Notes on Detection[*]

EE 480F

February 1, 2005

1 Classical Cryptography

1.1 Some basic ciphers

Since we will use detection to crack codes, let's start with a brief list of classical ciphers that we will be breaking.

- **Caesar cipher** consists of shifting each letter three positions on the alphabet. `HELLO` becomes `KHOOR`.
- The **shift cipher** consists of shifting each letter by a constant but unknown shift. $\texttt{Encrypt}(\texttt{HELLO}, 2)$ becomes `JGNNQ`. There are 26 keys. Sometimes we represent each key with a letter: $\texttt{Encrypt}(\texttt{HELLO}, \texttt{B}) = \texttt{Encrypt}(\texttt{HELLO}, 2)$
- The cipher described in **art #45 of the Kāma-Sūtra** consists of pairing off the alphabet, and swapping each letter with its pair. There are many keys, $26!/13!2^{13} = 7905853580625$.
- The general **substitution cipher** maps each letter with another, possibly itself. There are $26! = 403291461126605635584000000 \approx 2^{88}$ keys.

These are *monoalphabetic* ciphers: each input letter is mapped to one output letter. A *polyalphabetic* cipher can map the same input letter to different outputs. Examples:

[*]Section 2 is based on the second chapter of *An Introduction to Signal Detection and Estimation* by H. Vincent Poor. This is only a smidgen of that book, which you should buy and guard jealously if you are a graduate student in EE.

- The **Vigenère cipher** is a shift cipher combined with a keyphrase rather than a single shift. If the keyword is `DOG`, then

$$\begin{aligned}
\texttt{L} &= \texttt{Encrypt}(\texttt{H},\texttt{D})\\
\texttt{T} &= \texttt{Encrypt}(\texttt{E},\texttt{O})\\
\texttt{S} &= \texttt{Encrypt}(\texttt{L},\texttt{G})\\
\texttt{P} &= \texttt{Encrypt}(\texttt{L},\texttt{D})\\
\texttt{D} &= \texttt{Encrypt}(\texttt{O},\texttt{O})\\
\texttt{E} &= \texttt{Encrypt}(\texttt{W},\texttt{G})\\
\texttt{S} &= \texttt{Encrypt}(\texttt{O},\texttt{D})\\
\texttt{G} &= \texttt{Encrypt}(\texttt{R},\texttt{O})\\
\texttt{S} &= \texttt{Encrypt}(\texttt{L},\texttt{G})\\
&\ldots
\end{aligned}$$

- The **Enigma machine** of Arthur Scherbius is an extreme example, but a polyalphabetic cipher nonetheless. At any time, the machine partitions the alphabet into pairs, and maps each letter to its pair. However, the machine state changes every time a letter is entered.

1.2 Basic frequency analysis

For a monoalphabetic cipher, the proportions of letters will match the distribution of the language in which it is written. Hence a histogram of the cipher symbols will leak all sorts of information about the key.

Here is some notation we will use for the rest of the class:

For a string s:

- $N_s[\texttt{a}]$ is the number of times the letter `a` appears in s. N_s is simply the frequency histogram of the string.

- $T_s[\texttt{a}]$ is the *fraction* of times the letter `a` appears in s: $T_s[\texttt{a}] = \frac{1}{|s|}N_s[\texttt{a}]$. This is just a normalized histogram, and can be treated like a probability distribution. The "T" stands for "type," and this is sometimes called the *type of s*, or the *type distribution of s*.

The **Index of Coincidence** of a string, written $IC(s)$, is the probability that two randomly chosen letters in the string are equal. Using the above notation, $IC(s) = \sum_a (T_s[a])^2$, the sum of squares of the type of s.[1]

Facts about $IC(s)$:

- $IC(s) \leq 1$.
- $IC(s) = 1$ when a string contains only one kind of symbol.
- $IC(s) \geq 1/|\mathcal{A}|$, where $|\mathcal{A}|$ is the size of the alphabet.
- $IC(s) = 1/|\mathcal{A}|$, when a string contains every letter in equal proportion.

