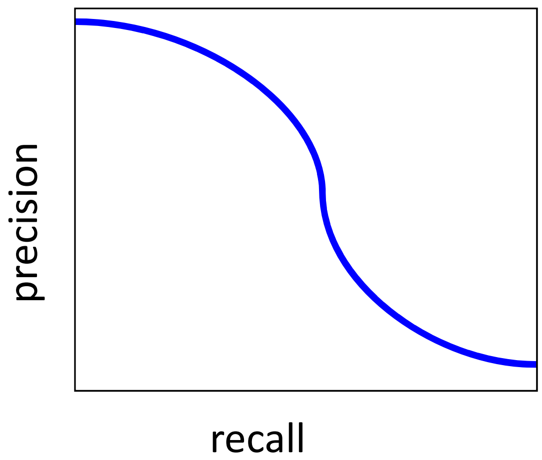
data	label	predicted	confidence	precision	recall
	1	1	0.80	1.0	0.33
	0	1	0.60	0.5	0.33
	1	1	0.55	0.66	0.66
	0	1	0.50	0.5	0.66
	1	0	0.20	0.6	1.0
	0	0	0.75	0.5	1.0
	0	0	0.90	0.43	1.0

precision-recall curve



Which is system is better?



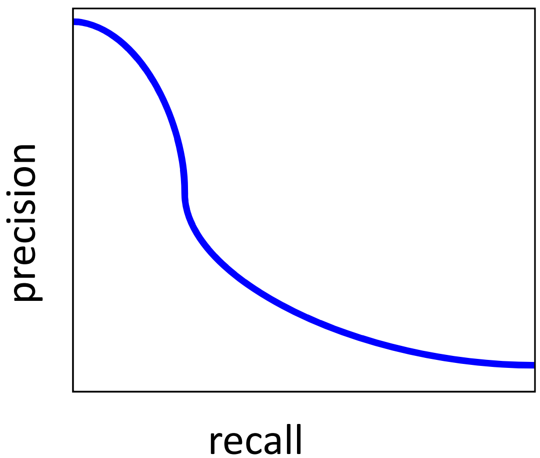


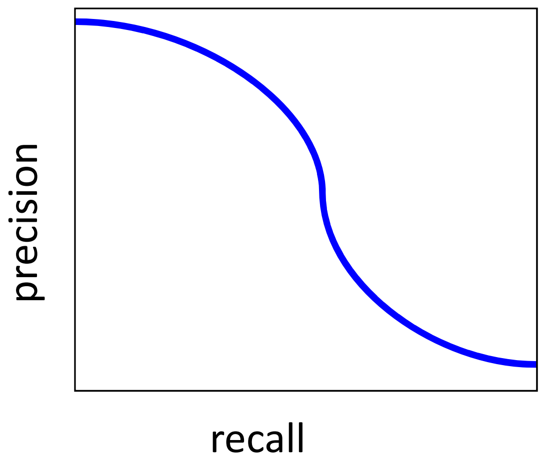
How can we quantify this?

Area under the curve (AUC)

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

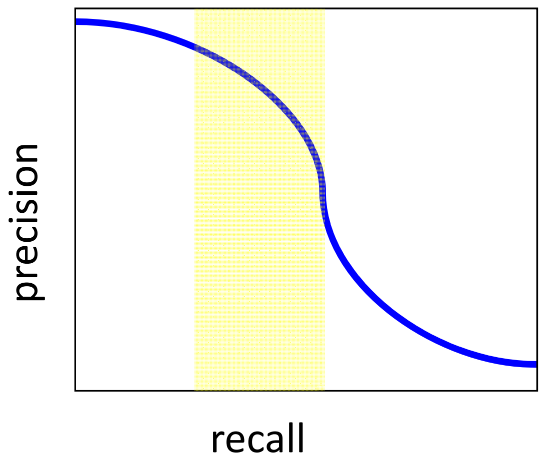
Area under the curve?



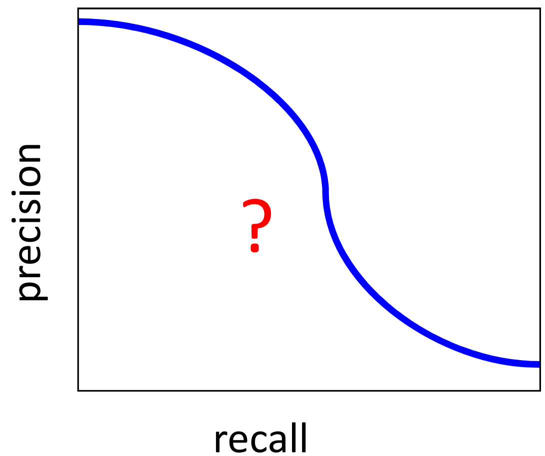


Any concerns/problems?

Area under the curve?

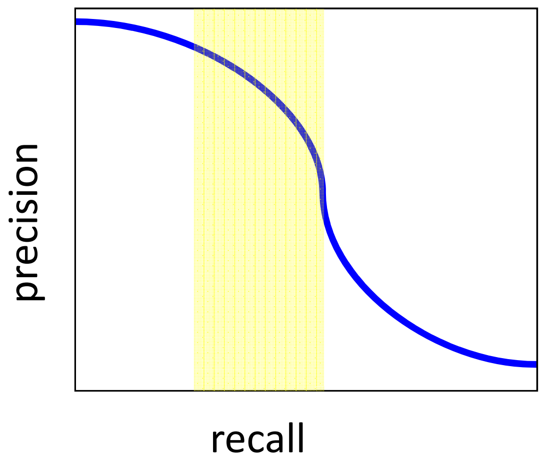


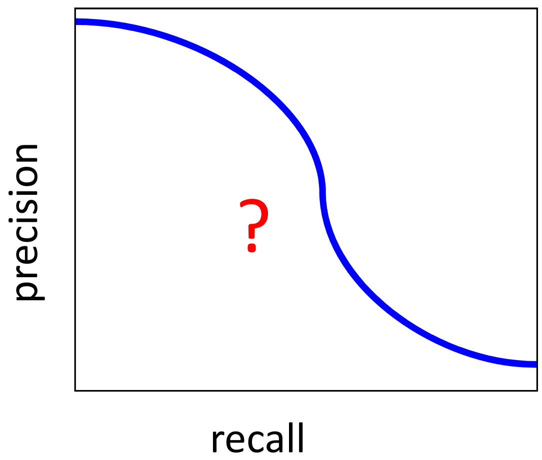
For real use, often only interested in performance in a particular range



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):

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

Harmonic mean encourages precision/recall values that are similar!

F1-measure

Most common α=0.5: equal balance/weighting between precision and recall:

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

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

Imbalanced Data

- 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)

Imbalanced Data: Another Viewpoint

We have a generic binary classifier (loss function is accuracy), can we use it for imbalanced data?

