PROBABILISTIC PASSWORD MODELING: PREDICTING PASSWORDS WITH MACHINE LEARNING

JAY DESTORIES
MENTOR: ELIF YAMANGIL

Abstract

Many systems use passwords as the primary means of authentication. As the length of a password grows, the search space of possible passwords grows exponentially. Despite this, people often fail to create unpredictable passwords. This paper will explore the problem of creating a probabilistic model for describing the distribution of passwords among the set of strings. This will help us gain insight into the relative strength of passwords as well as alternatives to existing methods of password candidate generation for password recovery tools like John the Ripper. This paper will consider methods from the field of natural language processing and evaluate their efficacy in modeling human-generated passwords.

Contents

1. Introduction

2. Classifying the Language of Passwords

2.1. **Regular Languages.**

2.2. **Context-Free Languages.**

2.3. **Context-Sensitive and Recursively Enumerable Languages.**

3. Probability Modeling

3.1. **N-Gram Language Model.**

3.2. **Hidden Markov Model.**

3.3. **Context-Free Grammar.**

4. Generating Passwords

4.1. **Baseline.**

4.2. **N-Gram Language Model.**

4.3. **Hidden Markov Model.**

4.4. **Context-Free Grammar.**

5. Applications

5.1. **Password Strength.**

5.2. **Password Recovery.**

6. Conclusion

References

[1] Ghahramani, Z. (2001) An Introduction to Hidden Markov Models and Bayesian Networks. *International Journal of Pattern Recognition and Artificial Intelligence.*

[2] Rosenfeld, Ronald. Two Decades of Statistical Language Modeling: Where do we go from here? www.cs.cmu.edu/˜roni/papers/survey-slm-IEEE-PROC-0004.pdf

7. Appendix

7.1. Language Model Evaluation.

```
"""
    Test the performance of various password models
"""

# basic wordlist attack
def baselineGenerator(training_corpus):
    for pwd in training_corpus:
        yield pwd
    while True:
        yield ""

# See how many things in test_corpus the generator can guess with some number of
# tries
def testGenerator(gen, test_corpus, tries):
    found = 0
    test_set = set(test_corpus)
    guesses = set()
    for i in xrange(tries):
        guess = gen.next()
        if not guess in guesses:
            guesses.update([guess])
            if guess in test_set:
                found += 1
    return found

def testCopora(training_corpus, test_corpus):
    print "First 5 training passwords: ", training_corpus[:5]
    print "First 5 test passwords: ", test_corpus[:5]

    tries = 1000000
    baseline = testGenerator(baselineGenerator(training_corpus), test_corpus, tries)
    print "Baseline wordlist attack -- %d tries: %d." % (tries, baseline)


def main():
    print "############################################################################"
    print "Training corpus: rockyou"
    print "Test corpus: gmail"
    print "############################################################################"
    rockyou_nocount = open('corpora/rockyou_nocount', 'r')
    training_corpus = [pwd.rstrip() for pwd in rockyou_nocount]
    gmail_nocount = open('corpora/gmail_nocount', 'r')
    gmail_corpus = [pwd.rstrip() for pwd in gmail_nocount]
    test_corpus = gmail_corpus[:-5000]
    held_out_corpus = gmail_corpus[-5000:]
    testCorpora(training_corpus, test_corpus)


if __name__ == "__main__":
    main()
```

7.2. N-Gram Language Model Implementation.

7.3. Hidden Markov Model Implementation.

7.4. Context-Free Grammar Implementation.