The index of coincidence is used as a test for natural language. If ciphertext has the same distribution as english, its IC will be abnormally large. Note: we don't use $IC(s)$ today, or we shouldn't, because it is suboptimal.

1.3 How to break Vigenère

To break ciphertext encrypted with Vigenère, you must first determine the key length. There are two ways to do this:

> *Attack 1*: Guess the length of the key phrase k, divide the ciphertext into $n = |k|$ parts, each representing text from a simple shift cipher. Use a test for natural language on each part. The value will be high when the key length is properly guessed.
>
> *Attack 2*: Look for strings in the ciphertext which appear more tthan once. With good probability, the distance between these strings will be a multiple of the keyphrase length.

After the length is known, the text can be partitioned into pieces, and each piece broken by monoalphabetic frequency analysis. More in the section on detection, below.

[1] We are assuming we might pick the same letter twice. Many textbooks give a different definition of IC(s), assuming that the two randomly chosen letters are in different places. It really doesn't matter in practice, so we use the simple definition.

2 Detection

2.1 Detecting English

Imagine we have a piece of ciphertext c that has been encrypted with one of several million keys. A computer could simply decrypt c with every possible key—short work for a computer—until it finds the right one. But how does the computer know when it guesses the right key?

Presumably, under the wrong key, $\texttt{Decrypt}(c, k_{\text{wrong}})$ looks like gibberish, whereas $\texttt{Decrypt}(c, k_{\text{right}})$ possesses the statistics of English. How does the computer test for the statistics of English? A better question might be, what is the *best* algorithm for doing so?

2.2 Elementary Hypothesis Testing

We will be describing an hypothesis test $\delta()$, which inputs a string and outputs a 0 or a 1. $\delta(s) = 1$ if the algorithm detects signs of English.

The algorithm will occasionally make mistakes. After all, random letters will form English words with some tiny probability; and some English phrases look very much like gibberish.[2]

Given an hypothesis test $\delta()$:

- The **false alarm probability** P_F is the probability that δ incorrectly outputs 1.
- The **miss probability** P_M is the probability that δ incorrectly outputs 0 — that δ misses the plaintext.
- The **detection probability** $P_D = 1 - P_M$ is the probability that δ correctly outputs 1, detecting the plaintext.

Note a potential misconception: P_F is not the rate of false alarms in general. It is the probability that $\delta(s) = 1$ given that s is noise. In other words, if you only ever input noise to the detector, about P_F of the detections will be wrong. Likewise, if you only ever input English into the detector, About P_M of the detections will be wrong.

In contrast, if we feed any old input to the detector and count the results, the rate of false alarms is not necessarily P_F. Ultimately, the rate of misses

[2]Claude Shannon discovered the sentence, "Squdgy fez, blank jimp crwth vox," which contains each letter of the alphabet exactly once. Each word is a real word in the English dictionary, although the sentence is somewhat nonsensical.

and false alarms depends on how often the input is noise versus English text. If the input is always text, then a false alarm can never happen!

So, to press the point just a bit more: these quantities P_F and P_M are properties of the algorithm $\delta()$. They do not account for what the user types in.

Now, let us generalize a bit, to arbitrary decision making. Our algorithm is trying to decide between the two following hypotheses, involving a random observation:

H0 The observation Y is distributed according to probability distribution $P_0[y]$

H1 The observation Y is distributed according to probability distribution $P_1[y]$

For example, you might have a fair coin with $\Pr[HEADS] = 1/2$ or a biased coin with $\Pr[HEADS] = 1/3$. After how many coin flips should you decide which is the case? How many coin flips will you want before making a decision? Is the string `HTTHTHTTHTHHHHTHTHHHHTH` a sign that the coin is fair or biased?

2.3 What exactly do we want?

In the end, we want an algorithm that does a "good job" of detecting English amid the noise of cryptotext. But to accomplish this, we must define quantitatively what we mean by "good."

