**Automated Personalizable Backup Software**

Matthew Dill

Bradley Erickson

Jack Williams

CS 445 - Artificial Intelligence

May 3rd, 2019

**Introduction:**

Lots of users want their files backed up; however, some users only want a small subset of desirable files backed up rather than the their entire file system. Our goal is to create and implement an Automated Personalizable Backup Software in a UNIX file system environment. This software will automatically backup files that the program deems desirable and ask users about files it is uncertain about.

There are a lot of standard backup softwares like Acronis True Image free and Replibit. These softwares typically ask the user for a specified directory, an external backup media, and a time interval for how often the software should run backups. Companies are beginning to implement some Machine Learning and Artificial Intelligence algorithms into their backup software. For example, both Commvault and Acronis True Image Pro have implemented some of these algorithms. What are they doing with these algorithms? Don Foster said the following about Commvault. “By analyzing the data you create with every action, every backup, every restore, Commvault software analyzes patterns and performance.” Analyzing patterns allow for a better understanding of what to backup and when to backup any given file.

Acronis also looks at patterns and performance in file manipulation. However, Acronis looks at the various processes accessing the files for possible anomalies or ransomware attacks. Peter Hale describes the action as, “detect[ing] unusual activity, it checks the suspicious process using both heuristics analysis and ML models of expected and unexpected behavior.” When the program observes unexpected behavior, the user will be alerted.

Our program will use some prior knowledge and then begin analyzing the file system. Using a probability, the program will decide what files are desirable for backup. It will backup those files to a specified device. From there, it will ask the user about some files the program is unsure about. After receiving feedback from the user, the probability network will be updated for future use.

**Algorithm Description:**

Here is how our agent learns user preferences and balances the goals of backing up desirable files and conserving storage space.

First, obtain a list of files that we haven’t decided on in the past. (A decision is either KEEP which means always backup this file, or IGNORE which means never backup the file.) Find their size and extension.

Now, the decision element will make decisions on whether to KEEP or not (remain undecided) for these files:

For each undecided file, find the expected utility of KEEPing it. First, find the probability of the file being desirable by multiplying P(Desirable | Size) \* P(Desirable | Extension). The probability Desirable given Size is the number of KEPT files with that value of size divided by the total number of files with that size that we have decided on. The expected utility is the probability of desirable (previously calculated) multiplied by the performance measure’s bonus for keeping a desirable file plus the probability of undesirable \* the performance measure’s deduction for keeping an undesirable file plus the deduction for storage space consumed. The expected utility of not keeping the file is 0. Create a list of all files that have expected utility greater than 0. Order the list descending on expected utility.

Next, backup these files in order. This way, maximum utility is achieved if we run out of storage space. The act of KEEPing / backing up includes moving the file to the storage media location, updating the probability counts for both size and extension that file, and storing that file persistently so we don’t have to decide on it again.

Finally, the Critic can run a training session with the user. We will ask for input on those files we are most undecided about -- files which we calculated probability of desirable closest to .5. For each file query, the user can KEEP the file as described above, or IGNORE the file. IGNORing the file includes updating the probability counts for the size and extension of that file, removing the file from the storage media as necessary, and storing a record of the file persistently so that we don’t have to decide on it again.

**User’s Manual:**

1. Prepare all dependencies required to run the backup bot
   1. Bash available at /usr/bin/bash
   2. rsync available on path
   3. GNU ls and find programs available on path
   4. Python 3 available on path
   5. Eclipse IDE available at [www.eclipse.org](http://www.eclipse.org)
   6. Install Java and JDK version 8 or later
2. In Eclipse, import the BackupBot project as a Maven project
3. Set backup properties in UserConfigConstants.java

Path to directory to be backed up assigned to CRAWLBASEDIR

Path to backup location assigned to STORAGEDIR

Backup storage space assigned to STORAGESIZE (in kilobytes)

