# IMBALANCED DATA

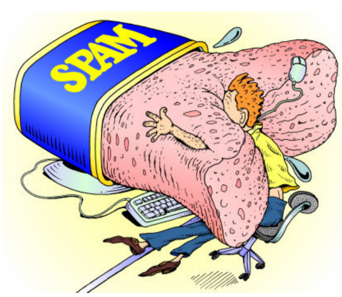
David Kauchak
CS 158 – Fall 2016

## Admin

Assignment 3:
- how did it go?
- do the experiments help?

Assignment 4

Exam schedule



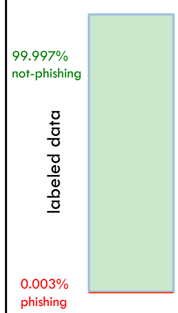

## Phishing

## Setup

1. for 1 hour, google collects 1M e-mails randomly
2. they pay people to label them as "phishing" or "not-phishing"
3. they give the data to you to learn to classify e-mails as phishing or not
4. you, having taken ML, try out a few of your favorite classifiers
5. you achieve an accuracy of 99.997%

Should you be happy?

## Imbalanced data



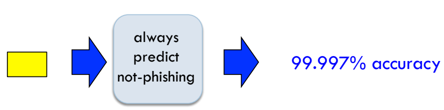

The phishing problem is what is called an **imbalanced data** problem

There is a large discrepancy between the number of examples with each class label

e.g. for 1M examples only ~30 would be phishing e-mails

What is probably going on with our classifier?

## 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

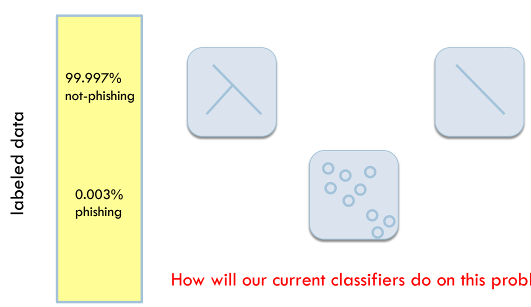
Medical diagnosis

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




How will our current classifiers do on this problem?

## Imbalanced data: current classifiers

All will do fine if the data can be easily separated/distinguished

Decision trees:

- explicitly minimizes training error
- when pruning/stopping early: 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

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

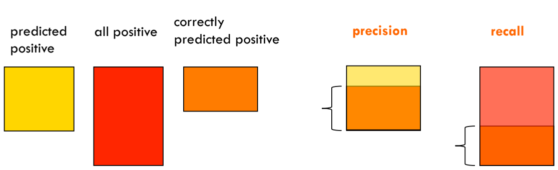
## "identification" tasks

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

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

**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 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 | 1/1 = 1.0 | 1/3 = 0.33 |
| | 0 | 1 | 0.60 | | |
| | 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 | 1/2 = 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 | | |

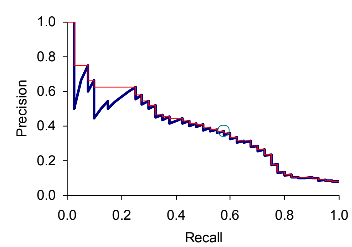
## precision/recall tradeoff

| data | label | predicted | confidence | precision | recall |
|---|---|---|---|---|---|
| | 1 | 1 | 0.80 | | |
| | 0 | 1 | 0.60 | | |
| | 1 | 1 | 0.55 | 2/3 = 0.67 | 2/3 = 0.67 |
| | 0 | 1 | 0.50 | | |
| | 1 | 0 | 0.20 | | |
| | 0 | 0 | 0.75 | | |
| | 0 | 0 | 0.90 | | |

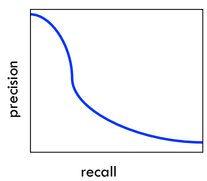
## precision/recall tradeoff

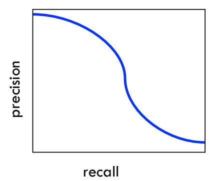
| data | label | predicted | confidence | precision | recall |
|---|---|---|---|---|---|
| | 1 | 1 | 0.80 | 1.0 | 0.33 |
| | 0 | 1 | 0.60 | 0.5 | 0.33 |
| | 1 | 1 | 0.55 | 0.66 | 0.66 |
| | 0 | 1 | 0.50 | 0.5 | 0.66 |
| | 1 | 0 | 0.20 | 0.5 | 1.0 |
| | 0 | 0 | 0.75 | 0.5 | 1.0 |
| | 0 | 0 | 0.90 | 0.5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

## precision-recall curve



## Which is system is better?




How can we quantify this?

