CS540 Intro to AI

Natural Language and Statistics

Lecturer: Xiaojin Zhu *jerryzhu@cs.wisc.edu*

1 Common Preprocessing Steps

Counting words alone gives interesting information. This is known as unigram word count (or word frequency, when normalized). For example Amazon concordance for the book *The Very Hungry Caterpillar* by Eric Carle shows high frequency *content* words "hungry, ate, still, caterpillar, slice, ..." Another interesting website is wordle.net.

How do we count words? First we need to define what a word is. This is highly non-trivial for languages without space like Chinese. Even for seemingly simple English, getting text into a form where you can count words is quite involved, as we see below.

Preprocessing text is called **tokenization** or **text normalization**. Things to consider include

- Throw away unwanted stuff(e.g., HTML tags – but sometimes they are valuable, UUencoding, etc.)
- Word boundaries: white space and punctuations – but words like *Ph.D.*, *isn't*, *e-mail*, *C|net* or *$19.99* are problematic. If you like, we can spend a whole semester on this... This is fairly domain dependent, and people typically use manually created regular expression rules.
- Stemming (Lemmatization): This is optional. English words like 'look' can be *inflected* with a morphological *suffix* to produce 'looks, looking, looked'. They share the same *stem* 'look'. Often (but not always) it is beneficial to map all inflected forms into the stem. This is a complex process, since there can be many exceptional cases (e.g., department vs. depart, be vs. were). The most commonly used *stemmer* is the Porter Stemmer. There are many others. They are not perfect and they do make mistakes. Many are not designed for special domains like biological terms. Some other languages (e.g., Turkish) are particularly hard.
- Stopword removal: the most frequent words often do not carry much meaning. Examples: "the, a, of, for, in, ..." You can also create your own stopword list for your application domain. If we count the words in *Tom Sawyer*, the most frequent word types are (from [MS p.21]):

Word	Count
the	3332
and	2972
a	1775
to	1725
of	1440
was	1161
it	1027
in	906
that	877
he	877

  For many NLP purposes (i.e. text categorization) they are a nuisance, and stopword removal is a common preprocessing step. SMART is such a stopword list.

- Capitalization, case folding: often it is convenient to lower case every character. Counterexamples include 'US' vs. 'us'. Use with care.

People devote a large amount of effort to create good text normalization systems.

Now you have clean text, there are two concepts:

- **Word token**: occurrences of a word.
- **Word type**: unique word as a dictionary entry.

For example, "The dog chases the cat." has 5 word tokens but 4 word types. There are two tokens of the word type "the".

A **vocabulary** lists the word types. A typical vocabulary has 10,000 or more words (types). Sometimes also in the vocabulary are `<s>` for the special start-of-sentence symbol, `</s>` for end-of-sentence, and `<unk>` for all out-of-vocabulary words (they all map to this symbol).

A corpus is a large collection of text, e.g., several years' newspapers. A vocabulary can be created from a corpus. Often people apply a *frequency cutoff* to exclude word types with small counts (see below). The cutoff is usually determined empirically (anywhere from one to tens or more).

2 Zipf's Law

If we rank word types by their count in *Tom Sawyer*, and compute count $\times$ rank, we see an interesting pattern:

Word	Count f	rank r	fr
the	3332	1	3332
and	2972	2	5944
a	1775	3	5235
he	877	10	8770
but	410	20	8400
be	294	30	8820
there	222	40	8880
one	172	50	8600
two	104	100	10400
turned	51	200	10200
comes	16	500	8000
family	8	1000	8000
brushed	4	2000	8000
Could	2	4000	8000
Applausive	1	8000	8000

We see that $fr \approx$ constant, or $f \propto \frac{1}{r}$. If we plot $\log(r)$ on the x-axis and $\log(f)$ on the y-axis, the words roughly form a line from upper-left to lower-right. Note f can be the frequency (count divided by corpus size) and the relation still holds. This relation is known as *Zipf's law*. It holds for a variety of corpora. Mandelbrot generalizes Zipf's law with more parameters P, ρ, B so it is more flexible: $f = P(r + \rho)^{-B}$.

3 Miller's Monkeys (* Optional)

If we promise a monkey some bananas and ask it to type tirelessly on a computer keyboard, what do we get?[1] For simplicity, let us assume the keyboard has 27 keys: a to z, and white space. We also assume the

[1] No, not a software engineer.

monkey hit each key with equal probability. Let us call a sequence of letters separated by white space a 'word'. What frequency and rank relation do such monkey words possess?