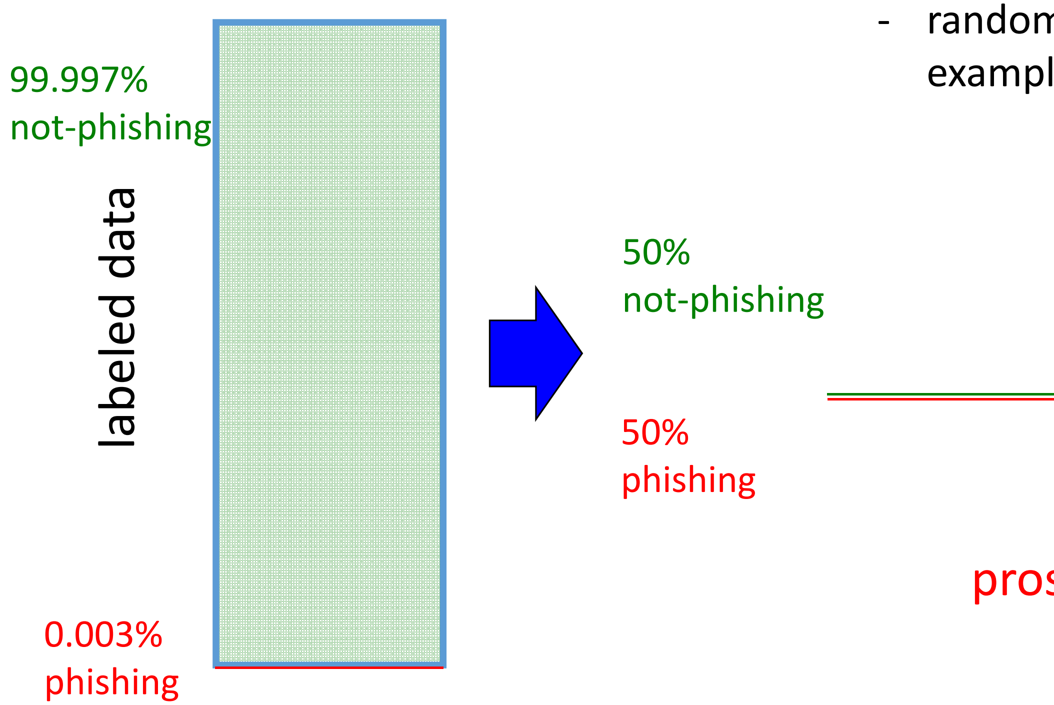
optionally: also output a confidence/score

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



pros/cons?

Subsampling

Pros:

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

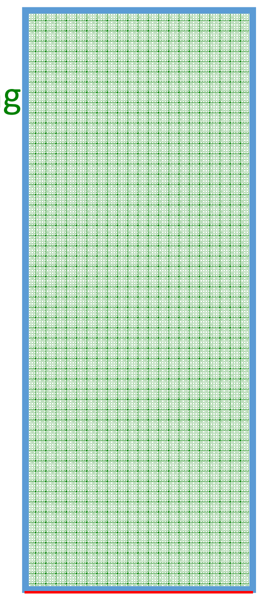
Cons:

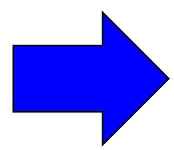
- Throwing away a lot of data/information

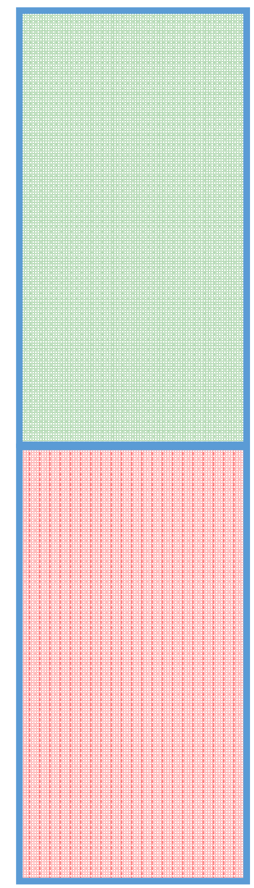
Idea 2: oversampling

Create a new training data set by:

- including all *m* “negative” examples
- include *m* “positive examples:
  - repeat each example a fixed number of times, or
  - sample with replacement







pros/cons?

oversampling

Pros:

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

Cons:

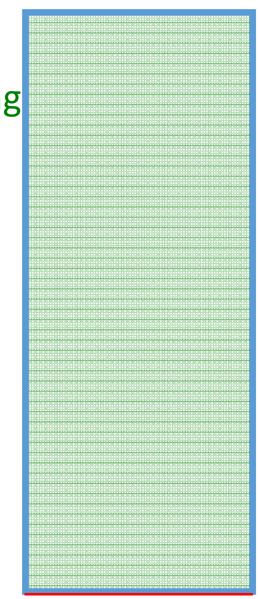
- Computationally expensive to train classifier

Idea 2b: weighted examples

cost/weights

99.997%
not-phishing

labeled data



0.003%
phishing

1

99.997/0.003 =
33332

Add costs/weights to the training set

"negative" examples get weight 1

"positive" examples get a much larger weight

change learning algorithm to optimize weighted training error

pros/cons?

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

Can our 3 classifiers be modified to handle weights?

Multiclass classification

examples

label


apple


orange


apple


banana

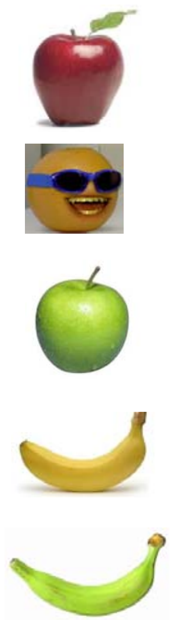

banana

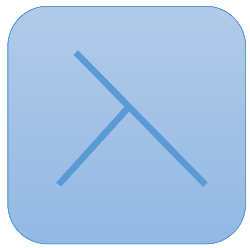

pineapple

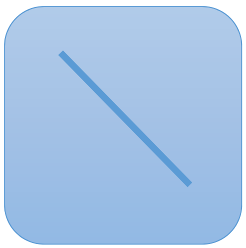
Same setup where we have a set of features for each example

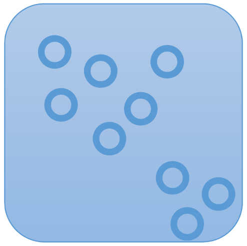
Rather than just two labels, now have 3 or more

Multiclass: current classifiers









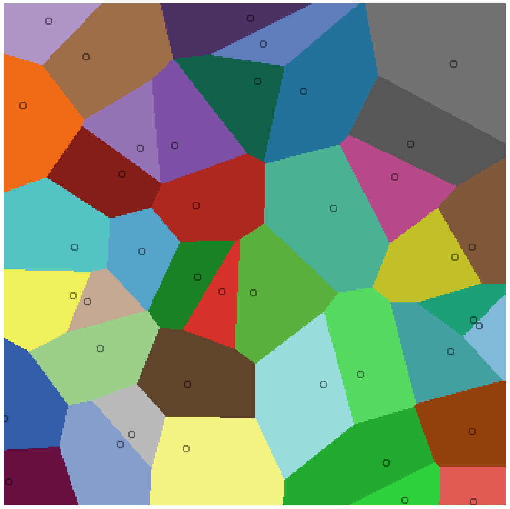
Any of these work out of the box?
With small modifications?

