Adversarial Classification

Nilesh Dalvi, Pedro Domingos, Mausam, Sumit Sanghai, Deepak Verma [KDD '04, Seattle]

Presented by: Aditya Menon

UCSD

April 22, 2008

Outline

Previous supervised learning research

- Learner typically agnostic about data producer
  - Assumes that data just arises "naturally"
- But is this realistic?
  - Spam: spammers try to fool Bayesian classifiers
  - Intrusion detection: intruder tries to cover up footprint
  - ...

Previous supervised learning research

- Learner typically agnostic about data producer
  - Assumes that data just arises "naturally"
- But is this realistic?
  - Spam: spammers try to fool Bayesian classifiers
  - Intrusion detection: intruder tries to cover up footprint
  - ...
- **Reality**: Producer very much aware data is being classified
  - And can change data to fool learner

Example of clever producer

Learner recognizes:

Cheap high quality Rolex 75% off November only

Spammer switches to:

This November, purchasing low cost Rolexes is simple

Consequences of producer intereference

- What happens if we ignore this fact?
  - Producer can generate "worst case" data for classifier
  - Classifier accuracy degrades rapidly
  - Have to change our classifier to keep up
- Net result: keep reconstructing classifier
  - Figure out how we were fooled, and fix it
- But matter of time before malicious producer strikes again...

Adversarial classification

- Realistic model: treat data producer as an *adversary*
  - Producer knows the classifier being used
  - Tweaks data to maximize misclassification
- **Question**: If we know the data will be tampered, can we improve our classifier?

It's all a game

- Learner and adversary are locked in a *game*
  - Learner makes a prediction
  - Adversary deduces prediction technique, modifies data to breaks it
  - Learner knows adversary's strategy, changes classifier
  - ...
- **Question**: What is the best strategy for the Learner?

This paper

- Assumes naive Bayes classifier
  - Important assumption, hence constrains applicability
- Derives optimal adversary strategy
- Consequently, derives optimal classifier strategy
- Shows that this has a significant impact on classifier accuracy
  - Spam detection application

Notions of optimality

- How do we define the "best" strategy?
- In game theory?
  - Typically seek a *Nash equilibrium*: neither player has an incentive to change his strategy
  - Does **not** mean that either player's payoff is maximized
- In this paper?
  - Simply a locally optimal strategy
  - "Best response" to what the other player did
  - Players constantly changing strategy based on what the other does

Outline

Standard naive Bayes classifier

- Suppose an instance has n features:

$$x = (x_1, x_2, \ldots, x_n)$$

$$x_i \in \mathcal{X}_i$$

- Given an instance x, probability of it having class y is

$$P(y|x) = P(x|y)\frac{P(y)}{P(x)} = \left[\prod_{i=1}^{n} P(x_i|y)\right]\frac{P(y)}{P(x)}$$

Standard naive Bayes classifier

- Suppose an instance has n features:

$$x = (x_1, x_2, \ldots, x_n)$$

$$x_i \in \mathcal{X}_i$$

- Given an instance x, probability of it having class y is

$$P(y|x) = P(x|y)\frac{P(y)}{P(x)} = \boxed{\left[\prod_{i=1}^{n} P(x_i|y)\right]}\frac{P(y)}{P(x)}$$

  - Conditional feature-independence is the naive Bayes assumption

Cost-sensitive prediction

- Paper uses the notion of the *utility* of a classification
  - $U_C(y', y)$ is the utility or benefit of predicting that something with *true* class y has class y'
- Then, just choose the y' that maximizes

$$U(y'|x) = \sum_y P(y|x) U_C(y', y)$$

Optimal cost-sensitive prediction

- Paper considers problems with two classes
  - Malicious (+) e.g. spam
  - Harmless (-) e.g. normal email
- Optimal prediction is the y' that maximizes

$$U(y'|x) = P(+|x)U_C(y', +) + P(-|x)U_C(y', -)$$

Example of a utility matrix

- Reasonable utility choices?