The probability that a specific monkey word type has length i is

$$P(i) = (1/27)^i(1/27) = (1/27)^{i+1}. \tag{1}$$

As we can see, the longer the word, the lower its probability – therefore the lower the expected count in the monkey corpus. Let us rank all monkey words by its probability. The number of monkey word-types with length i is 26^i. The rank r_i of a word with length i thus satisfies

$$\sum_{j=1}^{i-1} 26^j < r_i \leq \sum_{j=1}^{i} 26^j \tag{2}$$

Let us consider the word with rank $r = \sum_{j=1}^{i} 26^j$. The word actually has length i, but from

$$r = \sum_{j=1}^{i} 26^j = \frac{26}{25}(26^i - 1), \tag{3}$$

we can derive a 'fractional length' i'

$$i' = \frac{\log\left(\frac{25}{26}r + 1\right)}{\log 26}. \tag{4}$$

The frequency of this word is

$$\begin{aligned}
p(i') &= (1/27)^{i'+1} && (5)\\
&= (1/27)^{\frac{\log\left(\frac{25}{26}r+1\right)}{\log 26}+1} && (6)\\
&= (1/27)\left(\frac{25}{26}r + 1\right)^{-\frac{\log 27}{\log 26}} \quad \text{using the fact } a^{\log b} = b^{\log a} && (7)\\
&\approx 0.04(r + 1.04)^{-1.01}, && (8)
\end{aligned}$$

which fits Mandelbrot's law, and is fairly close to Zipf's law.

In light of the above analysis, Zipf's law may not reflect some deep knowledge of languages. Nonetheless, it still points to an important empirical observation, that almost all words are rare. This is also known as the heavy tail property.

4 Basic Probability and Statistics

In this section we use letters instead of words for illustration. Generalization to words is straightforward. Pick an arbitrary letter x at random from any English text ever written. What is x?

Random variable X: the random outcome of an experiment. (it really is a function mapping the outcome space to a number, but we will not go there...)

Probability distribution $P(X = x) = \theta_x$, for $x \in \{1, \ldots, 27\}$ if our alphabet consists of $\{a, b, c, \ldots, z, \sqcup\}$. Often use the probability vector $\theta = (\theta_1, \ldots, \theta_{27})^\top$.

Basic properties: $0 \leq \theta_x \leq 1$, $\sum_x \theta_x = 1$.

Useful analogy of θ: a k-sided die. Here $k = 27$. If $k = 2$, a coin. A *fair* die/coin is when $\theta_1 = \ldots = \theta_k = 1/k$, otherwise it is *biased*.

Sampling a letter x from the distribution θ ($x \sim \theta$):

1. create intervals of lengths $\theta_1, \ldots, \theta_k$.
2. generate a uniform random number $r \in [0, 1]$ (most programming languages have a random() function for this).
3. find the interval r falls into, output the interval index.

The **Multinomial distribution** for k-sided die with probability vector θ, N throws, outcome counts $n_1, \ldots, n_k$:

$$P(n_1, \ldots, n_k|\theta) = \binom{N}{n_1 \ldots n_k} \prod_{i=1}^{k} \theta_i^{n_i}. \tag{9}$$

Going back to language, if we ignore letter order, we can model the letters as draws from a multinomial distribution.

A **corpus** (*pl.* corpora) is a large, representative collection of text. See the Linguistic Data Consortium (LDC) for some examples.

The **likelihood function** is $P(\text{Data}|\theta)$. Here Data=$(n_1, \ldots, n_k)$, and the likelihood takes the form of (9). Likelihood is *a function of* θ, and in general is not normalized: $\int P(\text{Data}|\theta)d\theta \neq 1$.

Conditional probability $P(X = x|H = h)$. Given the latest letter is h, what is the next letter x?

How do we estimate conditional probabilities? Given a corpus with multiple sentences:

```
i am a student
i like this class
...
```

Let's add a special symbol "start-of-sentence" $\langle s \rangle$ to each sentence:

```
⟨s⟩ i am a student
⟨s⟩ i like this class
⟨s⟩...
```

and break each sentence into pairs of letters. Note we don't cross sentence boundaries, e.g., no (t $\langle s \rangle$):

($\langle s \rangle$ i) (i ⊔) (⊔ a) (a m) ... (n t)
($\langle s \rangle$ i) (i ⊔) (⊔ l) (l i) ... (s s)
...

