

Bayesian Spam Filtering in MMOs

Nandor T. Szots

nszots@soe.sony.com

Introduction

In this lecture we will be looking at spam trends in modern MMOs, why spamming works, why it is in the developers best interest to stop the spammers, and finally how EverQuest II goes about solving this issue.

Any modern MMO can expect between 10,000 and 1 million spam messages to be sent to its player base every day. This has a huge impact on the player experience and compromises fair game play, giving an unfair advantage to those players who illegally use RMT to purchase in-game currency or items.

Three years ago this problem became so prevalent in our games that SOE decided to invest in developing a company-wide solution. The result was a Bayesian filtering system with real-time user feedback and statistical “pushing” using various information about players and accounts available to MMOs.

Using this system, EverQuest II blocks between 30,000 and 500,000 spam messages per day with 99% - 100% accuracy (based on false-positive reports). This is done through a completely automated system that learns and grows in real-time through player feedback. This allows the community to react quickly to new spam signatures. Using a community-based system, instead of an individual one, allows this filter to block new messages as soon as the first few players receive the spam. So for the majority of our players the filter will already be blocking the spam without requiring any intervention. This also allows the community to police anything they feel is spam or allow anything they feel isn't spam but was flagged incorrectly. The advantage of using a game-world-wide system such as this is that it keeps spam-marking from becoming a mandatory “mini-game” for all players.

Why is spam such a huge problem in today's MMOs?

According to SoftPedia and Extreme Gamer, the third-party RMT industry is estimated to be generating \$2 billion annually and 30% of MMO players are estimated to be involved with either the buying or selling of in-game currency or goods. When you combine this information with the unlimited resources generated by MMO worlds, which can be gained simply through repetitive game-play, it becomes clear why this is such an attractive market.

Technical Overview

The SOE Bayesian filter implements what is known as a "Naive Bayes Classifier." This is because each word is interpreted independently of any other words and the entire phrase. The importance of using this type of solution is that it allows us to build a simple database of words, which describe the probability that a word is in a "clean" message and the probability that it is in a "spam" message.

In the case of EverQuest II, these databases were created from a large sample (~122,000 lines and ~1 million words) of normal messages and a smaller sample (~200 lines ~4,500 words) of spam messages. The reason for this huge discrepancy is that the spammers in our game tended to only use a few dozen different messages so this small sample is actually very representative of the spam we see in-game.

Using this set of messages we can create a database describing the probability that any words is in any given message, good or bad. Each word has two entries, one for good probability and one for bad.

Once a working database is created you can use the following Bayesian formula to determine the spam-value of any give message. Assume a Document D (in our case the players message), which we want to classify as either Spam S or Not Spam $\neg S$. We can describe the probability that D is Spam as follows:

$$\ln \frac{p(S|D)}{p(\neg S|D)} = \ln \frac{p(S)}{p(\neg S)} + \sum_i \ln \frac{p(w_i|S)}{p(w_i|\neg S)}$$

Essentially using log-likelihood ratios we start with the \ln of the probability that any message is spam over the probability that it is not spam. This gives us our starting ratio and should remain more or less static unless the spam patterns in your game change drastically. To this static value we add the summation of the \ln of each word being in a spam message divided by the probably of that word being in a clean message (using the DB we built above). This gives us a single number usually in the range of -100 to 100, statistically anything above 0 should be considered spam, in our testing we found that using 5 as our baseline gave much fewer false-positive and had little to no impact on detecting actual spam.

It is generally wise to ignore "glue" words such as "the", "and", "it", etc. We did not implement this and found no practical problems in doing so. However, we only count each word in a statement once, so "Hi hi hi hi there" has the same value as "Hi there."

Simply using this method we can already get a good handle on which messages are spam and which are not. But, we have also created a weakness for ourselves. While making the filter global means that fewer users will ever see any particular type of spam message and the filter learns much faster; it also means that spammers will be able to bounce their messages off the filter that everyone uses to find weaknesses and valid messages. SOE system takes this into account and has added numerous features to help combat this problem, some of which will be discussed in the next section.

When EverQuest II first introduced the spam filter, a typical spam message read as follows:

"Safe Gold, Fast power leveling, Special service of gold farming. Enjoy the fun we give you and your guild at www.gmworker.com"

With-in a few months of using the filter the spammers had to change their message to something much less meaningful just to be able to get through to the players:

“Stick to buy 1000 gold per-day for 5214 weeks on our site, you will be 100 years old, and become the strongest person in the game forever!”

As you can see this new message is much harder to understand, and does not even contain their website address, which was typically sent in a second line of text and in a mangled format:

“Visit \\w\\.g###rk3r.c0/\\, ###=mwo, 3=e”

Even the URL has become almost unintelligible, and as always with-in the first few hours this signature was learned by the filter and became detectable.

Along these same lines the spammers have also tried using Unicode characters to confuse the filter, if your game support Unicode you will probably have to work in a Unicode down converter to solve that issue.

Once you have this solution working for a single game it becomes obvious that some certain keywords need to be generalized for a multi-game solution. Some of what we did includes converting all currency references to keyword “#MONEY# and all in-game currency to #CURRENCY#. In this way it does not matter if they are selling using USD or GBP or EUR and similarly it does not matter if they are selling gold or platinum or unobtainium.

Bayesian Filtering Example

In this section I wanted to give a few examples of how the filter works step by step..

We must now define an example DB of words.