		Actual	
Predicted	U_C	$+$	-
	$+$	True positive	False positive
	-	False negative	True negative

U_C	$+$	-
$+$	1	-10
-	-1	1

Outline

The nature of the adversary

- Now suppose there is an adversary that modifies the data
- Adversary's goal: For each example, modify its feature values so that the probability they are classified as "harmless" increases
- Adversary will transform an instance x using a function $\mathcal{A}(x)$:

$$\mathcal{A} : \mathcal{X}_1 \times \ldots \times \mathcal{X}_n \to \mathcal{X}_1 \times \ldots \times \mathcal{X}_n$$

- We need to know the nature of the optimal $\mathcal{A}(x)$

Adversary's limitations

- Why can't adversary just change every feature value?
  - Naturally, there is some notion of the *cost* to the adversary
- We suppose the adversary has a set of matrices W_i, where i runs over all the features
  - $W_i(x_i, x_i') =$ cost for the adversary to modify value of feature i from x_i to x_i'

Adversary's utility matrix

- How can the adversary measure success?
  - Has a matrix U_A, like the U_C matrix for the classifier
- Reasonable utility choices for U_A?

		Actual	
	U_A	$+$	-
Predicted	$+$	Malicious, prevented	Harmless, prevented
	-	Malicious, let through	Harmless, let through

U_A	$+$	-
$+$	-1	0
-	20	0

Cost-sensitive classification: adversary's perspective

- Recall that the utility for the learner to classify x as class y is

$$U(y|x) = P(+|x)U_C(y,+) + P(-|x)U_C(y,-)$$

- Learner classifies x as harmless (-) when

$$\begin{aligned} & U(+|x) \leq U(-|x) \\ \implies & \frac{P(+|x)}{P(-|x)} \leq \frac{U_C(-,-) - U_C(+,-)}{U_C(+,+) - U_C(-,+)} \end{aligned}$$

  - Call these the odds and the threshold
- So, this is what the adversary wants to ensure

Adversary's goal

- Now define logarithmic equivalents:

$$L(x) = \log \frac{P(+|x)}{P(-|x)}, \quad \text{"log odds"}$$

$$T(U_C) = \log \frac{U_C(-,-) - U_C(+,-)}{U_C(+,+) - U_C(-,+)}, \quad \text{"log threshold"}$$

- Prediction is "harmless" when the gap between the two is non-positive:

$$gap(x) := L(x) - T(U_C) \leq 0$$

Tricking the learner

- Adversary's goal: Make x classified as harmless (-), but not if it costs the adversary too much to do so
- Two questions
  - How to make classification of x harmless?
  - How much cost is too much?

How can adversary trick learner?

- First problem: how to make x be classified as harmless?
- Transform $x \mapsto x'$, so that

$$gap(x') = L(x') - T(U_C) \leq 0$$

How can adversary trick learner?

- In a naive Bayes classifier, we know

$$P(+|x) = \prod_i P(x_i|+)\frac{P(+)}{P(x)}$$

$$P(-|x) = \prod_i P(x_i|-)\frac{P(-)}{P(x)}$$

- Dividing and taking logs,

$$\log \frac{P(+|x)}{P(-|x)} = \log \frac{P(+)}{P(-)} + \sum_i \log \frac{P(x_i|+)}{P(x_i|-)}$$

$$L(x) := \log \frac{P(+)}{P(-)} + \sum_i \text{logodds}(x_i)$$

How can adversary trick learner?

- Recall that we want
$$L(x') - T(U_C) \leq 0$$
- Rewrite in terms of x's gap:
$$\begin{aligned} L(x) + L(x') - T(U_C) &\leq L(x) \\ L(x) - T(U_C) &\leq L(x) - L(x') \\ \implies gap(x) &\leq \sum_i \text{logodds}(x_i) - \text{logodds}(x'_i) \\ &:= \sum_i D_i(x_i, x'_i) \end{aligned}$$
- $D_i(x_i, x'_i)$ measures the change in log-odds if we change the ith feature from x_i to x'_i

