

Name: _____

CS573 Midterm: Fall 2010

This is a closed-book, closed-notes exam. Non-programmable calculators are allowed for probability calculations.

There are 11 pages including the cover page. The total number of points for the exam is 50 and you have 2 hours to complete the exam. Note the point value of each question and allocate your time accordingly. Read each question carefully and show your work.

Question	Score
1	
2	
3	
4	
5	
6	
7	
8	
Total	

1 Data mining components (8 pts)

Read the excerpts from Kolter, J. Z. and Maloof, M. A. *Learning to detect malicious executables in the wild*. In Proceedings of the Tenth ACM SIGKDD international Conference on Knowledge Discovery and Data Mining, 2004 on pages 9-11.

1. Describe the data mining task.
Classification
Detecting malicious executables.
2. Describe the data representation.
IID, tabular data, where each instance is a tuple of features
Features are n-grams
3. Describe the knowledge representation.
Nearest neighbor formulation for lazy prediction
500 n-grams with highest information gain
4. Describe the learning algorithm (both search and scoring function).
Similarity coefficient is used for scoring internally
Search is implicit in selection of nearest neighbors

2 Models and patterns (6pts)

1. Describe the difference between a model and a pattern. Give an example of each.
Pattern describes local property of data.
Model describes global property of data.
pattern example (e.g., association rule)
model example (e.g., decision trees)
2. Discuss the difference between parametric and non-parametric models. Give an example model of each type and describe a situation in which it may outperform the other type of model.
Parametric models assume a particular parametric form (e.g., Gaussian) and learning consists of optimizing those parameters.
Non-parametric models make fewer assumptions about the function form of the model and are data-driven (e.g., size of the model can grow without bound as the size of the training data increases).
Parametric example (e.g., Naive Bayes)
Non-parametric example (e.g., decision trees)
Situation: parametric may do better with less data because parametric assumptions reduce variance of model.

3 Sufficient Statistics (4pts)

Suppose s is a statistic for which $p(\theta|\mathbf{x}, D) = p(\theta|s)$. Assume $p(\theta|s) \neq 0$ and prove that $p(D|s, \theta)$ is independent of θ . Create an example to show that the inequality $p(\theta|s) \neq 0$ is required for your proof.

By Bayes theorem we have:

$$P(D|s, \theta) = \frac{P(\theta|s, D)P(s, D)}{P(s, \theta)}$$

Since $P(\theta|s, D) = P(\theta|s)$:

$$\begin{aligned} P(D|s, \theta) &= \frac{P(\theta|s)P(s, D)}{P(s, \theta)} \\ &= \frac{P(\theta|s)P(s)P(D|s)}{P(s, \theta)} \\ &= \frac{P(\theta, s)P(D|s)}{P(s, \theta)} \\ &= P(D|s) \end{aligned}$$

Thus $P(D|s, \theta)$ is independent of θ .

The inequality $p(\theta|s) \neq 0$ is required. Otherwise, $P(\theta, s) = 0$ and the division operation is undefined.

4 Central Limit Theorem (6pts)

In a given city it is assumed that the average number of automobile accidents in each year is 15 with variance 3.

1. Explain why the distribution of number of automobile accidents cannot be normally distributed.

If the response time X is normally distributed, the probability of X being negative is non-zero, i.e., $P(X < 0) > 0$. This is unrealistic. Therefore, the number of accidents cannot be normally distributed.