## Area under the curve

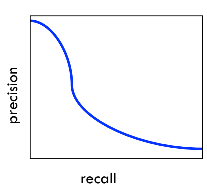
Area under the curve (PR-AUC) is one metric that encapsulates both precision and recall

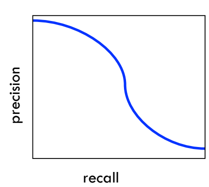
calculate the precision/recall values for all thresholding of the test set (like we did before)

then calculate the area under the curve

can also be calculated as the average precision for all the recall points (and many other similar approximations)

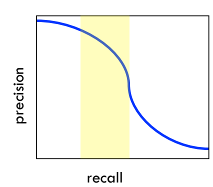
## Area under the curve?

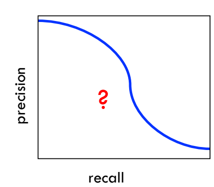



Any concerns/problems?

## Area under the curve?

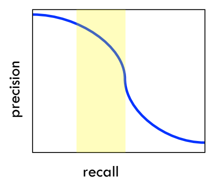


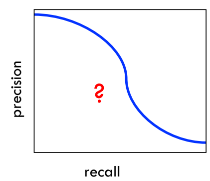




For real use, often only interested in performance in a particular range

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):

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

where $\alpha$ (or $\beta$ ) is a parameter that trades biases more towards precision or recall

$$\alpha = \frac{1}{1+\beta^2}$$

## F1-measure

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

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

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

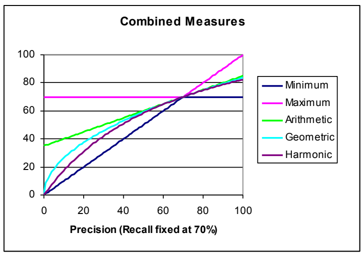
## A combined measure: *F*

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

PR-AUC and F1 can be used as a single metric to compare algorithm variations (and to tune hyperparameters)

## Phishing – imbalanced data

## Training classifiers?

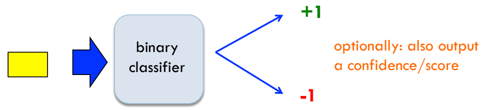
precision/recall capture different characteristics of our classifier

PR-AUC and F1 can be used as a single metric to compare algorithm variations (and to tune hyperparameters)

Can we train our classifiers to maximize this (instead of accuracy/error)?

## Black box approach

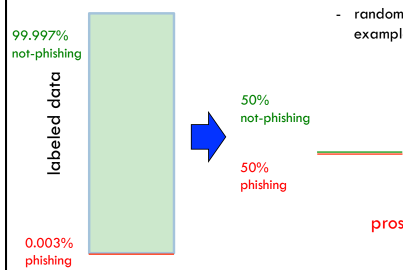
Abstraction: we have a generic binary classifier, how can we use it to solve our new problem




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 dataset by:
- including all *k* "positive" examples
- randomly picking *k* "negative" examples




pros/cons?

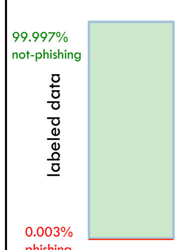
## Subsampling

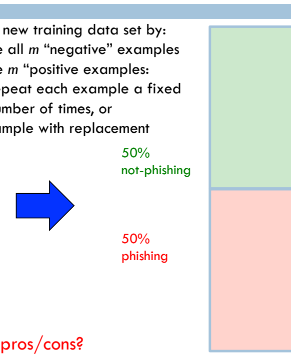
Pros:
- Easy to implement
- Training becomes much more efficient (smaller training set)
- For some domains, can work very well

Cons:
- Throwing away ***a lot*** of data/information

## Idea 2: oversampling

Create a new training data set by:
- include all *m* "negative" examples
- include *m* "positive examples:
  - repeat each example a fixed number of times, or
  - sample with replacement




pros/cons?

## oversampling

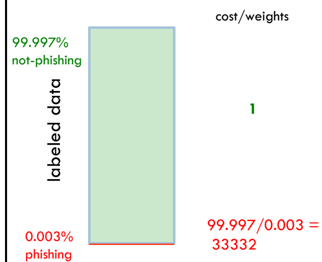
Pros:

- 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




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

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

## Building decision trees with weights

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