Formulating an integer program

- We can find the optimal strategy by formulating an integer program
  - Want to change a subset features to cause a misclassification
- How to model cost of feature modification?
- Let $\delta_{i,x_i'}$ be a binary denoting if we modify feature i into x_i'
- Recall: $W_i(x_i, x_i')$ is the cost to change ith feature from x_i to x_i'
- Then

$$\text{Cost} = \sum_i \sum_{x_i'} W_i(x_i, x_i')\delta_{i,x_i'}$$

Finding optimal strategy

- Optimal strategy can be found by solving a integer program
- Natural linear constraints with integer variables:
  - Minimize cost of making changes (**Our goal**)
  - Changes cause the example to be classified as harmless
  - Feature i can only be changed once

Finding optimal strategy

- Optimal strategy can be found by solving a integer program
- Natural linear constraints with integer variables:
  - Minimize cost of making changes (**Our goal**)

$$\min \left\{ \sum_i \sum_{x'_i} W_i(x_i, x'_i) \delta_{i,x'_i} \right\}$$

  - Changes cause the example to be classified as harmless
  - Feature i can only be changed once

Finding optimal strategy

- Optimal strategy can be found by solving a integer program
- Natural linear constraints with integer variables:
  - Minimize cost of making changes (**Our goal**)

$$\min \left\{ \sum_i \sum_{x'_i} W_i(x_i, x'_i) \delta_{i,x'_i} \right\}$$

  - Changes cause the example to be classified as harmless

$$\sum_i \sum_{x'_i} D_i(x_i, x'_i) \delta_{i,x'_i} \geq gap(x)$$

  - Feature i can only be changed once

Finding optimal strategy

- Optimal strategy can be found by solving a integer program
- Natural linear constraints with integer variables:
  - Minimize cost of making changes (**Our goal**)

$$\min \left\{ \sum_i \sum_{x'_i} W_i(x_i, x'_i) \delta_{i,x'_i} \right\}$$

  - Changes cause the example to be classified as harmless

$$\sum_i \sum_{x'_i} D_i(x_i, x'_i) \delta_{i,x'_i} \geq gap(x)$$

  - Feature i can only be changed once

$$\sum_{x'_i} \delta_{i,x'_i} \leq 1$$

Optimal strategy program

- Optimal program

$$\min \left\{ \sum_i \sum_{x'_i} W_i(x_i, x'_i) \delta_{i,x'_i} \right\}$$

$$\sum_i \sum_{x'_i} D_i(x_i, x'_i) \delta_{i,x'_i} \geq gap(x)$$

$$\sum_{x'_i} \delta_{i,x'_i} \leq 1$$

$$\delta_{i,x'_i} \in \{0, 1\}$$

- The solution to the program is denoted by $MCC(x)$: the "minimum cost camouflage"

Adversary's output

- Assuming we find a solution, what do we do?
- Only use the solution if the cost is smaller than the benefit
  - The benefit is the change in utility from misclassification,

$$\Delta U_A = U_A(-,+) - U_A(+,+)$$

- Given x, we output $\mathcal{A}(x)$, where

$$\mathcal{A}(x) = \begin{cases} MCC(x) & NB(x) = +, W(x, MCC(x)) < \Delta U_A \\ x & \text{otherwise} \end{cases}$$

But can we find the MCC?

- But how easy is it to find $MCC(x)$?
- This is an integer program...
  - $\implies$ NP-hard!
- How to get around this?
  - Show that P $=$ NP...
  - ...or use an approximation!

Breaking intractability of MCC

- Solve the integer program by discretizing the problem space
  - Use dynamic programming to solve the discretized version
- Use two pruning rules to further simplify the results

▸ Skip to summary of adversarial algorithm

Breaking intractability of MCC

- Discretize our problem space, so that we can use dynamic programming
- Make logodds$(x_i) = \log \frac{P(x_i|+)}{P(x_i|-)}$ discrete
  - Minimum interval of δ, say
  - Forces D_i to be discrete too
- Focus on a new problem...