2. The government has recorded the average number of accidents over the past 20 years ($\bar{X} = \frac{1}{20} \sum_{i=1}^{20} X_i$). What are the mean and variance of \bar{X} ?

$$\begin{aligned} E(\bar{X}) &= E\left(\frac{1}{20} \sum_{i=1}^{20} X_i\right) \\ &= \frac{1}{20} \sum_{i=1}^{20} E(X_i) \\ &= \frac{1}{20} \cdot 20 \cdot 15 = 15 \end{aligned}$$

$$\begin{aligned} Var(\bar{X}) &= Var\left(\frac{1}{20} \sum_{i=1}^{20} X_i\right) \\ &= \left(\frac{1}{20}\right)^2 \sum_{i=1}^{20} Var(X_i) \\ &= \frac{1}{20 \cdot 20} \cdot 20 \cdot 3 = 0.15 \end{aligned}$$

3. What does the central limit theorem tell us about the distribution of \bar{X} ?

The central limit theorem tells us that the distribution of \bar{X} can be approximated by a normal distribution with mean 15 and variance 0.15.

5 Hypothesis Testing (8pts)

Suppose a box contains four marbles, θ white ones and $4 - \theta$ black ones. Test the hypothesis $H_0 : \theta = 2$ against the hypothesis $H_1 : \theta = 3$ as follows. Draw two marbles with replacement and reject H_0 if both marbles are the same color; otherwise do not reject.

1. What is the probability of Type I error?

$$\begin{aligned}P(\text{Type I error}) &= P(\text{reject } H_0 | H_0 \text{ true}) \\&= P(2 \text{ balls have same color} | \theta = 2) \\&= \frac{2}{4} \cdot \frac{2}{4} + \frac{2}{4} \cdot \frac{2}{4} \\&= \frac{1}{2}\end{aligned}$$

2. What is the probability of Type II error?

$$\begin{aligned}P(\text{Type II error}) &= P(\text{fail to reject } H_0 | H_1 \text{ true}) \\&= P(2 \text{ balls are different color} | \theta = 3) \\&= \frac{1}{4} \cdot \frac{3}{4} + \frac{3}{4} \cdot \frac{1}{4} \\&= \frac{3}{8}\end{aligned}$$

3. What is the power of the test?

$$\begin{aligned}\text{Power} &= 1 - \text{Type II error} \\&= 1 - \frac{3}{8} \\&= \frac{5}{8}\end{aligned}$$

4. Name one way to increase the power of the test.

State a method in generality (e.g increase the critical threshold or α)

Connect it to the example (e.g., increase draws of marbles and/or lower decision threshold on resulting draws)

6 Data Transformation and Distance (6pts)

Consider a document-term matrix, where tf_{ij} is the frequency of the i^{th} word (term) in the j^{th} document and m is the number of documents. Consider the variable transformation that is defined by:

$$tf'_{ij} = tf_{ij} \cdot \log \frac{m}{df_i}$$

where df_i is the number of documents in which the i^{th} term appears and is known as the document frequency of the term. This transformation is known as the inverse document frequency transformation.

1. What is the effect of this transformation if a term occurs in one document? In every document?
If a term appears in just one document, then it is transformed by the highest weight (i.e., $\log m$).
If a term appear in all documents then it gets zeroed out.
2. What might be the purpose of this transformation?
The transformation of word counts serves as an indication of document relevance.
It upweights rare terms and downweights frequent terms.
3. Cosine distance is a common measure of document similarity and is defined as:

$$\cos(x, y) = \frac{x \cdot y}{\|x\| \|y\|}$$

where x and y are two document vectors, $x \cdot y$ is the vector dot product and $\|x\|$ is the length of the vector x . Is cosine distance a metric? Explain why or why not.

The cosine distance is not a metric.

It fails to satisfy the identity requirement: $d(x, x) = \frac{x \cdot x}{\|x\| \|x\|} = \frac{\|x\|^2}{\|x\|^2} = 1$.

7 Decision Tree Split Criteria (4pts)

Entropy is a natural measure to quantify the impurity of a data set. Recall that entropy is defined as $H(X) = -\sum_x p(x)\log_2 p(x)$. The Decision Tree learning algorithm uses entropy as a splitting criterion by calculating the information gain to decide the next attribute to partition the current node. However, there are other impurity measures that could be used as the splitting criteria too. Assume the current node n has k classes c_1, c_2, \dots, c_k .

$$\text{Misclassification Impurity} : i(n) = 1 - \max_{i=1}^k P(c_i)$$

1. Assume node n has two classes, c_1 and c_2 . Draw a figure in which the impurity measures of Entropy and Misclassification are represented as a function of $P(c_1)$.