There are several different approaches to defining the performance of an hypothesis test. Here are three common approaches:

- The Bayesian approach: mistakes incur some kind of penalty. Maybe we face different penalties for different kind of mistakes (*e.g.*, a false alarm isn't so bad compared to a miss.) Likewise, we are rewarded for success. Each kind of success and failure has a probability as well as a cost, so we will try to *minimize the average cost.* This will be the approach we use.

- The Minimax approach: this is like the Bayesian approach, except we don't know exactly the probability of success or failure. To correct this we consider the worst case (the "max" in "minimax") and try to minimize the cost in the worst case (the "min" in "minimax.")

- The Neyman-Pearson approach: suppose you must keep the false alarms below an absolute limit. So we fix that $P_F < \alpha$ for some value $\alpha > 0$, and try to maximize P_D subject to that constraint.

You can probably see the motivation behind these three ideas. The first approach makes sense in a lot of engineering applications, in which what really matters is overall cost, and we have all the information about what we are testing. The third approach makes sense in a court of law or other adversarial situations; it also makes sense when we can't realistically assign relative costs to false positives and false negatives.

Thankfully, all three of these approaches lead us to the same basic kind of statistical test.

2.4 How to read a ROC (receiver operating characteristic)

Suppose we have settled on an algorithm to separate English from gibberish. We now want a way to assess how good it is. Maybe it's the best possible algorithm, but even the best is pretty worthless. Maybe its performance can be tweaked to make it more cautious or more sensitive. A ROC diagram allows us to visualize its performance.

A ROC is simply a plot of the false alarm and detection probabilities of an hypothesis test (or detector). The x-axis shows P_F, and the y-axis shows $P_D = 1 - P_M$.

Many detector algorithms have a variable sensitivity or threshold, so there is no single value for P_D or P_F. Rather, the plot often takes the form of a curve, each point corresponding to one value of the detector threshold.

Notice that ROC curves stretch from the point (0,0) to (1,1). This is because with a suitable threshold, the detector will always declare a miss or always declare a hit. Also note that the curves fall above the line $y = x$. They had better: if you imagine a "detector" that ignores the input and simply guesses an answer, its performance will fall along the line $y = x$, because $P_F = P_D$. A detector that falls below the line would make no sense; you could turn that into a better detector by reversing the outputs.

How do we use a ROC to decide the best parameter value for our test? In other words, what point on the ROC curve is "best?" If we are using a Bayesian approach, any pair (P_F, P_D) gives us an overall cost. For a fixed cost c, the set of all points (P_F, P_D) with that cost will form a straight line L_c, of fixed slope. All the cost lines are parallel, for reasons we discuss in detail below. So, the "best" point on the ROC curve is one tangent to a cost line.

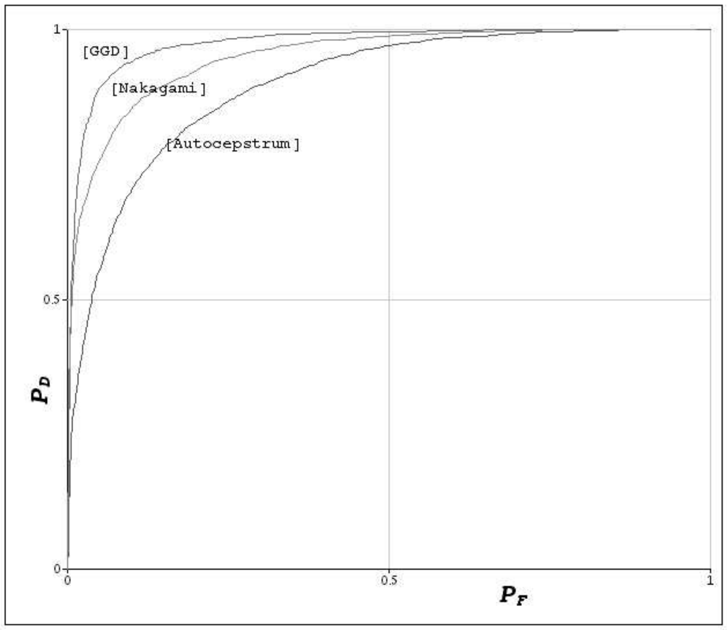

Figure 1: A ROC diagram, showing curves for 3 different detectors. The one labeled "GGD" gives the best performance.