Splitting related problem into subproblems

- Using only the first i features, what is the least-cost set of changes that decreases the log-odds by w?
  - If we change the ith feature to x_i', then we can change the log-odds by $D_i(x_i, x_i')$
  - So, recursively find minimum cost needed to change the log-odds by $w - D_i(x_i, x_i')$, using the first $(i-1)$ features

Using dynamic programming for the MCC

- Suppose $FindMCC(i, w)$ finds the minimum cost needed to change the log-odds by w, using the first i features
- Consider $\widehat{gap}(x)$ to be $gap(x)$ in the discrete space
- Now run the algorithm $FindMCC(n, \widehat{gap}(x))$
  - The MCC requires us to change the log-odds by $gap(x)$, using the first n features i.e. all features

The MCC algorithm

FindMCC(i, w):

MinCost $= \infty$, MinList $= []$

for $x_i' \in \mathcal{X}_i$

if $D_i(x_i, x_i') \geq 0$

Cost, List $\leftarrow$ FindMCC$(i-1, w - D_i(x_i, x_i'))$

Cost $+= W_i(x_i, x_i')$

List $+= (i, x_i')$

if Cost $<$ MinCost

MinCost $=$ Cost

MinList $=$ List

return MinCost, MinList

Further improvements to tractability

- Even after discretization, we might take a lot of time to solve the program
  - Around $O(\widehat{gap}(x) \sum_i |\mathcal{X}_i|)$
- Can prune results with two further insights
  - Easily detect when we would require too much cost
  - Discretized coarse metric

First pruning optimization

- Can immediately strike out instances that are "too positive"
- Don't need to spend time finding their minimum camouflage

Theorem

If

$$\max\left(\frac{D_i(x_i, x_i')}{W_i(x_i, x_i')}\right) < \frac{gap(x)}{\Delta U_A}$$

then $\mathcal{A}(x) = x$

- Not intuitive, but the proof is simple

First pruning optimization: proof

- Proof is by simple lower bound

$$
\begin{aligned}
W(x, MCC(x)) &= \sum W_i(x_i, x_i') \\
&= \sum \frac{W_i(x_i, x_i')}{D_i(x_i, x_i')} D_i(x_i, x_i') \\
&\geq \min \left(\frac{W_i(x_i, x_i')}{D_i(x_i, x_i')} \right) \sum D_i(x_i, x_i') \\
&\geq \min \left(\frac{W_i(x_i, x_i')}{D_i(x_i, x_i')} \right) .gap(x) \text{ by 2nd LP constraint}
\end{aligned}
$$

- If this quantity is $> \Delta U_A$, so is the camouflage cost

Second pruning optimization

- Can eliminate redundant checks
- Sort the (i, x_i') tuples in increasing order of $W_i(x_i, x_i')$
- For identical values of $W_i(x_i, x_i')$, only keep the one with largest $D_i(x_i, x_i')$
  - Works because optimal solution is invariant under choice of $D_i(x_i, x_i')$
  - With coarse discretization, remove a lot of pairs from consideration

Summary thus far

- Thus far, we have shown the following

Fact

Given a naive Bayes classifier, it is possible for an adversary to efficiently compute a transformation $\mathcal{A}$, which, given a malicious instance x, maximizes the probability that $x' = \mathcal{A}(x)$ is classified as harmless

  - Since we can *efficiently* compute $\mathcal{A}$, we cannot just ignore the adversarial presence

- So now, the question is what the classifier can do...

Outline

Learner strategy

- So now assume that the adversary has applied $\mathcal{A}$ on the data
- How can the classifier try to foil his plan?
  - In some sense, cannot "trust" the harmless values as much
  - $P(x|+)$ is now suspect

Learner strategy

- Brute force way to deal with adversary?
  - Look at each instance x, and estimate the probability is was modified into the instance we see now
- Denote the new estimate for the conditional probability P_A:

$$P_A(x'|+) = \sum P(x|+)P(x'|x,+)$$

- In terms of the adversarial function $\mathcal{A}$,

$$P_A(x'|+) = \sum_{x \in \mathcal{X}_A(x')} P(x|+)$$

Learner algorithm

- After adjusting the probability, Learner proceeds as normal
- Recall that the classification is whichever class y maximizes

$$U(y|x) = P(+|x)U_C(y,+) + P(-|x)U_C(y,-)$$

- Classifier simply computes $U(+|x), U(-|x)$
  - Estimates P's based on training set

Learner algorithm

Classify(x):

$P(-|x) = P(-) \prod_i P(X_i = x_i|-)$

$P(+|x) = P(+) P_A(x|+)$

for $y = \{+, -\}$

$U(y|x) = P(y|x) U_C(y, +) + P(-|x) U_C(y, -)$

return $\operatorname{argmax}_y U(y|x)$

Simple as that?

