

# Profit Curves and Imbalanced Classes



# Problem Motivation

- Classification datasets can be “imbalanced”.
  - i.e. many observations of one class, few of another
- Costs of a false positive is often different from cost of a false negative.
  - e.g. missing fraud can be more costly than screening legitimate activity
- Accuracy-driven models will over-predict the majority class.

# Solutions

Cost-sensitive learning:

- thresholding (aka “profit curves”)
- modified objective functions

Sampling:

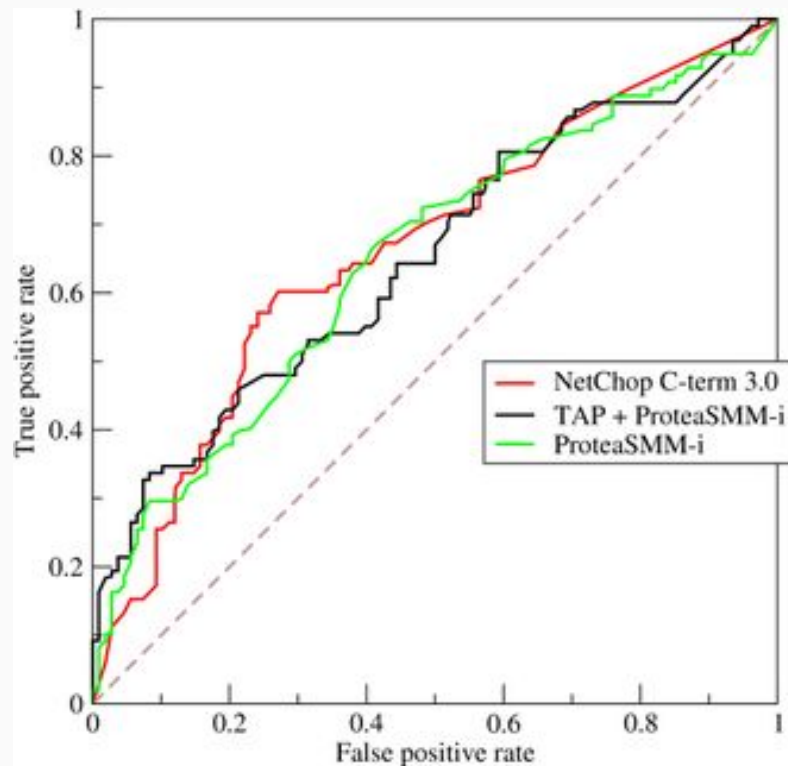
- Oversampling
- Undersampling
- SMOTE - Synthetic Minority Oversampling TEchnique

## Recall the ROC Curve:

- ROC shows  $FPR = (1 - TNR)$  vs  $TPR$  (aka Recall)
- doesn't give preference to one over the other

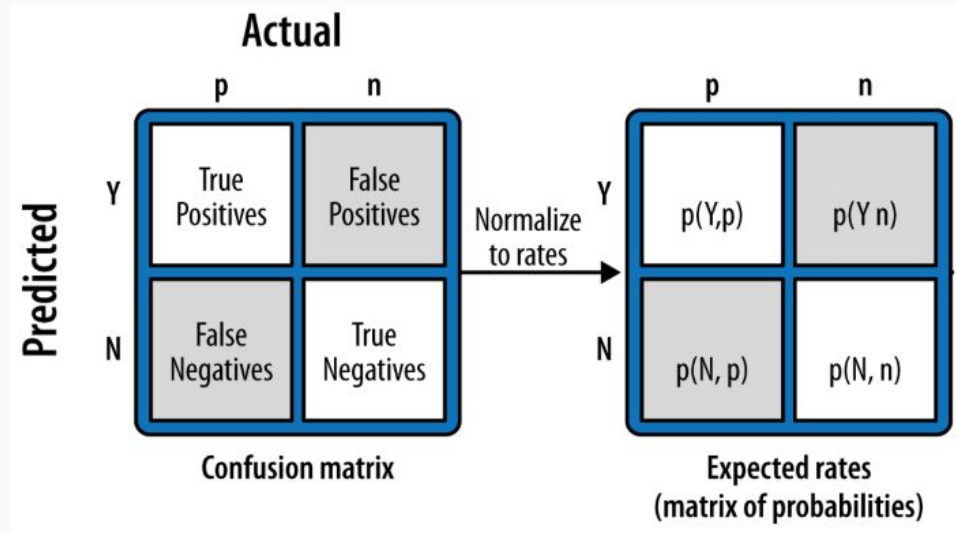
**Q:** How to handle unequal error costs?