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:

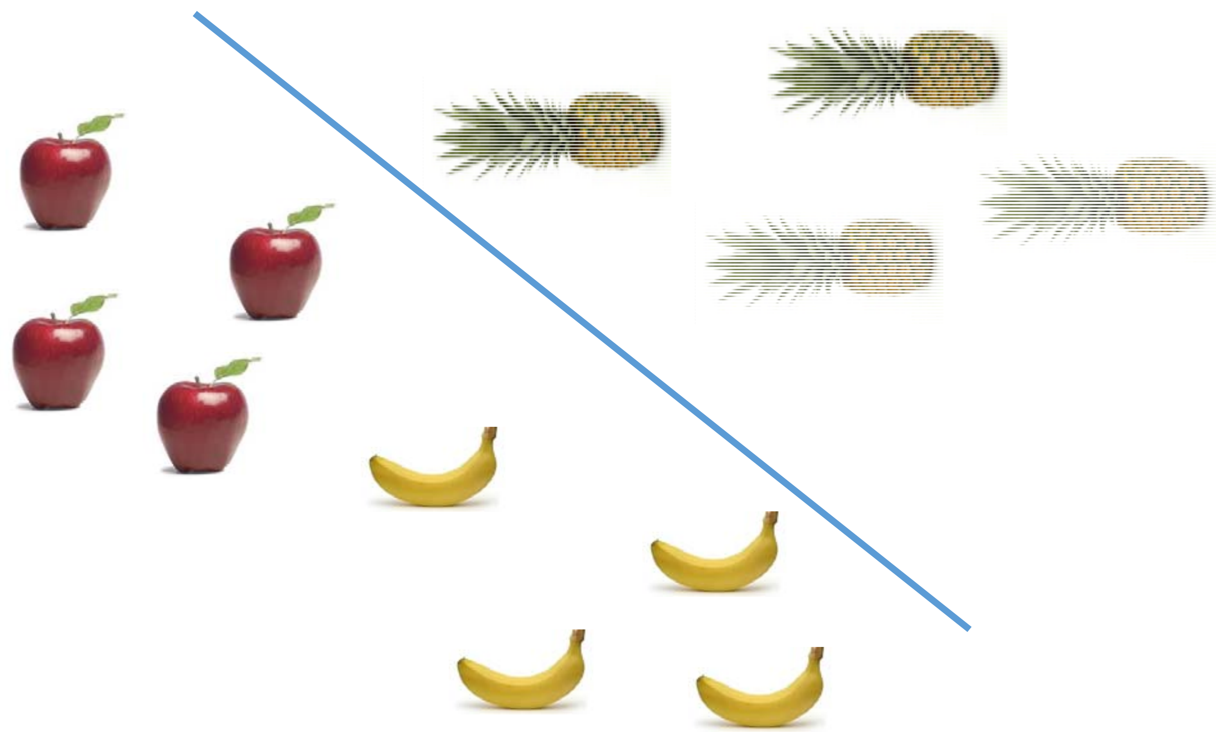
1. If all data belong to the same class, pick that label
2. If all the data have the same feature values, pick majority label
3. If we're out of features to examine, pick majority label
4. If the we don't have any data left, pick majority label of *parent*
5. *If some other stopping criteria* exists 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!

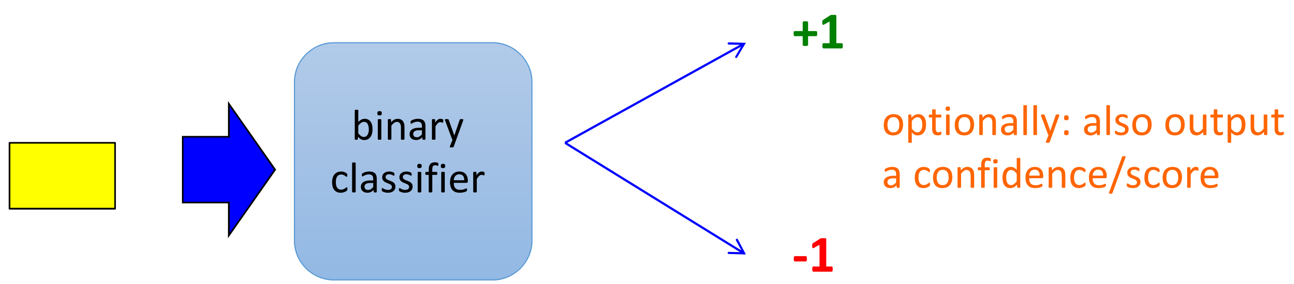
Perceptron learning



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

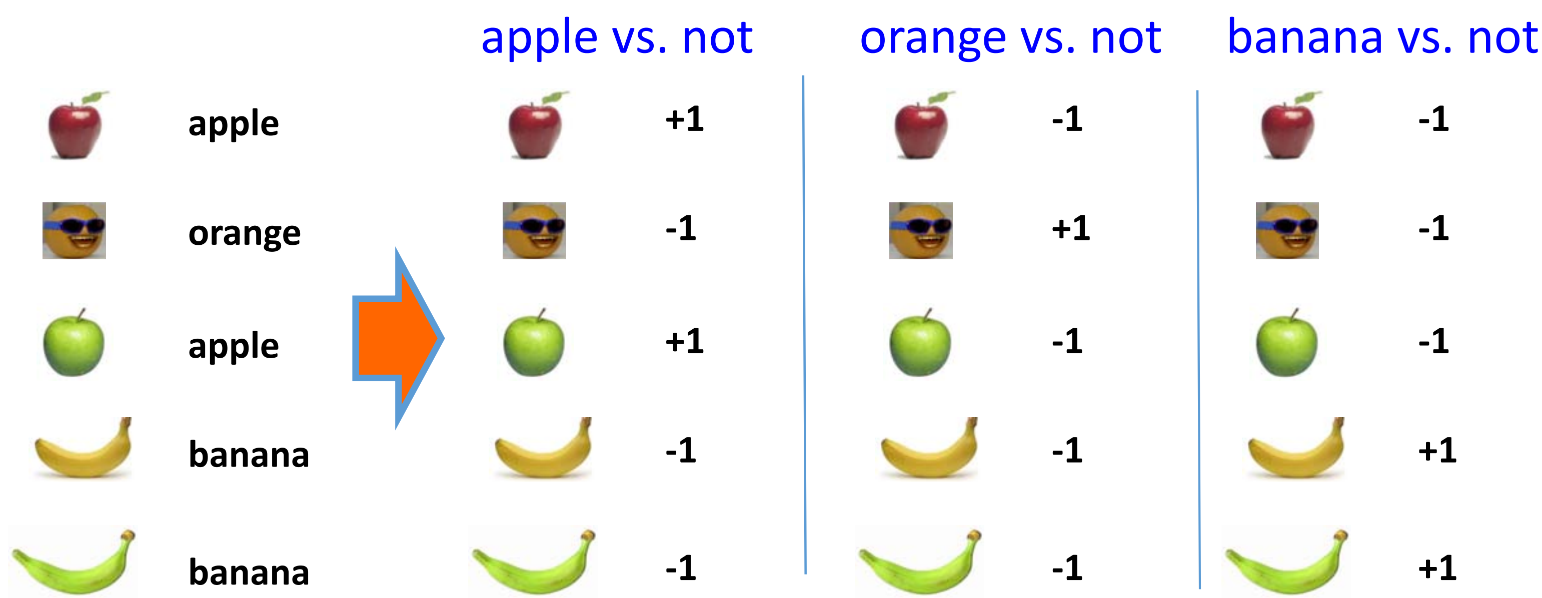


Can we solve our multiclass problem with this?