Using the Neyman-Pearson approach, we draw a vertical line for a fixed P_F value. We use a point where this line intersects the curve.

2.5 The optimal hypothesis test

Recall that we have two hypotheses about a random observation Y, either that $Y \sim P_0$ or $Y \sim P_1$ (the $\sim$ symbol means, "is distributed according to.") We also have several other parameters that factor into our decision:

- The **costs** of guessing: the number C_{ij} is the cost of guessing hypothesis i, when in fact hypothesis j is the right one. Usually $C_{ii} = 0$ (no cost for guessing right,) and in many cases $C_{ij} = 1, i \neq j$ – a situation called *uniform costs.*

  Sometimes it will be handy to set the costs differently, if a false alarm is worse than a miss.

- The **a priori probabilities**, or priors, are the probabilities of the two hypotheses before any data is observed. These are denoted π_0 and $\pi_1 = 1 - \pi_0$: the probabilities that $H0$ and $H1$ are correct, respectively.

Now let's derive the "best" algorithm $\delta(y)$ for deciding from y which hypothesis is true. For the set of all observations $\mathcal{Y}$, we can denote Γ_1 the set of all inputs that cause $\delta(y)$ to output 1. Γ_0 is defined similarly, and $\Gamma_0 \cup \Gamma_1 = \mathcal{Y}$.[3] We can see that $P_F = \Pr_0[\Gamma_1]$ and $P_D = \Pr_1[\Gamma_1]$. $P_M = \Pr_1[\Gamma_0] = 1 - \Pr_1[\Gamma_1]$. Our average cost is

$$\texttt{cost}(\delta) = \pi_0[C_{10} \Pr{}_0[\Gamma_1] + C_{00}(1 - \Pr{}_0[\Gamma_1])] + \pi_1[C_{11} \Pr{}_1[\Gamma_1] + C_{01}(1 - \Pr{}_1[\Gamma_1])]$$

This is just the sum $\sum_{ij} C_{ij} \Pr[C_{ij}]$. Rearranging tells us how to devise an optimal algorithm:

$$\texttt{cost}(\delta) = \pi_0 \Pr{}_0[\Gamma_1](C_{10} - C_{00}) + \pi_1 \Pr{}_1[\Gamma_1](C_{11} - C_{01}) + [\pi_0 C_{00} + \pi_1 C_{01}]$$

The last part in brackets is the same regardless of our choice for $\delta(y)$. To minimize the first part, we want to minimize

$$\sum_{y \in \Gamma_1} \pi_0 \Pr{}_0[y](C_{10} - C_{00}) + \pi_1 \Pr{}_1[y](C_{11} - C_{01})$$

So it makes sense to choose Γ_1 to consist of those particular values of y for which $\pi_0 \Pr{}_0[\Gamma_1](C_{10} - C_{00}) + \pi_1 \Pr{}_1[\Gamma_1](C_{11} - C_{01})$ is negative:

$$\begin{aligned}
\Gamma_1 &= \{y : \pi_0 \Pr{}_0[y](C_{10} - C_{00}) + \pi_1 \Pr{}_1[y](C_{11} - C_{01}) < 0\} \\
&= \left\{ y : \frac{\Pr{}_1[y]}{\Pr{}_0[y]} > \frac{\pi_0(C_{10} - C_{00})}{\pi_1(C_{01} - C_{11})} \right\}
\end{aligned}$$

So, this is our algorithm: given an observation y, output 1 if $\Pr_1[y]/\Pr_0[y] > \tau$, where our threshold $\tau = \frac{\pi_0(C_{10} - C_{00})}{\pi_1(C_{01} - C_{11})}$.

[3] This is not always true. There are some algorithms which will make a random guess for some inputs y. We will ignore such randomized decision rules right now.