First we will create a Good word values DB:

Type	Word	Value
Good	and	0.099968
Good	buy	0.001099
Good	com	0.001904
Good	did	0.011099
Good	fast	0.001831
Good	#GAMECUR#	0.096825
Good	go	0.018819
Good	hello	0.003216
Good	how	0.018254
Good	lets	0.001871
Good	level	0.010167
Good	leveling	0.000462
Good	power	0.002584
Good	safe	0.000964
Good	see	0.011730
Good	sun	0.000219
Good	the	0.146493
Good	to	0.140442
Good	up	0.024952
Good	was	0.027252
Good	welcome	0.002220
Good	what's	0.000008
Good	www	0.000867
Good	you	0.097830

Second we will create a Bad value DB:

Type	Word	Value
Bad	and	0.143651
Bad	buy	0.214512
Bad	com	0.171429
Bad	did	0.002381
Bad	fast	0.067460
Bad	#GAMECUR#	0.096825
Bad	go	0.034127
Bad	hello	0.018254
Bad	how	0.018254
Bad	lets	0.001587
Bad	level	0.017460
Bad	leveling	0.017460
Bad	power	0.022222
Bad	safe	0.045238
Bad	see	0.010317
Bad	the	0.135714
Bad	to	0.208730
Bad	up	0.012698
Bad	was	0.011905
Bad	welcome	0.027778
Bad	www	0.162698
Bad	you	0.151587

Using these two tables we can now parse our first legitimate statement:

“Hello, what’s up? Did you see how gold the sun was? Lets go power level!”

Phase I: Split out words and remove unneeded characters

“hello what’s up did you see how #GAMECUR# the sun was lets go power level”

Phase II: Parse each word using the Bayesian formula

Starting Value	Bad	Good	ln(Bad/Good)	-2.944438
hello	0.018254	0.003216	1.736245	-1.208193
what's	0.000001	0.000008	-2.079441	-3.287634
up	0.012698	0.024952	-0.675509	-3.963143
did	0.002381	0.011099	-1.539334	-5.502477
you	0.151587	0.097830	0.437928	-5.064549
see	0.010317	0.011730	-0.128356	-5.192905
how	0.018254	0.017855	0.022100	-5.170805
#GAMECUR#	0.096825	0.003338	3.367533	-1.803272
the	0.135714	0.146493	-0.076427	-1.879699
sun	0.000001	0.000219	-5.389071	-7.26877
was	0.011905	0.027252	-0.828168	-8.096938
lets	0.001587	0.001871	-0.164627	-8.261565
go	0.034127	0.018819	0.595221	-7.666344
power	0.022222	0.002584	2.151744	-5.5146
level	0.017460	0.010167	0.540765	-4.973835

So our final spam value for this statement is -4.973835 which is clearly not spam and this message will be allowed through.

Our next example is going to be a spam message using similar words:

“Hello! Welcome to www.buygold.com. Power leveling, and fast safe gold!”

Phase I: Split out words and remove unneeded characters

“hello welcome to www buygold com power leveling and fast safe #GAMECUR#”

Phase II: Parse each word using the Bayesian formula

Starting Value	Bad	Good	ln(Bad/Good)	-2.944438
hello	0.018254	0.003216	1.736245	-1.208193
welcome	0.027778	0.002220	2.526737	1.318544
to	0.208730	0.140442	0.396246	1.71479
www	0.162698	0.000867	5.234612	6.949402
buygold	0.000001	0.000001	0.000000	6.949402
com	0.171429	0.001904	4.500212	11.449614
power	0.022222	0.002584	2.151744	13.601358
leveling	0.017460	0.000462	3.632102	17.23346
and	0.143651	0.099968	0.3632102	17.5966702
fast	0.067460	0.001831	3.606672	21.2033422
safe	0.045238	0.000964	3.848601	25.0519432
#GAMECUR#	0.096825	0.003338	3.367533	28.4194762

In this case our spam value is 28.4194762 which is very clearly spam, this message would be logged and blocked by the filter.

MMO Specific Filter Changes

In order to increase the accuracy of the filter and decrease as many false positives as possible, SOE has made some MMO specific changes on top of the Bayesian filtering system.

Part of these changes discussed earlier was the key-word replacement of game-currency units. This allows for more accurate detection as well as easier portability between games. This type of rule can easily be applied to any genre-specific concepts that are only slightly different between games.

Since MMOs typically have persistent accounts, we can use this extra information to our advantage. For example, Account Age, Account IP, Account Payment Type, Character Age, Character Level, etc. can all be used to set our Bayesian starting point more accurately for any given message. More over these starting values can be mathematically computed using game data so they will highly accurate.

Additionally player relationships can also be used to determine the probability of spam. Things like friends lists, guilds, and groups all tell us about the implied trust between two characters who are messaging back and forth. Using this information can greatly decrease the number of false positives that your players will see.

Because spammers can change their messages in real-time, we need to be able to adapt in real-time. This is where real-time word DB updates and user-feedback scrapping come into play. It is imperative that users be able to mark things as spam or not-spam in a reasonable timeframe. Part of this feedback system has to be trust-rules that the server has in place to decide which user reports to trust and which ones to toss. Trust rules can be anything from payment method, to account age, to character level. Anything that gives us a hint that the reports are likely legitimate can be used.

There are also disadvantages that are unique to MMOs. Spammers will be able to pre-test their message using two game clients and bouncing messages off the filter until they find one that works. The best solution we have found for these types of attacks is using IP address information to allow spam messages through as not spam to similar hosts. This logic can be expanded to be based on any account information that is similar between the two clients, such as billing information, addresses, etc.

Conclusion

Using the information in this presentation anyone can build a solid Bayesian-based filter implementation for stopping spam. The keys to a successful system are based around real-time user feedback, in conjunction with real-time server updates and as much account specific information as your game has available.