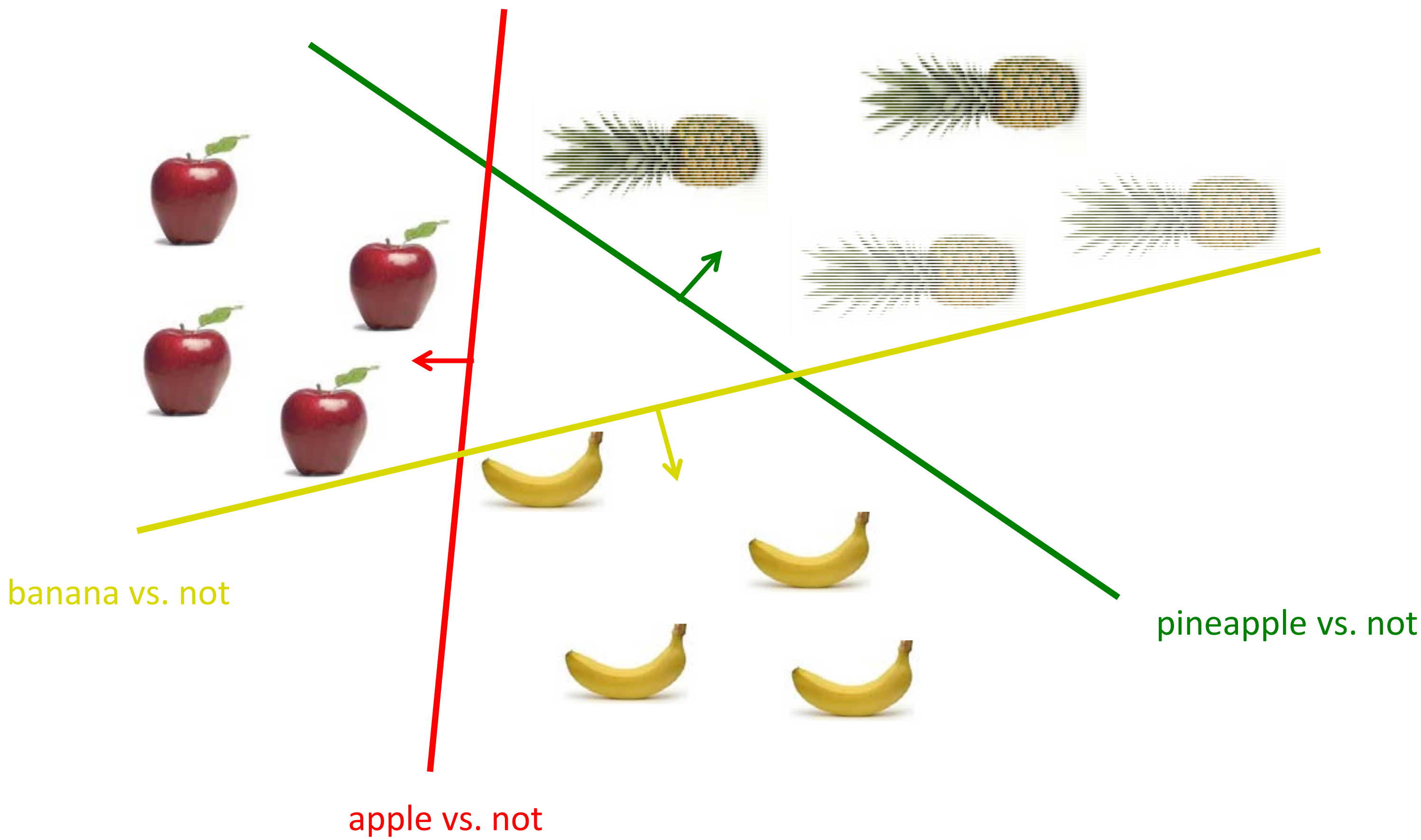
Approach 1: One vs. all (OVA)

Training: for each label *L*, pose as a binary problem

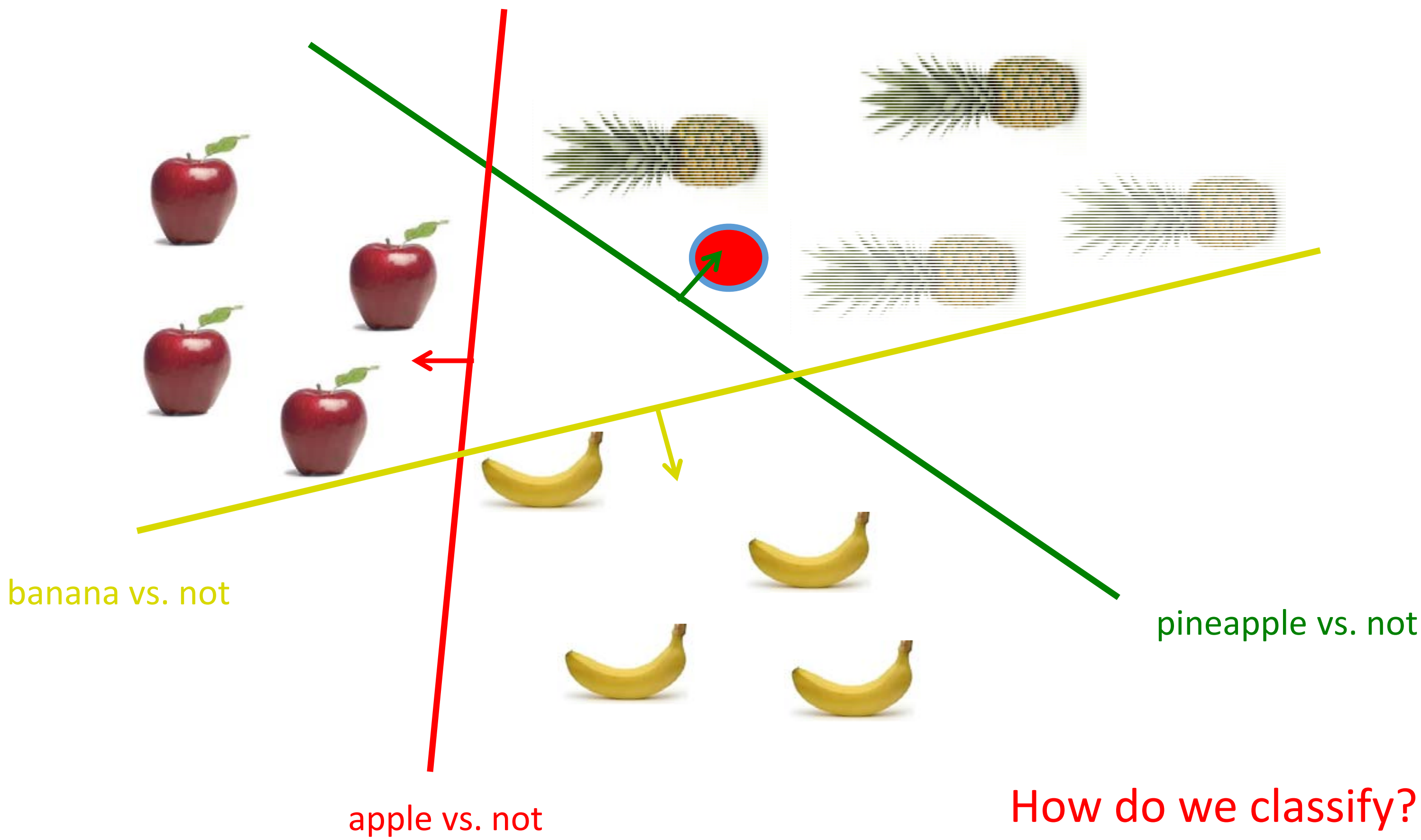
- all examples with label *L* are positive
- all other examples are negative



OVA: linear classifiers (e.g. perceptron)