**A:** Plot expected profit!



## Computing Expected Profit

Step 1 - Estimate error probabilities.



## Computing Expected Profit

Step 2 - Define the cost-benefit matrix.

		Actual	
		p	n
Predicted	Y	$b(Y,p)$	$c(Y,n)$
	N	$c(N,p)$	$b(N,n)$

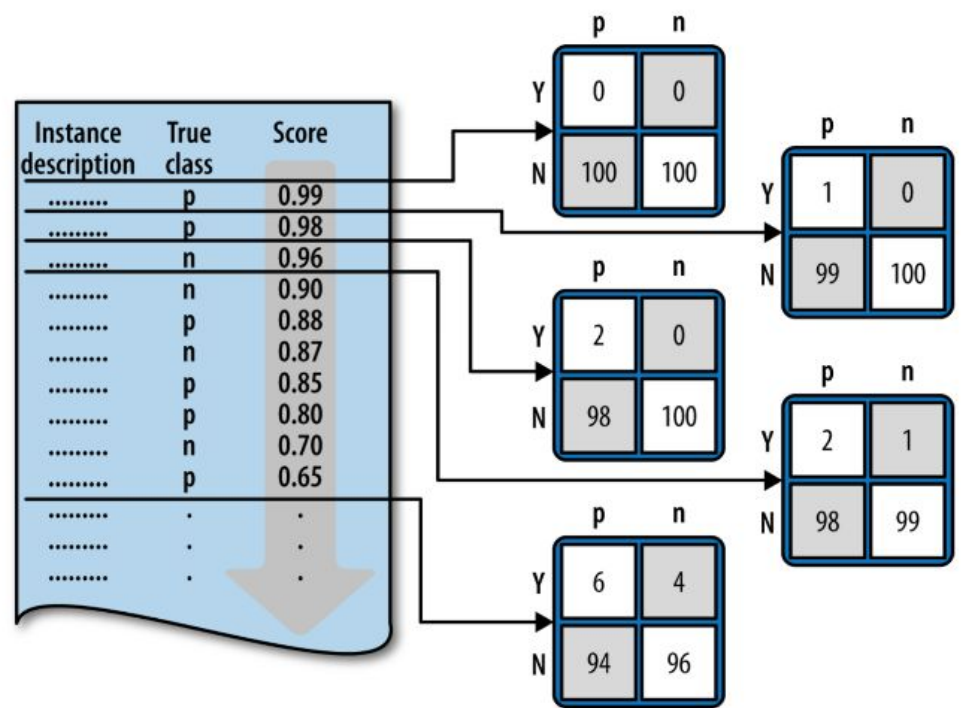
## Computing Expected Profit

Step 3 - Combine probabilities and payoffs.

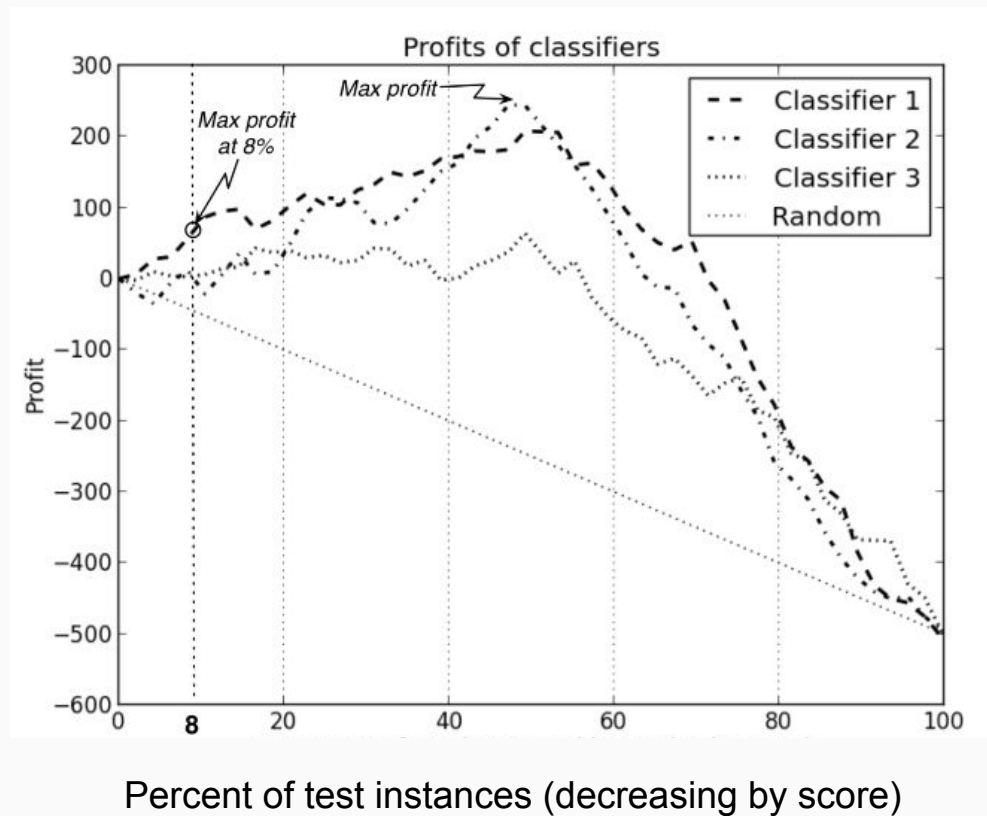
$$\begin{aligned} E[\textit{Profit}] &= P(Y, p) \cdot b(Y, p) + P(Y, n) \cdot c(Y, n) + \\ &\quad P(N, p) \cdot c(N, p) + P(N, n) \cdot b(N, n) \\ &= P(Y|p) \cdot P(p) \cdot b(Y, p) + P(Y|n) \cdot P(n) \cdot c(Y, n) + \\ &\quad P(N|p) \cdot P(p) \cdot c(N, p) + P(N|n) \cdot P(n) \cdot b(N, n) \\ &= P(p) \cdot [P(Y|p) \cdot b(Y, p) + P(N|p) \cdot c(N, p)] + \\ &\quad P(n) \cdot [P(Y|n) \cdot c(Y, n) + P(N|n) \cdot b(N, n)] \end{aligned}$$