1. Add regular expressions of files that should be ignored to ignore.txt
2. Run the program

a) Run Driver.java in the Eclipse IDE

OR

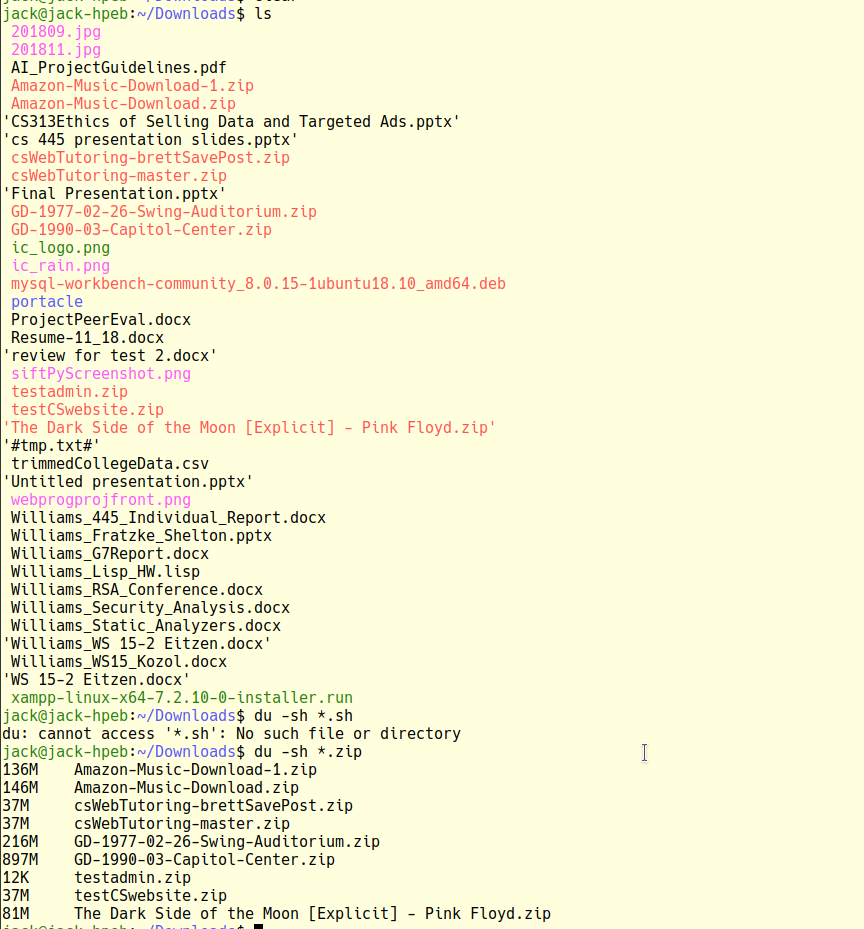
b) Run through the terminal

* + 1. Change directory (cd path/to/project) where the project is located
    2. Run mvn install
    3. Run java -jar target/backup-bot-0.0.1-SNAPSHOT.jar