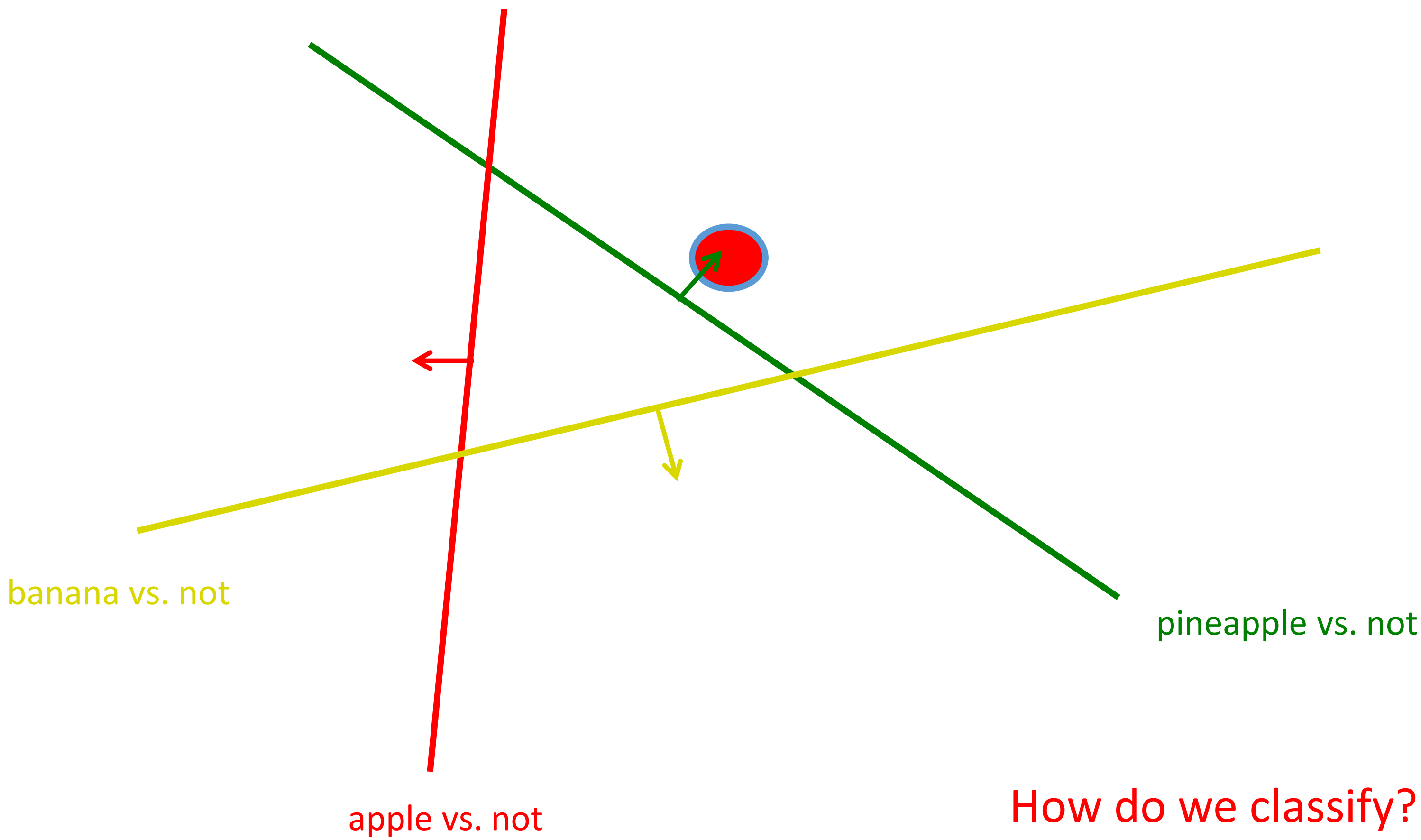


OVA: linear classifiers (e.g. perceptron)



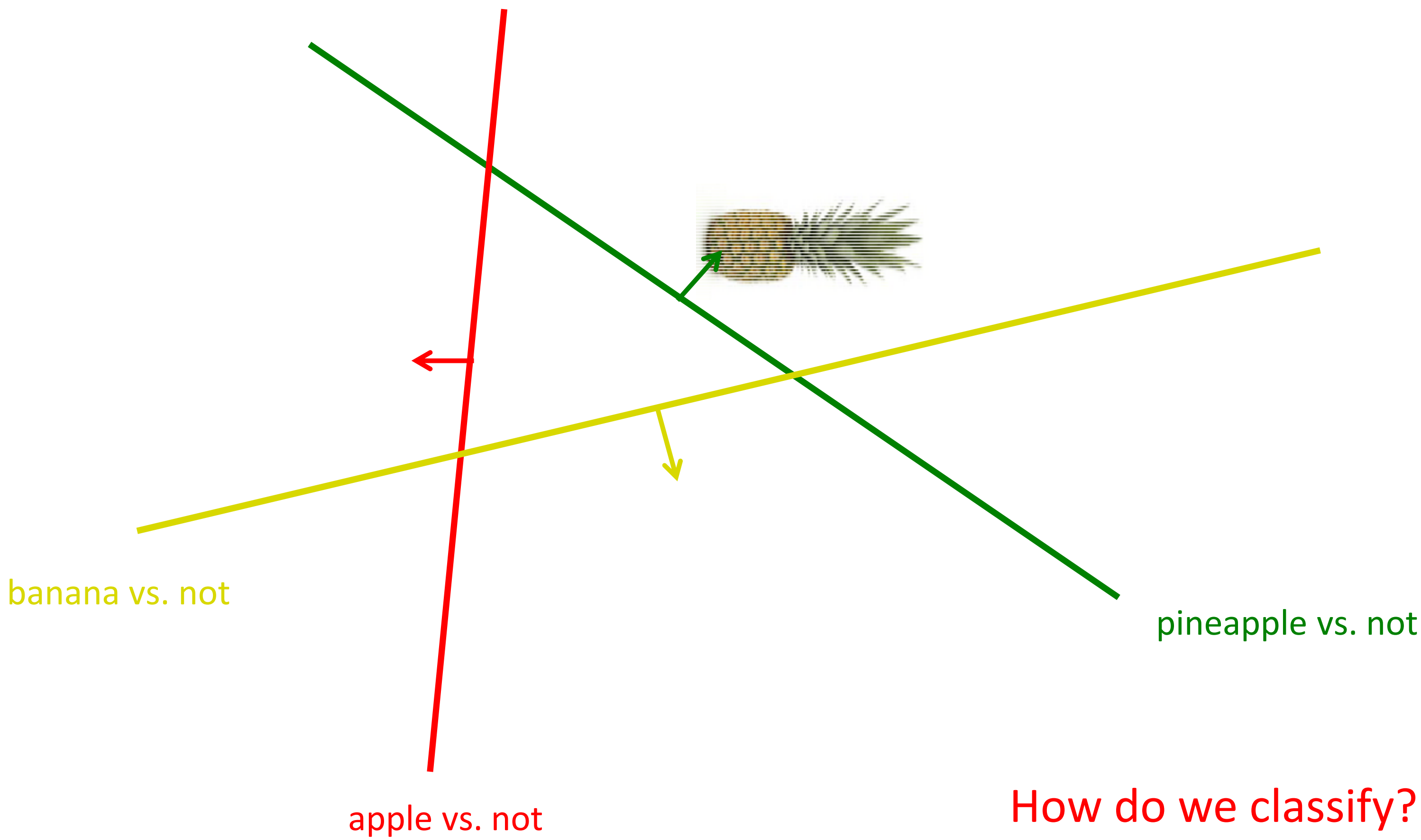
How do we classify?