Figure setup with x-axis ($P(C = c_1)$) from 0 to 1 and y-axis (E/M) from 0 to 1

Correct lines:

$$\text{Entropy}(P(c_1 = 0)) = \text{Entropy}(P(c_1 = 1)) = 0; \text{Entropy}(P(c_1 = .5)) = 1;$$

$$\text{Entropy}(P(c_1 = .75)) = 0.81$$

$$\text{Misclassification}(P(c_1 = 0)) = \text{Misclassification}(P(c_1 = 1)) = 0;$$

$$\text{Misclassification}(P(c_1 = .5)) = .5; \text{Misclassification}(P(c_1 = .75)) = 0.25$$

2. Now we can define a new split criteria based on the Misclassification Impurity, which is called *Drop-of-Impurity* in some literatures. That is the difference between the impurity of the current node and the impurities of children. For binary category splits, *Drop-of-Impurity* is defined as:

$$\Delta i(n) = i(n) - P(n_l)i(n_l) - P(n_r)i(n_r)$$

where n_l and n_r are the left and right child of node n after splitting. Calculate the *Drop-of-Impurity* for the following example data set in which y is the class variable to be predicted.

Y	X
+	1
+	1
-	1
-	0
-	0
-	0

$$i(n) = 1 - \frac{4}{6} = \frac{1}{3}$$

$$i(n_l) = 1 - \frac{2}{3} = \frac{1}{3}$$

$$i(n_r) = 1 - 1 = 0$$

$$\Delta i(n) = \frac{1}{3} - \frac{1}{2} \cdot \frac{1}{3} - \frac{1}{2} \cdot 0 = \frac{1}{6}$$

8 Decision Tree Growth (8pts)

The Decision Tree growing algorithm we have looked at so far is a greedy, heuristic search with one-step look-ahead and no backtracking. Lets consider how this would improve if we were to look-ahead two steps.

Originally, we determined which attribute to select for splitting a node by comparing the information gain of each of the attributes from X_1 to X_p (assume we have p discrete binary attributes). Now, for each attribute we will also consider the best way to split the resulting children. We will try each remaining attribute for both the left and right child separately, and then pick the pair (X_l, X_r) which maximizes:

$$IG(S|X) + \frac{N_l}{N_l + N_r} IG(S_l|X_l) + \frac{N_r}{N_l + N_r} IG(S_r|X_r)$$

where N_l and N_r are the number of data points in the left and right children of the node which is considered how to split, and S_l and S_r are those two sets of data points themselves. We will call this $IG_2(S|X)$, the two-step look-ahead information gain for attribute X on data set S .

1. Briefly explain why we need the factors $\frac{N_l}{N_l+N_r}$ and $\frac{N_r}{N_l+N_r}$.

Normalization accounts for amount of data that is considered in each IG term

Penalizes against splitting off small portion of data with large IG

2. Assuming there are p discrete, binary attributes, how many information gain computations must be carried out to determine how to split a single node?

Consider number of attributes at each of the two levels

p attributes at first level, then $p - 1$ for left child and $p - 1$ for right child. Note: that left and right can be chosen independently

Total number of tests is $p([p - 1] + [p - 1]) = 2p^2 - 2p = O(p^2)$

3. Is this a larger model space? That is, does this change to the decision tree algorithm allow us to represent new concepts that were not previously representable with the standard greedy decision trees? Briefly explain why or why not.

No.

The number of possible trees that can be represented remains the same.

However, the new search method may explore larger portion of the DT space and return a better model.