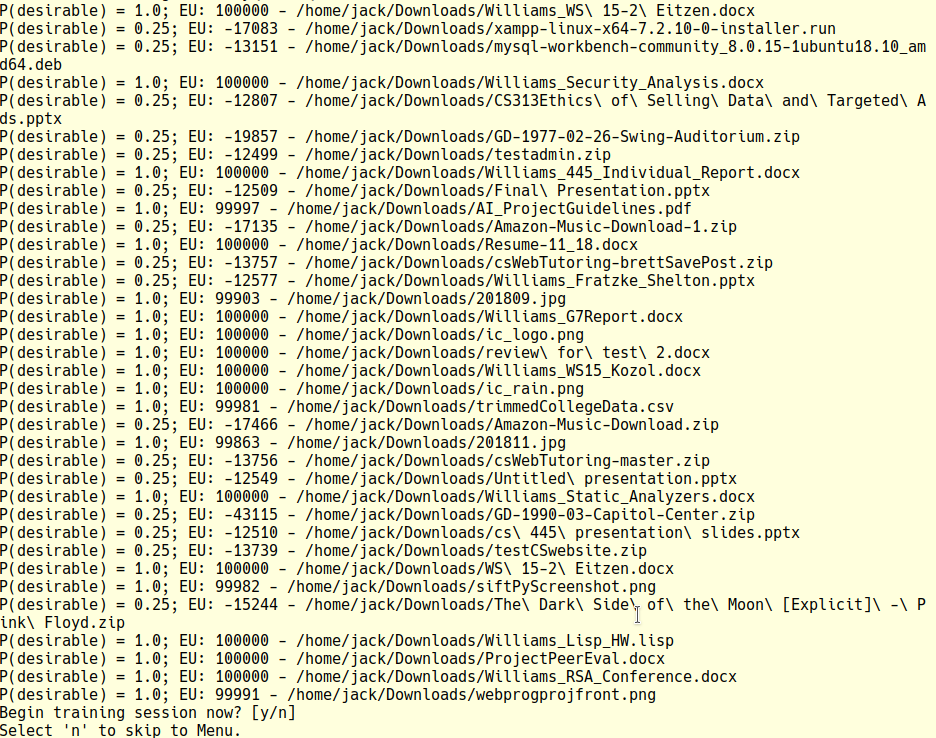
1. After the bot runs through the provided directory, it will prompt you to begin the training session. Enter “y” for yes and “n” for no.
2. The bot’s top 10 most uncertain files will then be presented one by one. If you want to back up that file, enter “y”. Otherwise enter “n” and the file will be ignored.
3. The critic will then present a list of options
   1. [1] to list kept files
   2. [2] to list ignored files
   3. [3] to keep a file
   4. [4] to ignore a file
   5. [5] to run the training session again
   6. [6] to exit the menu
4. If you wish to run the backup bot again with no user preferences, run “rm -rf persistent/\*” in the terminal before running

**Example Problems:**

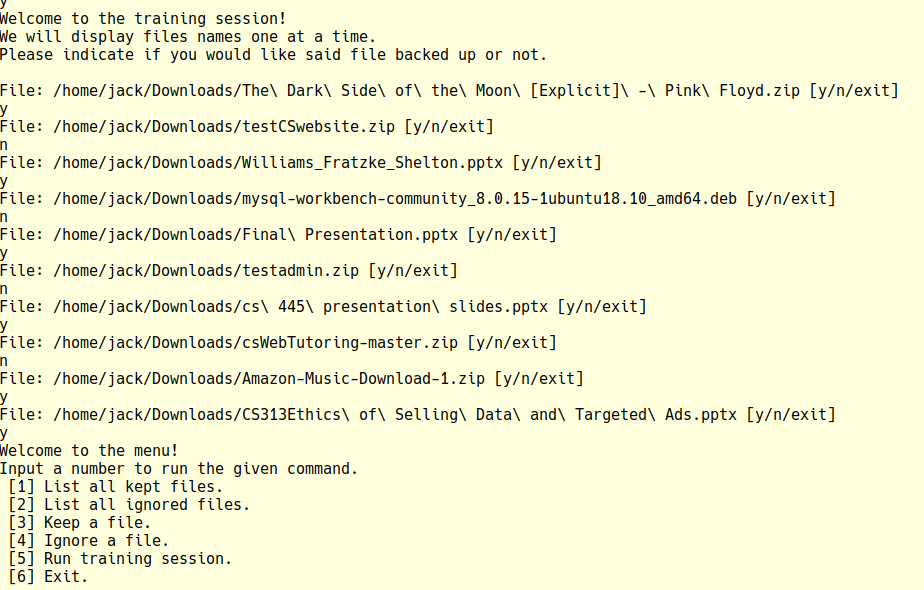
The agent starts with prior knowledge that .docx and image formats are desirable (among other extensions). Other extensions start as undesirable. All sizes are desirable by default. Our goal is to have the agent learn that we don’t desire to backup images, that we want to backup .pptx files, and that we like our .zip archives that have music in them, but not the smaller archives with code.