OVA: linear classifiers (e.g. perceptron)



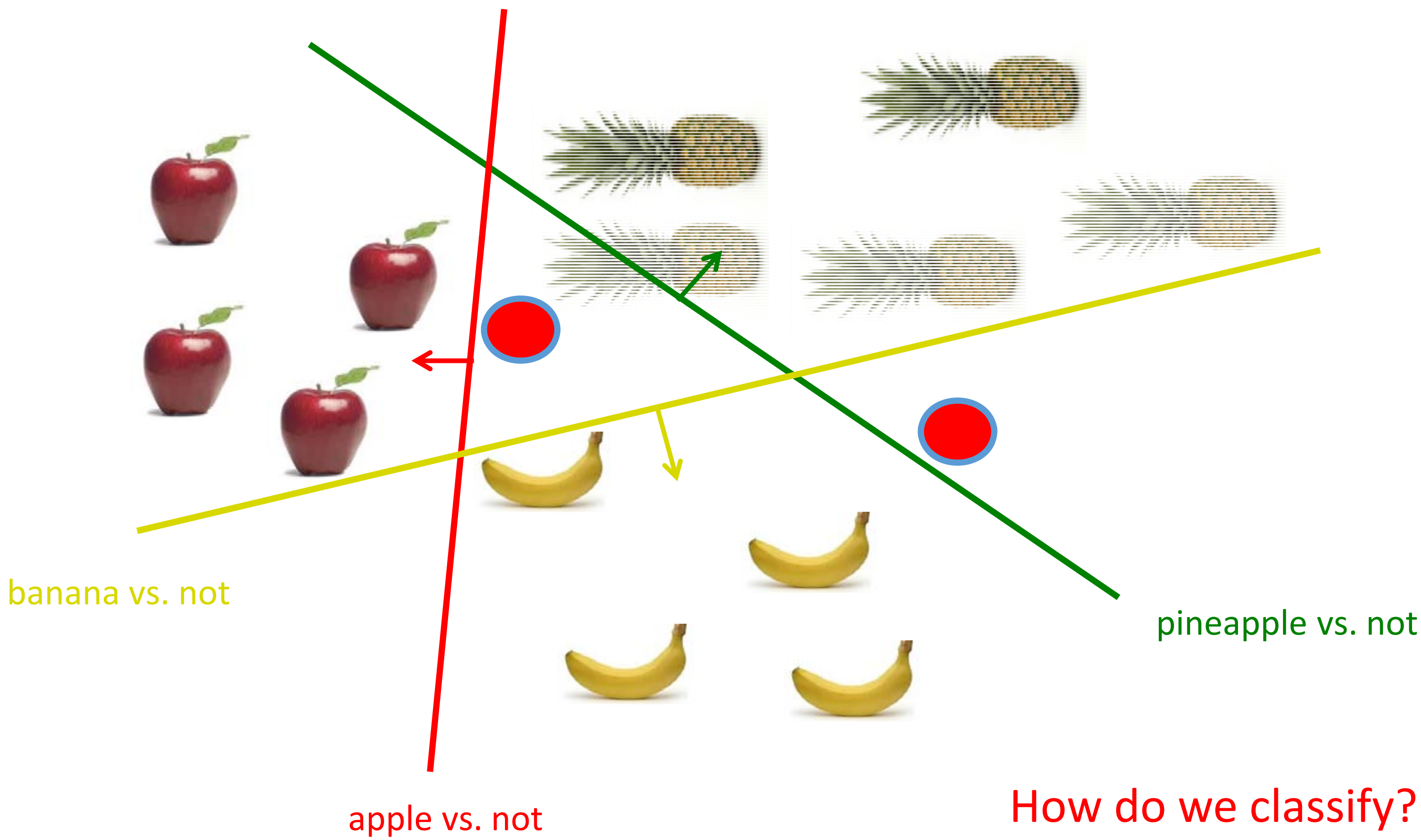
How do we classify?

OVA: linear classifiers (e.g. perceptron)



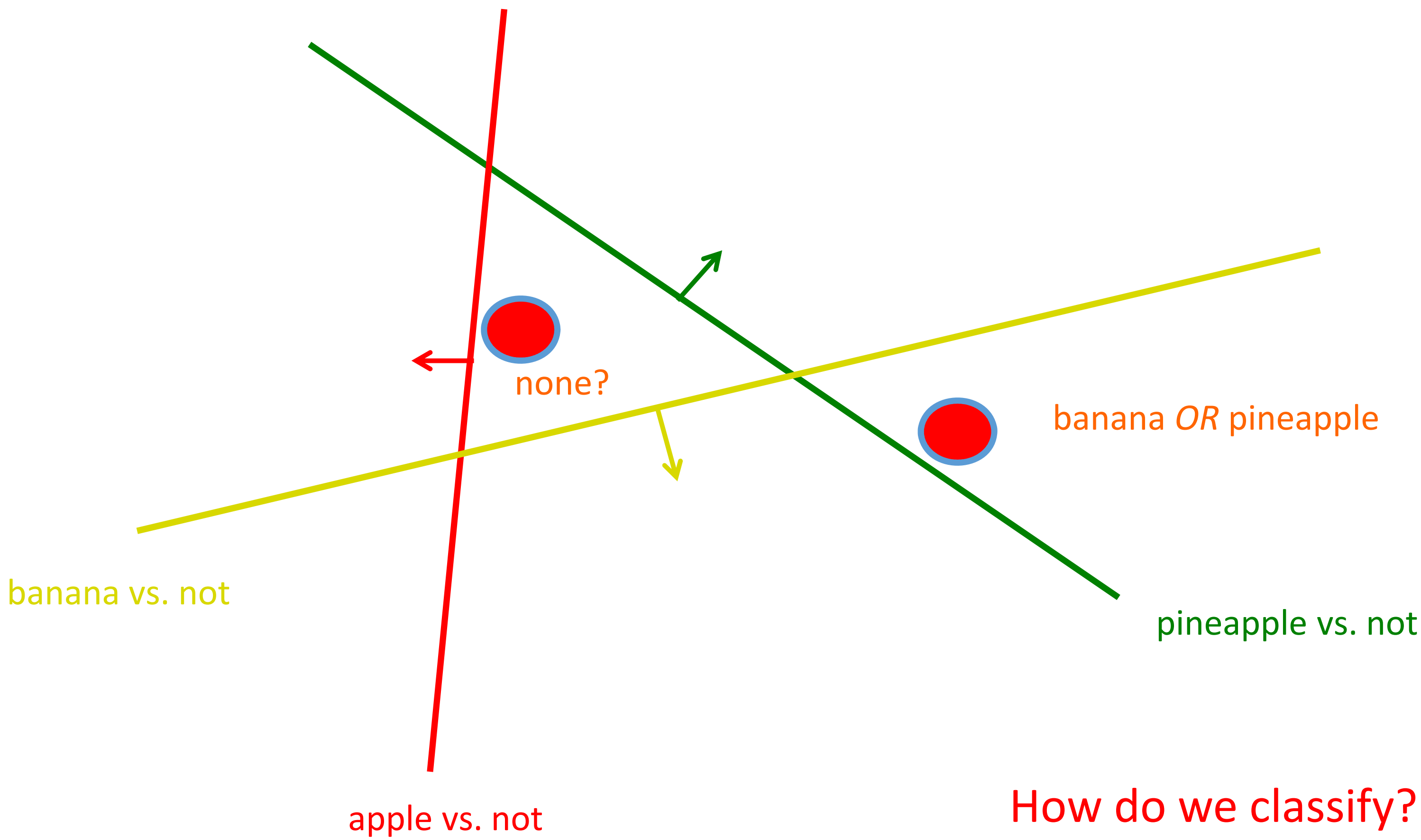
How do we classify?