Now our random variables x, h take value in $\{\langle s \rangle, a, \ldots, z, \sqcup\}$. Let c_{hx} be the count of the pair (hx) in the corpus. We want to estimate the parameters $\theta = \{\theta_{hx} = P(x|h)\}$. Note this θ is much bigger! We can arrange it in a *transition matrix* with rows for h and columns for x.

The maximum likelihood estimate (MLE) of θ is

$$\hat{\theta}_{hx} = \hat{P}(x|h) = c_{hx} / \sum_{x'} c_{hx'}.$$

Since a sentence always starts with $\langle s \rangle$, the likelihood function $P(\text{Data}|\theta)$ is

$$\begin{aligned} & P(i|\langle s \rangle)P(\sqcup|i) \ldots P(t|n)P(i|\langle s \rangle) \ldots & (10) \\ = \; & \theta_{\langle s \rangle i}\theta_{i\sqcup} \ldots & (11) \\ = \; & \prod_{h,x} \theta_{hx}^{c_{hx}}, & (12) \end{aligned}$$

with the constraints $0 \leq \theta_{hx} \leq 1$, $\sum_x \theta_{hx} = 1, \forall h$. One can solve the MLE with Lagrange multiplier as before.

The **joint probability** $P(x, h) = P(h, x)$ is the (much rarer event) that the two events both happening. $\hat{P}(x, h) = c_{hx} / \sum_{h,x} c_{hx}$.

The **marginal probability** is obtained by summing over some random variables. For example, $P(h) = \sum_x P(h, x)$. Relation:

$$P(x, h) = P(x|h)P(h). \tag{13}$$

Now, h and x do not have to be the same type of random variables. Consider this: randomly pick a word, and randomly pick a letter from the word. Let h be the length (number of letters) of the word, and x be the identity of the letter. It is perfectly fine to ask for $P(x = a|h = 2)$. What if you are told the letter is 'a', and you have to guess the length of the word?

The **Bayes rule** allows you to flip the conditional probability around:

$$P(x|h) = \frac{P(x, h)}{P(h)} = \frac{P(h|x)P(x)}{\sum_{x'} P(h|x')P(x')} = \frac{P(h|x)P(x)}{P(h)}. \tag{14}$$

5 Parameter Estimation (Statistics) = Learning a Model (Machine Learning) (* Optional)

Now given N draws from an unknown θ, and we observe the count histogram $n_1, \ldots, n_k$, can we *estimate* θ? In addition, how do we predict the next draw?

Example 1 *If in $N = 10$ coin flips, we observe $n_1 = 4$ heads, and $n_2 = 6$ tails, what is θ?*

Intuition says $\theta = (0.4, 0.6)$. But in fact pretty much any θ could have generated the observed counts. How do we pick one? Should we pick one?

Parameter estimation and future prediction are two central problems of statistics, and machine learning.

5.1 The MLE Estimate

The **Maximum Likelihood Estimate (MLE)** is

$$\theta^{MLE} = \text{argmax}_\theta P(\text{Data}|\theta). \tag{15}$$

Deriving the MLE of a multinomial (V is the same as k, c is the same as m):

$$\begin{aligned}
\theta^{ML} &= \arg \max_{\theta \in V-\text{simplex}} P(c_{1:V}|\theta) && (16)\\
&= \arg\max_\theta \prod_{w=1}^{V} \theta_w^{c_w} \quad \text{multinomial definition} && (17)\\
&= \arg\max_\theta \sum_{w=1}^{V} c_w \log \theta_w \quad \log() \text{ monotonic} && (18)
\end{aligned}$$

We are faced with the constrained optimization problem of finding $\theta_{1:V}$:

$$\begin{aligned}
\max_{\theta_{1:V}} \quad & \textstyle\sum_{w=1}^{V} c_w \log \theta_w && (19)\\
\text{subject to} \quad & \textstyle\sum_{w=1}^{V} \theta_w = 1. && (20)
\end{aligned}$$

The general procedure to solve equality constrained optimization problems is the following: We introduce a scalar β called a *Lagrange multiplier* (one for each constraint), rewrite the equality constraint as $E(x) = 0$, and define a new Lagrangian function of the form $G(x, \beta) = F(x) - \beta E(x)$, where $F(x)$ is the original objective. Solve for the *unconstrained* optimization problem on G.