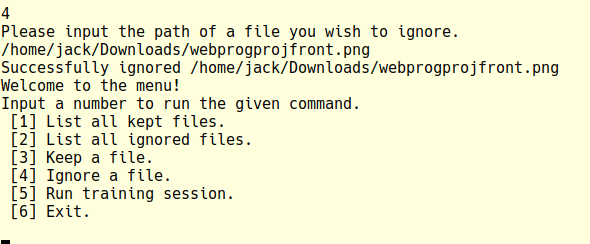
*All files in our base directory. Sizes of zips. We like the zips that are over 50M.*



*Starting probabilities and expected utilities for files in the Downloads directory.*

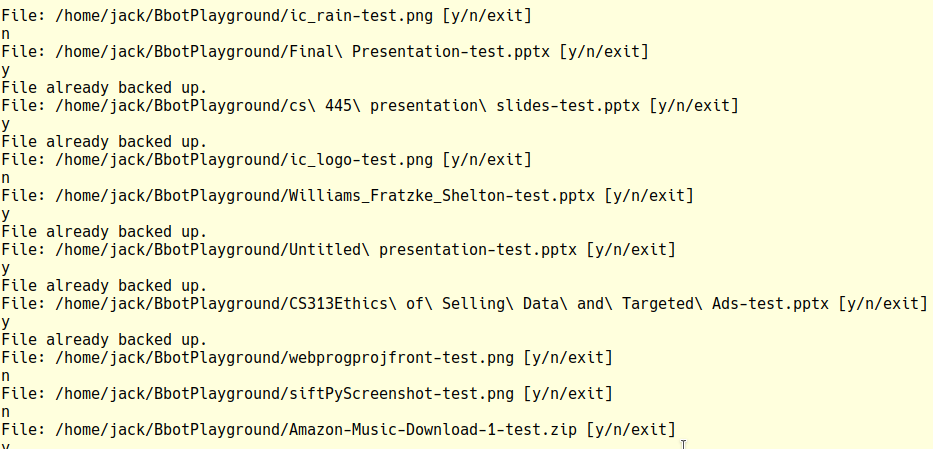


*Typical training session.*

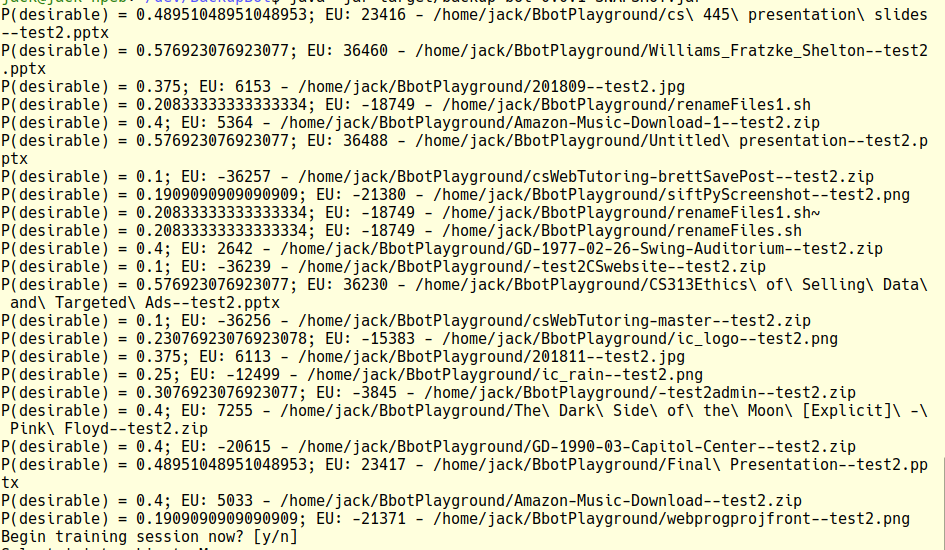
Next, we are going to manually ignore all the image files. 

Next (not shown) we are going to cp all of our zips, png’s, jpg’s, and ppptx’s over to a second directory. We will tell the bot to crawl that directory by altering UserConfigConstants. We will re-create our executable jar. Because the files have a new path, they will be recognized as new files which haven’t been decided on.