OVA: linear classifiers (e.g. perceptron)



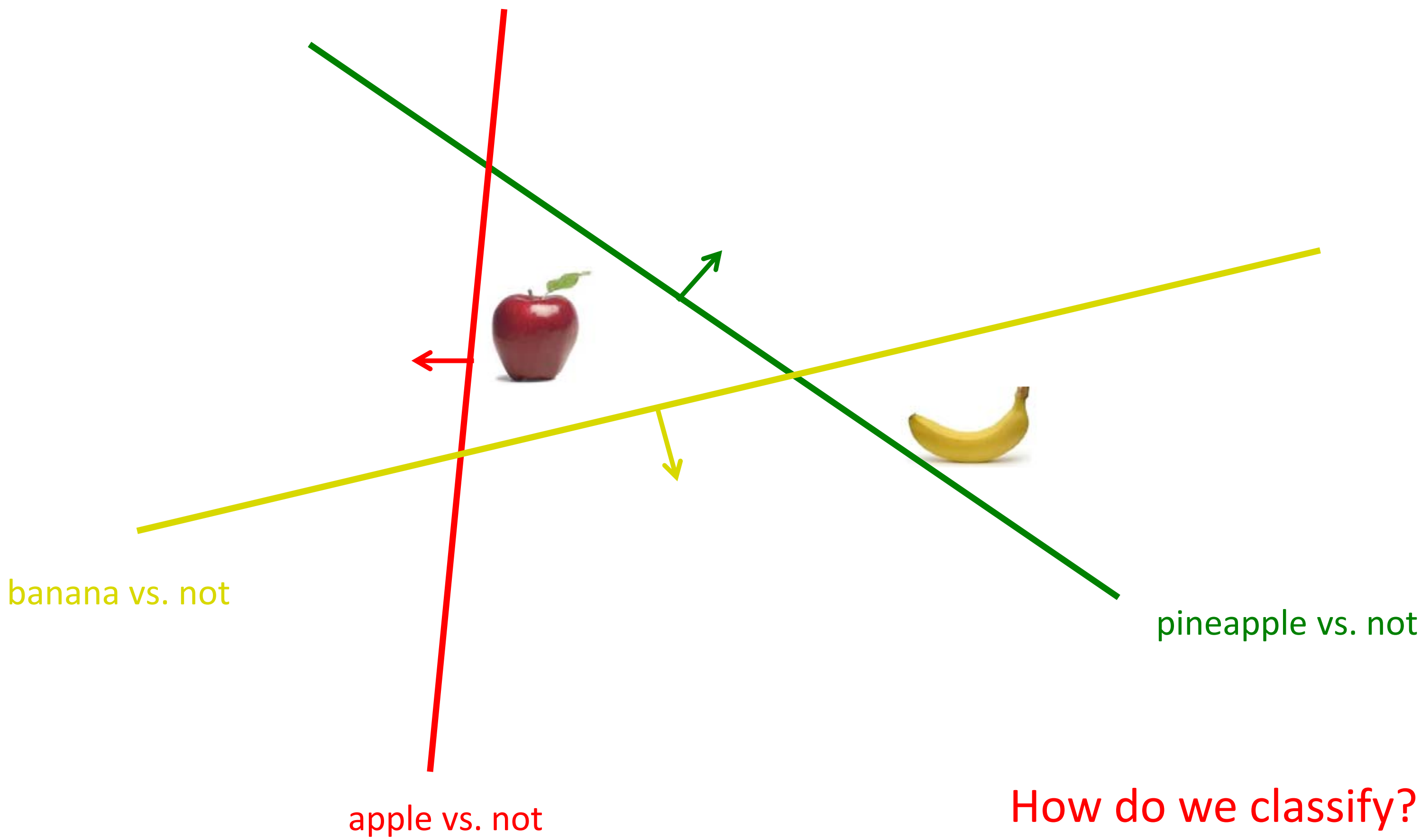
How do we classify?

OVA: linear classifiers (e.g. perceptron)



How do we classify?

OVA: linear classifiers (e.g. perceptron)



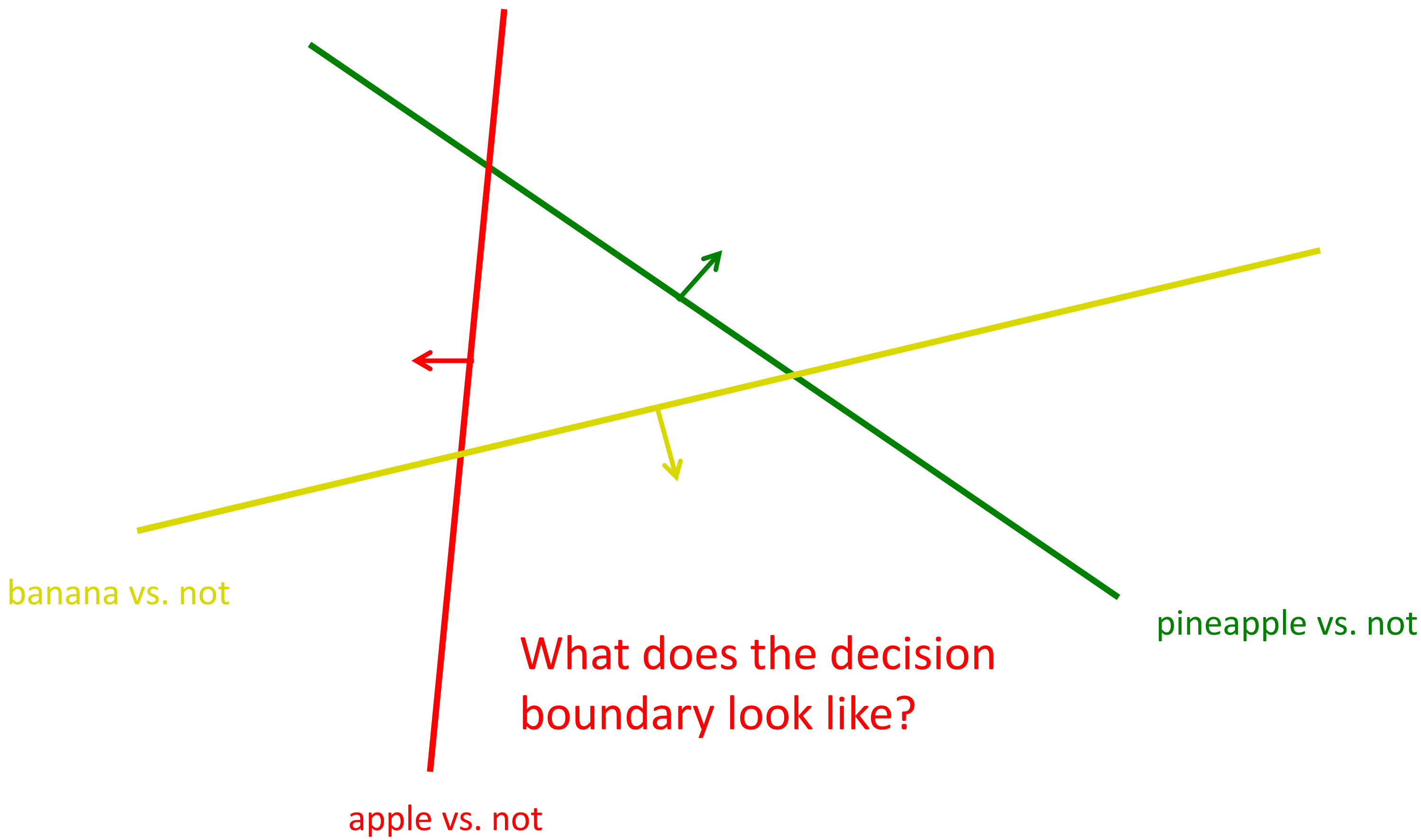
How do we classify?

