As discussed in Adams and Winter (1997), there are certain shorthands used in gang graffiti to convey affiliation or threats. The addition of a <k> after a letter representing a rival gang stands for "killer." For example, <ck> would represent "crip killer." A summary of these substitutions can be seen in Figure 2. These features were created by a rule-based algorithm that compares words against a standard dictionary as well as using some phonotactic constraints on the position of certain letters. The value of the measure represents the frequency of words containing the substitution in the average post.

Transformation	Origin or meaning
b^, c^, h^, p^	"Bloods up" Positive towards Bloods, Crips, Hoovers, Pirus, respectively
b → bk, c → ck h → hk, p → pk	Blood killer, Crip killer Hoover killer, Piru killer
b → c	Replace 'b' for Blood with 'c' for Crip
c → b	Replace 'c' for Crip with 'b' for Blood
ck → cc, kc	Avoid use of 'ck' since it represents Crip killer
o → x, o → ø	Represents crosshairs, crossing out the '0's in a gang like Rollin' **60s** Crips
b → 6	Represents the six-pointed star. Symbol of Folk Nation and the affiliated Crips.
e → 3	Various. One is the trinity in Trinitario.
s → 5	Represented the five-pointed star. Symbol of People Nation and the affiliated Bloods.

Table 1: Orthographical substitutions from gang graffiti symbolism

These features had a few different ways of being encoded. The two that made the most sense were counting the number of substitutions and comparing it to the number of words in which those substitutions could've occurred, or simply comparing it to the number of words in the document total. The former measure would sometimes result in division by zero if there were no opportunities for substitutions, and so we would leave those values blank in that case. Based on our experiments, the two measures performed similarly, but the latter did not have the division by zero problem which was a problem for users with little data.

We used a set of 13 of these features, chosen on the basis of how frequently they occurred and how strongly they were associated with particular gangs (for example, substituting '$' for 's' does not seem to be gang specific). Detection of the features works by removing words that occur in the AQUAINT corpus (the dictionary), then applying regular expressions to detect occurrences of the characters of interest. If there's a match, the substitution is reversed and passed to the next step. The features are done roughly in the

order of least likely to occur in normal text to the most likely. Since ‘bk’ only occurs in a handful of obscure words, for example, almost any occurrence of it can be assumed to be a substitution and the ‘k’ can safely be removed before the next step. By contrast, ‘cc’ and ‘ck’ occur in many common words so they must be saved for last to ensure that the final dictionary checks have any simultaneous substitutions already removed.

The feature coding was developed iteratively. After running the features on the posts, we created lists of words where the features were detected, sorted by frequency. I then manually examined the words to determine where the most errors were occurring and calculated a an estimated precision value on the more common words that represent the bulk of the detected words. Precision was deemed more important than recall in this case since most users don’t use any given substitution feature, and those that do tend to make use of them in many words. It was thus seen as better to slightly underestimate a Crips’ use of ‘cc’, for example, than to give a non-Crip that happened to frequently use a regular word with ‘cc’ an inflated score.

Performance of the style features was estimated by first creating a list of all the words marked by the program as containing a style feature and a list of all the remaining words that occurred that were not in the dictionary. The false positive rate was calculated for each feature by taking the words that occurred more than once, and then manually examining the top 200 most common words among those. Since many words would tie, additional words were generally included. It was also ensured that at least 55% of the total tokens were covered, so additional words may have been examined. The estimated precision for each feature on the development set and the test set is given below:

	Dev	Dev Set	Dev Set	Test	Test Set	Test Set
Feature	True tokens evaluated	Estimated False Positives	Estimated False Negatives	True tokens evaluated	Estimated False Positives	Estimated False Negatives

cc	72086	1.51%	0%	6383	8.67%	0%
bk	26923	0.85%	0%	755	3.70%	0%
ck	16144	25.25%	7.80%	1325	32.41%	4.60%
s>5	13754	6.32%	0%	158	26.17%	0%
x'd out letter	13646	14.97%	5.25%	1219	38.62%	15.91%
o>x	11395	26.02%	1.02%	767	54.61%	1.17%
hk	10053	0.82%	0%	115	14.18%	0%
e>3	8628	28.36%	0.52%	549	32.41%	6.56%
pk	5669	2.76%	0%	317	7.85%	0%
b>6	2470	16.40%	0.28%	15	54.55%	33.33%
b^	5325	0%	0%	36	0%	0%
p^	719	0%	0%	15	0%	0%
c^	3684	0%	0%	8	0%	0%
h^	5375	0%	0%	5	0%	0%

This is a description of the basic procedure for the style features. I wasn't sure how concise it should be, or what format the regular expressions should be, etc. The regular expressions are in Python's format right now. The steps must be applied in that order.

Style features procedure:

0. If the word or its altered result is in the dictionary at any point, skip further steps.
(i.e. add "if passed word is not in the dictionary" to the beginning of each step).

1a. Split word on dashes, and pass each half back to step 0.

1b. Replace "cck" or "ckk" with "ck" (for dictionary matching purposes) and pass result to next step.

1c. Replace "ø" with "o" and pass result to next step.

2. If word contains "b^", "c^", "h^" or "p^":
Add to b^, c^, h^ or p^ count, respectively.
Remove any "^" and pass result to next step.

3. If word contains “bk”:
Add to bk count.
Replace “bk” with “b” and pass result to next step.

4-5. Repeat step 3 for “hk” and “pk”.

6. If word matches:
“h[0-9][0-9][0-9]?d[sz]*”
“bl[0-9][0-9][0-9]?d[sz]*”
“(h|gr)[0-9][0-9][0-9]?v[a-z]*”

Replace “[0-9][0-9][0-9]?” with “oo” and pass result to next step.

7. If word matches “(.*[a-z]5?5([^0-9].*)?|(.*[^0-9])?55?[a-z].*)$”:
Add to 5s count.
Replace “5” with “s” and pass result to next step.

8. If word matches “(.*[a-z]3?3([^0-9].*)?|(.*[^0-9])?33?[a-z].*)$”:
Add to 3e count.
Replace “3” with “e” and pass result to next step.

9. If word matches “(.*[aeiou][rlm]?6?6([^0-9].*)?|(.*[^0-9])?66?[rl]?[aeiouy].*)$”:
Add to 6b count.
Replace “6” with “b” and pass result to next step.

10a. If word matches “.*xx.*”:
Add to xo count.
Replace “xx” with “oo” and pass to next step.

10b. If word matches “x.*” or “.*[^aeiou]x.*”:
Add to xo count.
Replace that “x” with “o” and pass to next step.

11. If word contains “cc” or “kc” and does not match “n[iu]cca+[sz]*”:
Add to cc count.

12a. If word matches “n[uix]cka+[sz]?” or
“m[oua][ftherauodzv]{0,4}f[aiou][ckg]+([aenoiusr][a-z]*)?” or
“f[eiou*]{0,1}[ckg]*ck[ckg]*([aenoius][a-z]*)?” (nicka RE, motherfucker variant RE, fuck variant RE):
End.

12b. If word contains “ck” and word matches “.{2,}ck.+”:
Split at first ck, and pass each half back to step 15a.

12c. If word contains “ck”:
Add to ck count.