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

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

Why does the classifier learn this?

always predict not-phishing

99.997% accuracy
**Imbalanced problem domains**

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

<table>
<thead>
<tr>
<th>Imbalanced problem domains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Medical diagnosis</td>
</tr>
<tr>
<td>Predicting faults/failures (e.g. hard-drive failures, mechanical failures, etc.)</td>
</tr>
<tr>
<td>Predicting rare events (e.g. earthquakes)</td>
</tr>
<tr>
<td>Detecting fraud (credit card transactions, internet traffic)</td>
</tr>
</tbody>
</table>

**Imbalanced data: current classifiers**

<table>
<thead>
<tr>
<th>Labeled data</th>
<th>99.997% non-phishing</th>
<th>0.003% phishing</th>
</tr>
</thead>
</table>

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

\[
\text{Precision} = \frac{\text{# correctly predicted as positive}}{\text{# examples predicted as positive}}
\]

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

\[
\text{Recall} = \frac{\text{# correctly predicted as positive}}{\text{# positive examples in test set}}
\]

precision and recall

<table>
<thead>
<tr>
<th>data</th>
<th>label</th>
<th>predicted</th>
<th>precision</th>
<th>recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
## precision and recall

<table>
<thead>
<tr>
<th>Data</th>
<th>Label</th>
<th>Predicted</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2/4</td>
<td>2/3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

# positive examples in test set
# correctly predicted as positive
# examples predicted as positive
# positive examples in test set

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

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

## Why do we have both measures?

- Maximizing precision: Don’t predict anything as positive!
- Maximizing recall: Predict everything as positive!

## How can we maximize precision?

Don’t predict anything as positive!

## How can we maximize recall?

Predict everything as positive!
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

<table>
<thead>
<tr>
<th>Data</th>
<th>Label</th>
<th>Predicted</th>
<th>Confidence</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0.60</td>
<td>1/2 = 0.5</td>
<td>1/3 = 0.33</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.55</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0.50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0.20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.75</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.90</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Precision-Recall Curve

![Precision-Recall Curve]
Which is system is better?

Area under the curve

Which is system is better?

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).

Any concerns/problems?

Area under the curve?

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?
A combined measure: $F$

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

$$F = \frac{1}{\frac{1}{P} + \frac{1}{R}} = \frac{\beta^2 + 1}{\beta^2} PR$$

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

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

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 dataset by:
- include all m "negative" examples
- include m "positive examples:
  - repeat each example a fixed number of times, or
  - sample with replacement
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

Pros:
- Achieves the effect of oversampling without the computational cost
- Utilizes all of the training data
- Tends to perform well in a broader set of 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)
<table>
<thead>
<tr>
<th>Idea 3: optimize a different error metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train classifiers that try and optimize F1 measure or AUC or …</td>
</tr>
<tr>
<td>or, come up with another learning algorithm designed specifically for imbalanced problems</td>
</tr>
<tr>
<td>pros/cons?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Idea 3: optimize a different error metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train classifiers that try and optimize F1 measure or AUC or …</td>
</tr>
<tr>
<td>Challenge: not all classifiers are amenable to this</td>
</tr>
<tr>
<td>or, come up with another learning algorithm designed specifically for imbalanced problems</td>
</tr>
<tr>
<td>Don’t want to reinvent the wheel!</td>
</tr>
<tr>
<td>That said, there are a number of approaches that have been developed to specifically handle imbalanced problems</td>
</tr>
</tbody>
</table>