OVA: classify

Classify:

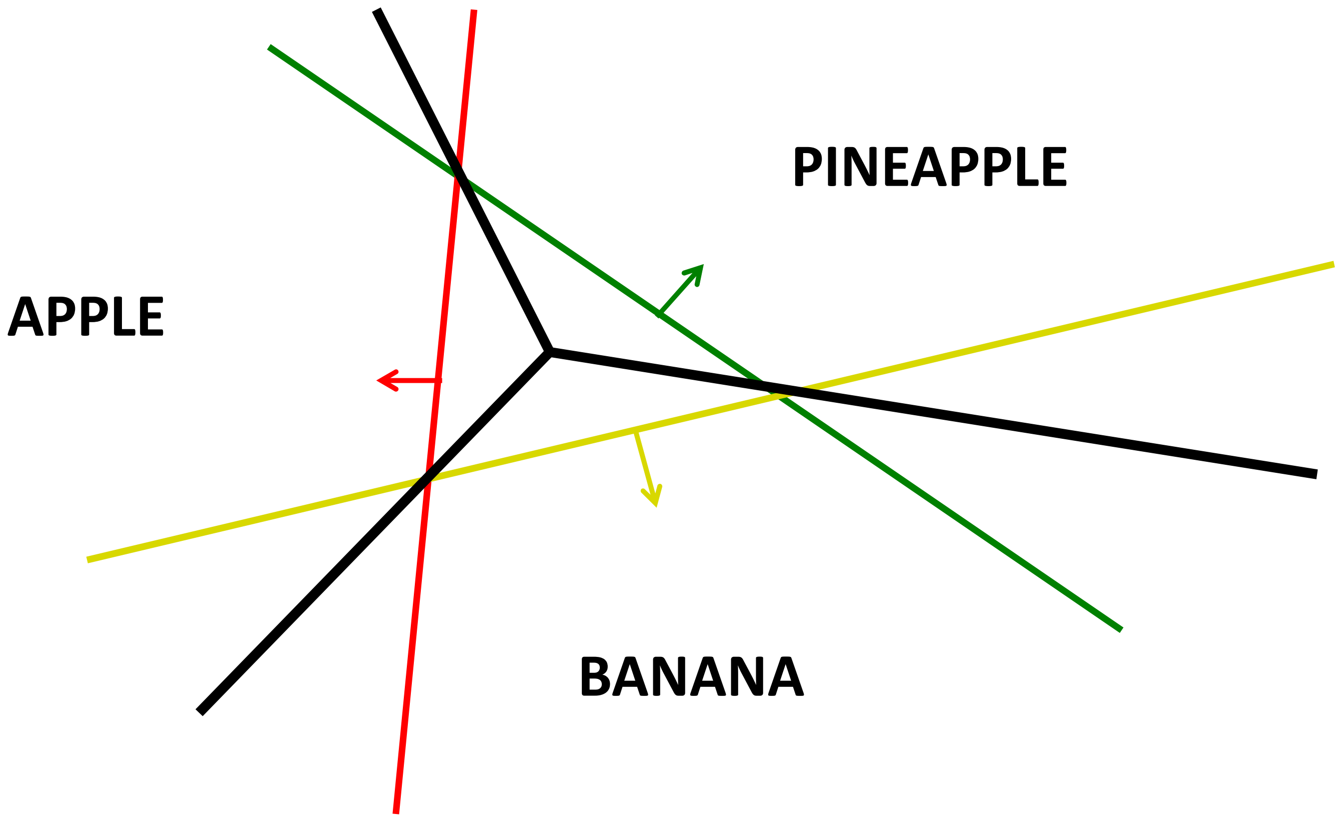
- If classifier doesn't provide confidence (this is rare) and there is ambiguity, pick one of the ones in conflict
- Otherwise:
  - pick the most confident positive
  - if none vote positive, pick *least* confident negative

OVA: linear classifiers (e.g. perceptron)



What does the decision boundary look like?

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

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

Distance from the hyperplane

Approach 2: All vs. all (AVA)

Training:

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

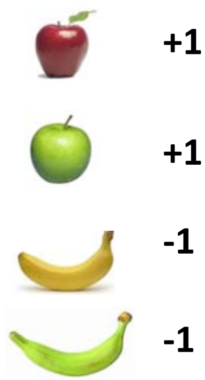
for i = 1 to number of labels:
    for k = i+1 to number of labels:
        train a classifier to distinguish between $label_j$ and $label_k$:
            - create a dataset with all examples *with* $label_j$ labeled positive
              and all examples with $label_k$ labeled negative
           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

AVA training visualized

AVA classify

apple vs orange

apple vs banana



orange vs banana

What class?

AVA classify

apple vs orange

 +1

 +1

 -1

orange

apple vs banana

 +1

 +1

 -1

 -1

apple

orange vs banana

 +1

 -1

 -1

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
  - $y = f_{jk}(e)$
  - $score_j$ += y
  - $score_k$ -= y

How does this work?

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

AVA classify

Take a weighted vote based on confidence

- $y = f_{jk}(e)$
- $score_j$ += y
- $score_k$ -= 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

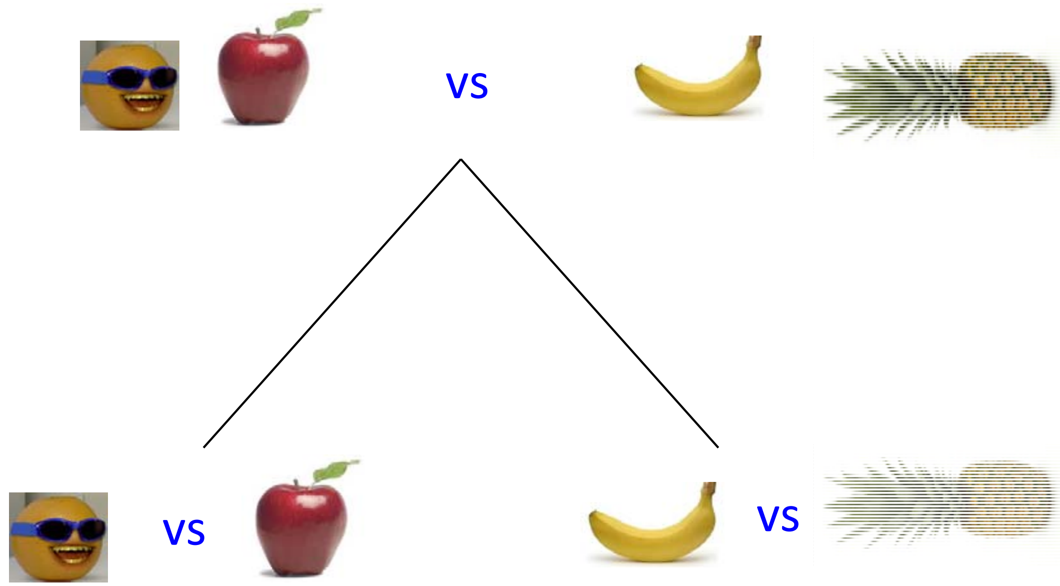
OVA vs. AVA

- Train time:
  - AVA learns more classifiers
  - However, trained on much smaller data sets
- Test time:
  - AVA has more classifiers

- Error (see CIML for additional details):
  - AVA trains on more balanced data sets
  - AVA tests with more classifiers and therefore has more chances for errors

Current research suggests neither approach is clearly better option

Approach 3: Divide and conquer



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