In our case, the Lagrangian is

$$\sum_{w=1}^{V} c_w \log \theta_w - \beta \left(\sum_{w=1}^{V} \theta_w - 1 \right) \tag{21}$$

After verifying that this is a concave function, we set the gradient (w.r.t. $\theta_{1:V}$ and β) to zero:

$$\frac{\partial}{\partial \theta_w} = \frac{c_w}{\theta_w} - \beta = 0 \tag{22}$$

$$\frac{\partial}{\partial \beta} = \sum_{w=1}^{V} \theta_w - 1 = 0, \tag{23}$$

which gives

$$\theta_w^{ML} = \frac{c_w}{\sum_{w=1}^{V} c_w} = \frac{c_w}{|C|}, \tag{24}$$

where $|C| = \sum_{w=1}^{V} c_w$ is the length of the corpus. It can be seen that the purpose of β is normalization. Therefore, in this case the MLE is simply the frequency estimate!

5.2 The MAP Estimate

The **Maximum A Posterior (MAP)** is

$$\theta^{MAP} = \text{argmax}_\theta P(\theta|\text{Data}) = \text{argmax}_\theta P(\theta) P(\text{Data}|\theta) \tag{25}$$

We need a **prior** distribution $P(\theta)$, which is usually taken to be the Dirichlet distribution since it is conjugate to multinomial:

$$P(\theta) = Dir(\theta|\alpha_1, \ldots, \alpha_k) = \frac{\Gamma(\sum_{i=1}^{k} \alpha_i)}{\prod_{i=1}^{k} \Gamma(\alpha_i)} \prod_{i=1}^{k} \theta_i^{\alpha_i - 1} \tag{26}$$

The posterior is again a Dirichlet

$$P(\theta|\text{Data}) = Dir(\theta|\alpha_1 + n_1, \ldots, \alpha_k + n_k) = \frac{\Gamma(\sum_{i=1}^{k} \alpha_i + n_i)}{\prod_{i=1}^{k} \Gamma(\alpha_i + n_i)} \prod_{i=1}^{k} \theta_i^{\alpha_i + n_i - 1} \tag{27}$$

The mode of the posterior (i.e., the MAP estimate) is

$$\theta_i = \frac{n_i + \alpha_i - 1}{N + \sum_{j=1}^{k} \alpha_j - k} \tag{28}$$

5.3 The Bayesian Approach

Keep the posterior distribution $P(\theta|\text{Data})$.

6 Bag of Words, tf.idf, and Cosine Similarity

A document d can be represented by a word count vector. The length of the vector is the vocabulary size. The ith element is the number of tokens for the ith word type. This is known as the Bag of word (BOW) representation, as it ignores word order. A variant is a vector of binary indicators for the presence of each word type.

A document d can also be represented by a $tf \cdot idf$ vector. tf is the *normalize term frequency*, i.e. word count vector scaled, so that the most frequent word type in this document has a value of 1:

$$tf_w = \frac{c(w,d)}{\max_v c(v,d)},$$

where $c(w,d)$ is the count of word w in d. idf is inverse document frequency. Let N be the number of documents in the collection. Let n_w be the number of documents in which word type w appears.

$$idf_w = \log \frac{N}{n_w}.$$

idf is an attempt to handle stopwords or near-stopwords: if a word (like "the") appears in most documents, it is probably not very interesting. Since $n_w \approx N$, its idf will be close to 0. Finally

$$tf \cdot idf_w = tf_w \times idf_w.$$

Given two documents d, q in feature vector representation, one way to define how similar they are is the *cosine similarity*: q and d form an angle θ in the feature space, and

$$\begin{aligned} sim(d,q) &= \cos(\theta) && (29)\\ &= \frac{d^\top q}{\|d\| \cdot \|q\|} && (30)\\ &= \frac{d^\top q}{\sqrt{d^\top d}\sqrt{q^\top q}}, && (31) \end{aligned}$$

where the dot product

$$u^\top v = \sum_{i=1}^{V} u_i v_i.$$

7 The Need for Finding the More Likely Text

In this lecture we will define the probability of a whole sentence or document, using the so-called n-gram approximation. We will also see why the MLE can be bad at times, and ways to smooth it. Consider these problems:

- Text input on a smart phone. How does the phone know the correct word when you mistype, or type by swiping?
- Say you're designing a speech recognizer. Why do you prefer

  s_1 =It's hard to recognize speech

  over

  s_2 =It's hard to wreck a nice beach?