- Are we done?
- Estimating the P's can be done easily
- But computing P_A requires summing over $\mathcal{X}_A(x')$
  - Any x whose camouflage is x'
  - This is a very large set...

Problem again?

- Again, computing the set $\mathcal{X}_A(x')$ is intractable
- Trivial simplification: consider $\mathcal{X'}_A(x') = \mathcal{X}_A(x') - x$, and

$$P(x'|+) = \sum_{x \in \mathcal{X'}_A(x')} P(x|+) + \delta_{x'} P(x'|+)$$

  - Easy to check if $x' \in \mathcal{X}_A(x')$

Estimating the set

- How to estimate ${\mathcal{X}'}_A(x')$?
- Again, two pruning rules
  - First try and eliminate those x who *cannot* have x' as a camouflage
  - Then try and bound those x who *must* have x' as a camouflage

Skip to summary of classifier algorithm

First pruning rule

- Use the following theorem

Theorem

If x is a malicious instance, and $x' = MCC(x)$, then for each i,

$$x_i \neq x'_i \implies gap(x) + LO(x_i) - LO(x'_i) > 0$$

That is, those features on which x and x' disagree have the above technical property for their gap

First pruning rule

- What does theorem tell us?
  - If x does *not* satisfy the above property, then $x' \neq MCC(x)$
- Reduce the instances we need to check
  - Only consider those x that satisfy the theorem
  - Rest *cannot* have x' as their camouflage
- Still could have exponentially large search space, though...

Second pruning rule

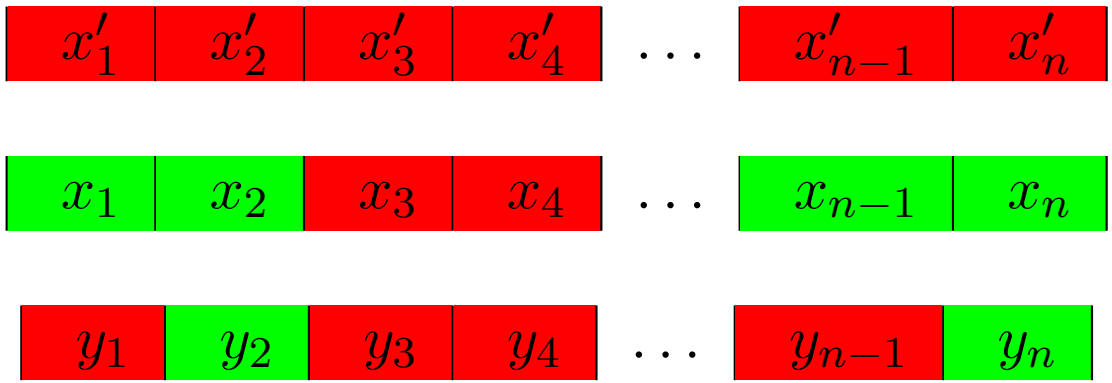
- One can prove that $x' \neq MCC(x)$ says something about other x, too
  - Being a camouflage of x means we are the camouflage for more restricted feature sets
  - Cannot make x' the camouflage of any instance formed by just changing more features
- Uses the following theorem

Theorem

Suppose x is a positive instance, and $x' = MCC(x)$. Let $\mathcal{D}$ be the features changed in x to produce x', and $\mathcal{E} \subseteq \mathcal{D}$. Let x'' be x modified only in the features $\mathcal{E}$. Then, $x' = MCC(x'')$ also.