Now, we will run one more training session with these new files. Then we will rename the files to test the bot a third time.



*Second Training Session. The agent is no longer certain about images. The “File already backed up” message simply indicates that the bot had stored the file, but still queried the user about it.*



*Results after two trainings and the six manual ignores of the images. We can see that it learned our png preferences and is closer to not backing up the jpg’s. It learned that we like pptx’s. It learned our zip preferences, minus the zip that was almost 1GB; the file ended up with a higher probability, but was ultimately not kept because of the large file size.*

**Computer Code Description:**

Program execution starts in drivers.Driver in the main method. First, persistent data is recovered, and instance coupling is performed. Persistent data includes KEPT and IGNORED file records, the probability counts which map a size or extension to the probability that the feature is desirable, and the priority queue of undecided files to query the user about. Next, a backup cycle will start:

First, a python script (“constructFind.py”) parses the “ignore.txt” file and creates a “find” command that will ignore those files. Then the find command is ran. Any files which we don’t have persistent records on are UNDECIDED. The storage.RecordStorage class keeps persistent data on these storage.FileRecord objects which can be either UNDECIDED, KEPT, or IGNORED as defined by the constants.BackupStatus enumeration.

The set of files we will work with are the UNDECIDED ones. First, attribute sizes and extensions to these files in the drivers.FeatureAttribution class’ static method attributeFeatures. Now, the drivers.Decision class’ static method getOrderedBackupList will decide on what files to backup. This method returns a priority queue of probability.BackupAction objects. This class keeps the FileRecord and the expected utility and has a compareTo method that orders based on expected utility.

The driver.BackupManagerImpl object has a method backupFiles which takes this list and backups files as described in the algorithm: the probabilities of the file’s features are updated, the file is moved onto the storage media, and the file’s BackupStatus is updated.

The BackupManager also composed the ordered list of most undecided files during the Decision process (using probability.QuestionableFile objects which keep the file and the probability of desirable and use the compareTo method to order), and the Critic will now use this list to query the user. The Critic will then use the BackupManager methods keepFile or ignoreFile methods as appropriate. These methods handle all KEEP and IGNORE operations as described in the algorithm above. Next, the Critic, which doubles as the user interface, displays menu options. From here, the user can manually KEEP or IGNORE files.

Finally, the appropriate objects (above) are stored persistently.

The above discussion gives a high-level overview of the program flow and what classes are used where. Following will be some bits and pieces that describe the remaining classes uses and other relevant details.

The performance measure can be easily modified using constants.PerformanceMeasure. The default values are explained in the comments. Our system could track more features. Simply, add features to constants.FeatureTypes and then attribute features to files in the AttributeFeatures class. A FeatureCountTup object keeps the FeatureType along with the specifics of that feature, and the prior knowledge and means of altering it are described in comments in the constructor. PersistentManager has methods for storing and recovering persistent data.

**References:**

* Hale, Peter. “Acronis True Image 2018: Artificial Intelligence Meets Intelligent Backup.” Acronis.com, 9 July 2018, [www.acronis.com/en-us/blog/posts/acronis-true-image-2018-artificial-intelligence-meets-intelligent-backup](http://www.acronis.com/en-us/blog/posts/acronis-true-image-2018-artificial-intelligence-meets-intelligent-backup).
* Foster, Don. “Putting Backup and Recovery to Work With Artificial Intelligence, Machine Learning.” *Commvault*, 26 April 2018, [www.commvault.com/blogs/2018/april/putting-backup-and-recovery-to-work-with-artificial-intelligence-machine-learning](http://www.commvault.com/blogs/2018/april/putting-backup-and-recovery-to-work-with-artificial-intelligence-machine-learning).
* Russell, Stuart J., and Peter Norvig. Artificial Intelligence: a Modern Approach. 3rd ed., Prentice Hall, 2010. (p. 54-55, 505, 622).
* Processes/APIS
  + rsync
  + GNU find and ls
  + Java Process Builder
  + java.util.Collections objects like HashSet and PriorityQueue