Excerpts from Kolter, J. Z. and Maloof, M. A. *Learning to detect malicious executables in the wild*. In Proceedings of the Tenth ACM SIGKDD international Conference on Knowledge Discovery and Data Mining, 2004.

Malicious code is “any code added, changed, or removed from a software system to intentionally cause harm or subvert the system’s intended function” [27, p. 33]. Such software has been used to compromise computer systems, to destroy their information, and to render them useless. Malicious executables generally fall into three categories based on their transport mechanism: viruses, worms, and Trojan horses. Viruses inject malicious code into existing programs, which become “infected” and, in turn, propagate the virus to other programs when executed. Viruses come in two forms, either as an infected executable or as a virus loader, a small program that only inserts viral code. Worms, in contrast, are self-contained programs that spread over a network, usually by exploiting vulnerabilities in the software running on the networked computers. Finally, Trojan horses masquerade as benign programs, but perform malicious functions. Malicious executables do not always fit neatly into these categories and can exhibit combinations of behaviors.

Excellent technology exists for detecting known malicious executables. Software for virus detection has been quite successful. These programs search executable code for known patterns, and this method is problematic. One shortcoming is that we must obtain a copy of a malicious program before extracting the pattern necessary for its detection. Obtaining copies of new or unknown malicious programs usually entails them infecting or attacking a computer system. To complicate matters, writing malicious programs has become easier: There are virus kits freely available on the Internet. Individuals who write viruses have become more sophisticated, often using mechanisms to change or obfuscate their code to produce so-called polymorphic viruses [3, p. 339]. Indeed, researchers have recently discovered that simple obfuscation techniques foil commercial programs for virus detection [7]. These challenges have prompted some researchers to investigate learning methods for detecting new or unknown viruses, and more generally, malicious code.

Our Malicious Executable Classification System (mecs) currently detects unknown malicious executables “in the wild”, that is, without removing any obfuscation. To date, we have gathered 1971 system and non-system executables, which we will refer to as “benign” executables, and 1651 malicious executables with a variety of transport mechanisms and payloads (e.g., key-loggers and backdoors). Although all were for the Windows operating system, it is important to note that our approach is not restricted to this operating system. We extracted byte sequences from the executables, converted these into n-grams, and constructed several classifiers. In this domain, there is an issue of unequal but unknown costs of misclassification error, so we evaluated the methods using receiver operating characteristic (roc) analysis [40], using area under the roc curve as the performance metric. Ultimately, boosted decision trees outperformed all other methods with an area under the curve of 0.996. ...

As stated previously, the data for our study consisted of 1971 benign executables and 1651 malicious executables. All were in the Windows pe format. We obtained benign executables from all folders of machines running the Windows 2000 and xp operating systems. We gathered additional applications from SourceForge (<http://sourceforge.net>). We obtained viruses, worms, and Trojan horses from the Web

site VX Heavens (<http://vx.netlux.org>) and from computer-forensic experts at the mitre Corporation, the sponsors of this project. Some executables were obfuscated with compression, encryption, or both; some were not, but we were not informed which were and which were not. For one collection, a commercial product for detecting viruses failed to identify 18 of the 114 malicious executables. Note that for viruses, we examined only the loader programs; we did not include infected executables in our study. We used the hexdump utility [29] to convert each executable to hexadecimal codes in an ascii format. We then produced n-grams, by combining each four-byte sequence into a single term. For instance, for the byte sequence ff 00 ab 3e 12 b3, the corresponding n-grams would be ff00ab3e, 00ab3e12, and ab3e12b3. This processing resulted in 255,904,403 distinct n-grams. One could also compute n-grams from words, something we explored and discuss further in Section 6.1. Using the n-grams from all of the executables, we applied techniques from information retrieval and text classification, which we discuss further in the next section. ...