Visualization

- What does theorem tell us?
  - Say $x' = MCC(x)$, with features $\mathcal{D}$ of x changed to get x'
  - Suppose y is like x, but with some of the features in $\mathcal{D}$ already changed
  - Then $x' = MCC(y)$



Combining the pruning rules

- Let $FV = \{(i, x_i)\}$ be those feature-value pairs we get from the first rule
- Let $x_{[i \to y]}$ denote the instance x with the ith feature value changed to y
- Now, using the second rule, only consider $GV = \{(i, x_i) \in FV\}$ so that

$$x'_{[i \to x_i]} \in \mathcal{X}_A(x')$$

  - The second rule tells us that if $x' = MCC(x)$, then the changes from $x \to x'$ must be contained in GV
- This means that we only check if a subset of feature changes produces a MCC

Combining the pruning rules

- Pruning rules tell us

Theorem

$$\sum_{(i,x_i) \in GV} P(x'_{[i \to x_i]}|+) \leq \sum_{x \in \mathcal{X}'_A(x')} P(x|+)$$

- Use the lower bound as an estimation of the true value

Summary thus far

- Thus far, we have shown the following

Fact

If the learner knows that there is an adversary modifying the data according to $\mathcal{A}(x)$, it is possible to efficiently create a new classifier strategy that minimizes the chance of prediction.

Outline

Experiments

- Experiments on spam filtering
- Two data sets
  - Ling spam: messages on a linguistics mailing list (16.6% spam)
  - Email-data: collection of emails (55.1% spam)
- Three data models
  - Add words: Spammer adds words to fool classifier, each word has unit cost
  - Add length: Add words, except each character has unit cost
  - Synonym: Spammer changes words in document to fool classifier

Add words

- Add words: Spammer adds words to fool classifier, each word has unit cost

Original

We offer cheap high quality watches.

Changed

Bob meeting field prevaricate. We offer cheap high quality watches.

Add length

- Add length: Add words, except each character has unit cost

Original

We offer cheap high quality watches.

Changed

prevaricate We offer cheap high quality watches.

Synonym

- Synonym: Spammer changes words in document to fool classifier

Original

We offer cheap high quality watches.

Changed

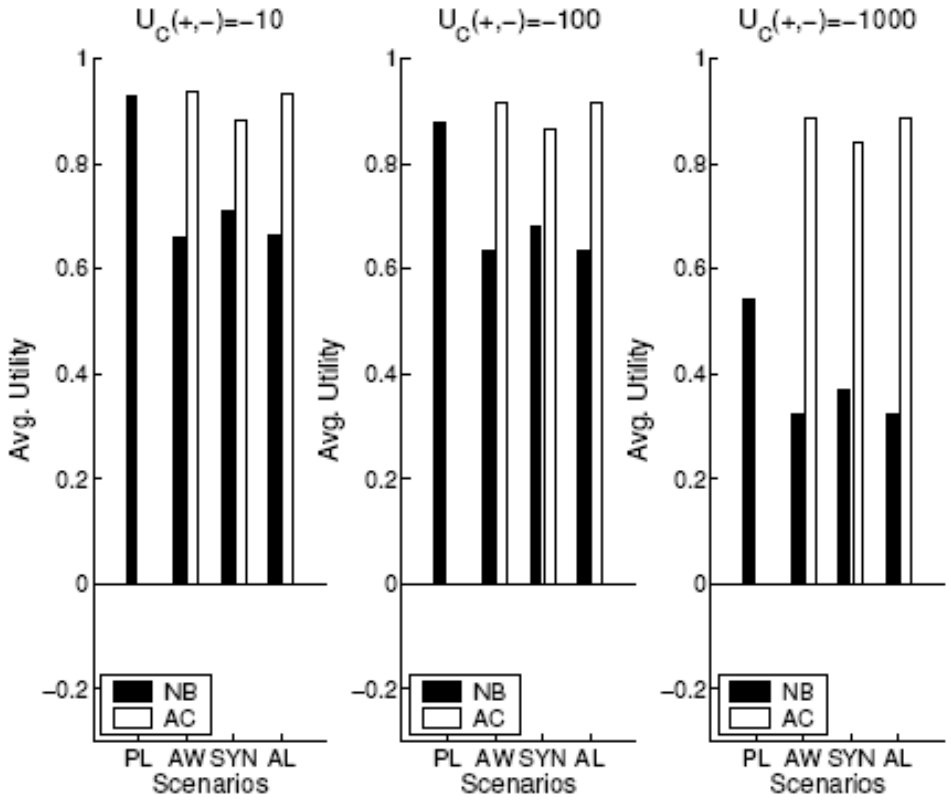
We provide inexpensive high quality watches.