  The two sentences have nearly the same acoustic signal. We must have some intrinsic preference over sentences.
- Similarly, if you are building an Optical Character Recognition (OCR) system, there are plenty of ambiguity in input (e.g., 1 vs. l, 0 vs. O). But we know that certain letter sequences are more likely than others.

Language modeling tries to capture the notion that some text is more likely than others. It does so by estimating the probability $P(s)$ of any text s. Given input signal a (be it smartphone input, acoustic sound, or digitized pixels), the recognition process can be defined as finding the text s that maximizes, among all texts, the conditional probability

$$s^* = \arg\max_{s \in \mathcal{S}} P(s|a). \tag{32}$$

By Bayes rule, we have

$$P(s|a) = \frac{P(a|s)P(s)}{P(a)}. \tag{33}$$

Note $P(a)$ is a constant w.r.t. the maximization over s, so that the problem reduces to

$$s^* = \arg\max_{s \in \mathcal{S}} P(a|s)P(s). \tag{34}$$

The term $P(a|s)$ describes the probability of producing signal a when the underlying text is s. In speech recognition this is known as the *acoustic model*, and is often a Hidden Markov Model (HMM). The term $P(s)$ is the *language model*, which we discuss now.

In the following we assume words are the basic units, i.e. $s = w_1 \dots w_n \equiv w_{1:n}$. Estimating $P(s)$ directly by counting is not feasible, if n is large. We will use the *chain rule*: [2]

$$P(s) = P(w_{1:n}|\theta) = P(w_1|\theta)P(w_2|w_1,\theta)P(w_3|w_{1:2},\theta)\dots P(w_n|w_{1:n-1},\theta). \tag{35}$$

This does not buy us anything yet – the conditional probabilities are impossible to estimate if the history is long. We will look at n-gram language modeling, which approximates the chain rule by shortening the histories.

8 Unigram Language Model

A *unigram* (1-gram) language model makes the strong *independence assumption* that words are generated independently from a multinomial distribution θ (of dimension V, the size of the vocabulary). That is,

$$P(w_{1:n}|\theta) = \prod_{i=1}^{n} P(w_i|\theta) = \prod_{w=1}^{V} \theta_w^{c_w}, \tag{36}$$

where c_w is the count of word w in s. This is the multinomial distribution over c, except that we do not have the combinatorial coefficient, because we know the particular sequence. Note there is no history or conditional probability at all. The unigram approximation is

$$P(w_i|w_{1:i-1},\theta) \approx P(w_i|\theta).$$

You might be alarmed that this seems to be a particularly bad model of language, as it ignores word orders! You are right – more on this later. But it is *useful*.

Where do we get θ? We estimate it from a corpus. We need a vocabulary of V word types, and counts $c_{1:V}$ of each word type in the corpus. The MLE is

$$\theta_w^{ML} = \frac{c_w}{\sum_{w=1}^{V} c_w} = \frac{c_w}{|C|}, \tag{37}$$

where $|C| = \sum_{w=1}^{V} c_w$ is the length of the corpus. In this case the MLE is simply the frequency estimate, as we've seen before.

There is a tiny problem with the MLE: if a word type w (e.g., aardvark) is in the vocabulary but not in the corpus (hence $c_w = 0$), the MLE is $\theta_w = 0$. Any new sentence with aardvark in it will thus be considered impossible since it have zero probability. This does not make sense. No problem — this can be avoided by restricting the vocabulary to the words in the corpus – for now. For document classification, people found that unigram is often sufficient.

[2] Technically this is not a complete definition of $P(s)$ if s varies in length: there should be a distribution over sentence length $P(n)$, and $P(s) = P(n)P(w_{1:n})$.