## Find the profit-maximizing threshold

- For each possible threshold, compute expected profit.
- Then select threshold with highest expected profit.







- Models with explicit objective function can be modified to incorporate classification cost.
  - e.g. **logistic regression**
- This will affect optimization.
  - e.g. cost-sensitive logistic regression is not convex!
- Not all models have a cost-sensitive implementation.

- Logistic regression's usual objective function:

$$\ln p(\vec{y}|X; \theta) = \sum_{i=1}^n (y_i \ln h_{\theta}(x_i) + (1 - y_i) \ln(1 - h_{\theta}(x_i)))$$

- New objective function, representing expected cost:

$$J^c(\theta) = \frac{1}{N} \sum_{i=1}^N \left( y_i(h_{\theta}(X_i)C_{TP_i} + (1 - h_{\theta}(X_i))C_{FN_i}) \right. \\ \left. + (1 - y_i)(h_{\theta}(X_i)C_{FP_i} + (1 - h_{\theta}(X_i))C_{TN_i}) \right).$$

# Sampling Techniques - Undersampling

- Undersampling randomly discards majority class observations to balance training sample.
- **PRO:** Reduces runtime on very large datasets.
- **CON:** Discards potentially important observations.

# Sampling Techniques - Oversampling

- Oversampling replicates observations from minority class to balance training sample.
- **PRO:** Doesn't discard information.
- **CON:** Likely to overfit.

(Often better to use SMOTE)

# Sampling Techniques - SMOTE

- SMOTE - Synthetic Minority Oversampling TEchnique
- Generates new observations from minority class.
- For each minority class observation and for each feature, randomly generate between it and one of its k-nearest neighbors.

## SMOTE pseudocode

```
synthetic_observations = []  
while len(synthetic_observations) + len(minority_observations) < target:  
    obs = random.choice(minority_observations):  
    neighbor = random.choice(kNN(obs, k)) # randomly selected neighbor  
    new_observation = {}  
    for feature in obs:  
        weight = random() # random float between 0 and 1  
        new_feature_value = weight*obs[feature] \  
                               + (1-weight)*neighbor[feature]  
        new_observation[feature] = new_feature_value  
    synthetic_observations.append(new_observation)
```

# Sampling Techniques - Distribution

## What's the right amount of over-/under-sampling?

- If you know the cost-benefit matrix:
  - Maximize profit curve over target proportion
- If you don't know the cost-benefit matrix:
  - No clear answer...
  - ROC's AUC might be more useful...



# Cost Sensitivity vs Sampling

- Neither is strictly superior.
- Oversampling tends to work better than undersampling on small datasets.
- Some algorithms don't have an obvious cost-sensitive adaptation, requiring sampling.

See also "Cost-Sensitive Learning vs. Sampling: Which is Best for Handling Unbalanced Classes with Unequal Error Costs?" <http://storm.cis.fordham.edu/gweiss/papers/dmin07-weiss.pdf>