Experimental setup

- Naive Bayes classifier is run on the untampered data
- For each data model, adversary's algorithm is run to compute $\mathcal{A}(x)$
- On the modified data:
  - Run Naive Bayes
  - Run optimal classifier strategy

Results

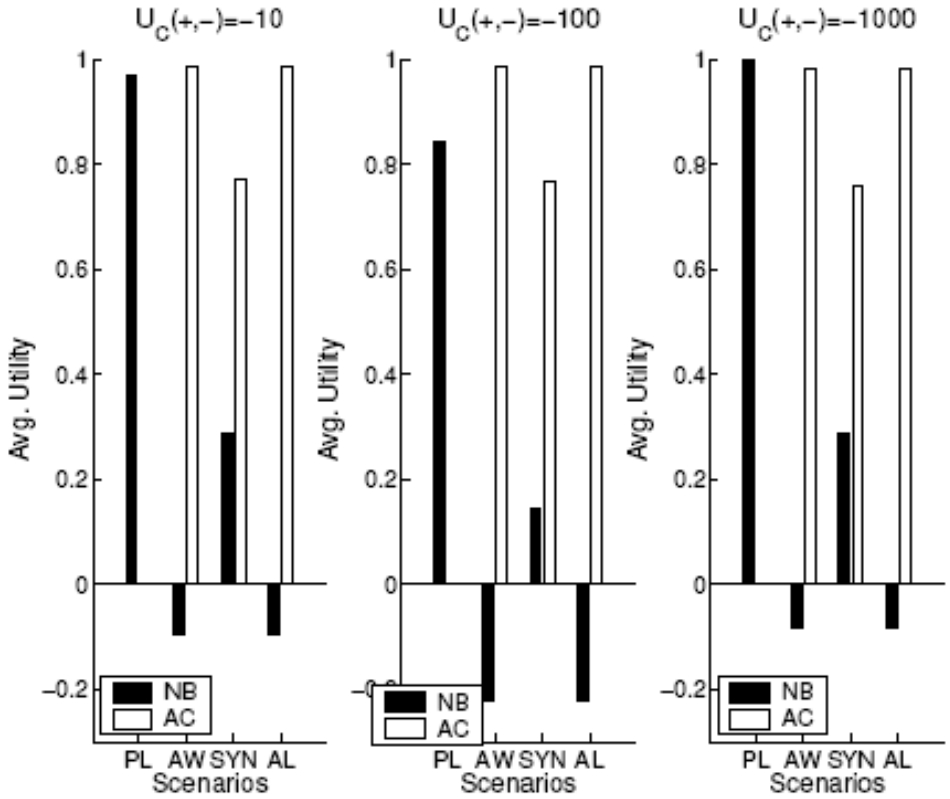
- Ling-spam with different misprediction costs (misclassify malicious as harmless)



- Adversarial classifier significantly improves results ($\sim 40\%$)
- Data model has little effect

Results

- Email-spam with different misprediction costs



- Adversarial classifier significantly improves results ($\sim 90\%$)
- Only Synonym is feasible for adversary-agnostic classifier

Runtime

- Informal comparison of runtime
- Add Length takes the longest time for adversary to compute strategy (~ 500 ms)

Outline

Critique on classifier front

- Only works on Naive Bayes
  - Good start, but other classifiers could also be studied
- Assumes that all parameters are known to classifier and adversary
  - Unrealistic, though they can probably be estimated

Critique on game theory front

- Only for single round of classifier-adversary interaction
  - Does not tell us what happens when adversary responds to the improved classifier
  - Also a good start, but a long-run optimal solution is also important
  - Does not eliminate manual intervention
- Nash equilbrium result seems inevitable
  - Theoretical importance

Subsequent work

- [?] removed assumption of adversary possessing perfect knowledge
  - Studied how adversary could deduce good values for the parameters
- Generally, something of a dead end
  - Hard to study theoretically
  - Unrealistic in practise: naive Bayes assumption itself is a major limitation

Outline

Conclusion

- In many classification problems, data generator is adversarial
- Classifier must try to minimize the damage caused by adversary
  - Or risk performance degradation
- Naive Bayes classifier can be made adversary aware
- Good performance for spam detection
- But can it be used in practise?

Questions?

Outline