9 N-gram Language Models

A better approximation to the chain rule is to keep *some* history. In particular, for an n-gram language model the approximation is

$$P(w_i|w_{1:i-1}) \approx P(w_i|w_{i-n+1:i-1}). \tag{38}$$

The conditioning part $w_{i-n+1:i-1}$ is called 'history', which has $n-1$ previous words. Compared to the chain rule (35), the n-gram language model dictates that

$$P(w_{1:n}|\theta) = \prod_{i=1}^{n} P(w_i|w_{i-n+1:i-1}, \theta). \tag{39}$$

Unigram is a special case when $n = 1$. Common names include bigram ($n = 2$) and trigram ($n = 3$). Trigrams was the standard for speech recognition circa 1990. N-grams traditionally do not run across sentences.

It is worth noting that the number of parameters in θ grows rapidly as $O(V^n)$. Another way to look at it is that for a given history h, $P(w|h)$ is a multinomial of size V, but there are V^{n-1} such possible histories, and θ consists of all these multinomials. For $V = 10,000$ which is typical, in theory there are 10,000-1 unigram parameters, 10^8 bigram parameters (compared to US population 3×10^8), and 10^{12} trigrams (about 150 trigrams per person in the world). In practice, the number of n-grams is bounded by corpus length. In 2006, Google released a "Web 1T 5-gram" corpus on 6 DVDs, with counts from 10^{12} tokens. `http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2006T13` In the Google 1T 5-gram corpus:

```
Number of tokens:    1,024,908,267,229
Number of sentences:    95,119,665,584
Number of unigrams:         13,588,391
Number of bigrams:         314,843,401
Number of trigrams:        977,069,902
Number of fourgrams:     1,313,818,354
Number of fivegrams:     1,176,470,663
```

It should be clear that the data sparseness problem is much more severe with a larger n, because the histories "fragment" the corpus. There are two issues:

1. Some history might not appear in corpus at all.
2. For a history that does appear, not all word types will follow it.

The MLE is in big trouble now. A more satisfying solution is known as smoothing in the language modeling community, which is related to shrinkage in statistics, or, in certain cases, arises from an estimate other than MLE as we show next.

10 Smoothing

There are a large number of smoothing methods for language modeling, e.g., Good-Turing, Jelinek-Mercer interpolated, Katz, Whitten-Bell, Absolute discounting, and Kneser-Ney. We introduce a particularly simple method: add-1 smoothing:

$$\hat{\theta}_w = \frac{c_w + 1}{|C| + V}, \tag{40}$$

which is also known as Laplace smoothing, with which Laplace allegedly computed the probability that the Sun will rise again tomorrow. A simple variation is called add-ϵ smoothing:

$$\hat{\theta}_w = \frac{c_w + \epsilon}{|C| + V\epsilon}, \tag{41}$$

where one chooses the value of ϵ (usually small).

11 Sampling from a Distribution

We can generate sentences by sampling from the distribution $P(s)$ described by a language model. We write $s \sim P(s)$ if we select a sentence s at randomly according to the probability $P(s)$. For a unigram model, the sampling process is as simple as generating random numbers (word index) from the multinomial $P(w|\theta)$. For n-gram models, we sample from the conditional probability $P(w_i|w_{i-n+1:i-1}, \theta)$ where $w_{i-n+1:i-1}$ are the most recent $n-1$ words we just generated.

If this reminds you of a random walk on a Markov Chain, you are right. An n-gram LM is a $(n-1)$-th Markov model, θ is the transition matrix.

Shakespeare unigram (JM p.203):

- To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have
- Every enter now severally so, let
- Hill he late speaks; or! a more to leg less first you enter
- Will rash been and by I the me loves gentle me not slavish page, the and hour; ill let
- Are where exeunt and sighs have rise excellency took of .. sleep knave we. near; vile like

Bigram:

- What means, sir. I confess she? then all sorts, he is trim, captain.
- Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.
- What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman?
- Enter Menenius, if it so many good direction found'st thou art a strong upon command of fear not a liberal largess given away, Falstaff! Exeunt
- Thou whoreson chops. Consumption catch your dearest friend, wekll, and I know where many mouths upon my undoing all but be, how soon, then; we'll execute upon my love's bonds and we do you will?

Trigram:

- Sweet prince, Falstaff shall die. Harry of Monmouth's grave.
- This shall forbid it should be branded, if renown made it empty.
- What is't that cried?
- Indeed the duke; and had a very good friend.
- Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.

4-gram:

- King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;
- Will you not tell me who I am?
- It cannot be but so.
- Indeed the short and the long. Marry, 'tis a noble Lepidus.
- They say all lovers swear more performance than they are wont to keep obliged faith unforfeited!