Our overall approach drew techniques from information retrieval (e.g., [16]) and from text classification (e.g., [12, 36]). We used the n-grams extracted from the executables to form training examples by viewing each n-gram as a binary attribute that is either present in (i.e., 1) or absent from (i.e., 0) the executable. We selected the most relevant attributes (i.e., n-grams) by computing the information gain (IG) for each:

$$IG(j) = \sum_{v_j \in \{0,1\}} \sum_{C \in \{C_i\}} P(v_j, C) \log \frac{P(v_j, C)}{P(v_j)P(C)},$$

where C is the class, v_j is the value of the j^{th} attribute, $P(v_j, C)$ is the proportion that the j^{th} attribute has the value v_j in the class C_i , $P(v_j)$ is the proportion that the j^{th} n-gram takes the value v_j in the training data, $P(C)$ is the proportion of the training data belonging to the class C . This measure is also called average mutual information [43]. We then selected the top 500 n-grams, a quantity we determined through pilot studies (see Section 6.1), and applied several learning methods, most of which are implemented in weka [42] ...

For the tfidf classifier, we followed a classical approach from information retrieval [16]. We used the vector space model, which entails assigning to each executable (i.e., document) a vector of size equal to the total number of distinct n-grams (i.e., terms) in the collection. The components of each vector were weights of the top n-grams present in the executable. For the j^{th} n-gram of the i^{th} executable, the method computes the weight w_{ij} , defined as $w_{ij} = \frac{tf_{ij}}{idf_j}$ where tf_{ij} (i.e., term frequency) is the number of times the i^{th} n-gram appears in the j^{th} executable and $idf_j = \log \frac{d}{df_j}$ (i.e., the inverse document frequency), where d is the total number of executables and df_j is the number of executables that contain the j^{th} n-gram. It is important to note that this classifier was the only one that used continuous attribute values; all others used binary attribute values. To classify an unknown instance, the method uses the top n-grams from the executable, as described previously, to form a vector, \vec{u} , the components of which are each n-gram's inverse document frequency (i.e., $u_j = idf_j$). Once formed, the classifier computes a similarity coefficient (SC) between the vector for the unknown executable and each vector for the executables in the collection using the cosine similarity measure:

$$SC(\vec{u}, \vec{w}_i) = \frac{\sum_{j=1}^k u_j w_{ij}}{\sqrt{\sum_{j=1}^k u_j^2 \cdot \sum_{j=1}^k w_{ij}^2}},$$

where \vec{u} is the vector for the unknown executable, \vec{w}_i is the vector for the i^{th} executable, and k is the number of distinct n-grams in the collection. After selecting the top five closest matches to the unknown, the method takes a weighted majority vote of the executable labels, and returns the class with the least weight as the prediction. It uses the cosine measure as the weight. Since we evaluated the methods using roc analysis [40], which requires case ratings, we summed the cosine measures of the negative executables in the top five, subtracted the sum of the cosine measures of the positive executables, and used the resulting value as the rating. In the following discussion, we will refer to this method as the tfidf classifier...

We conducted three experimental studies using our data collection and experimental methodology, described previously. We first conducted pilot studies to determine the size of words and n-grams, and the number of n-grams relevant for prediction. Once determined, we applied all of the classification methods to a small collection of executables. We then applied the methodology to a larger collection of executables, all of which we describe in the next three sections.

With success on a small collection, we turned our attention to evaluating the text-classification methods on a larger collection of executables. As mentioned previously, this collection consisted of 1971 benign executables and 1651 malicious executables, while processing resulted in over 255 million distinct n-grams of size four. We followed the same experimental methodology—selecting the 500 top n-grams for each run of ten-fold cross-validation, applying the classification methods, and plotting roc curves. Figure 2 shows the roc curves for the various methods, while Table 3 presents the areas under these curves (auc) with 95% confidence intervals. As one can see, boosted j48 outperformed all other methods. Other methods, such as ibk and boosted svms, performed comparably, but the roc curve for boosted j48 dominated all others.