AI Video Game Data Classifier

Our department collects large chunks of assorted data from legacy servers left over after development teams disband or studios operated by our company close. In most cases the people who created that data and knew how to use it have moved on resulting in a loss of tribal knowledge needed to make sense of it. Locating, identifying, and organizing data from these source dumps without guidance is therefore difficult and extremely time consuming since it contains millions of files most of which live in deep directory trees with multiple branches.

So, what if we could use machine learning instead to do the task? In other words, could we use an AI tool to locate and categorize files in server dumps? And if so, what kind of input would such a tool require to make its predictions?

In the absence of guiding documentation our main source of information is the file structure of extracted source dumps. Each file structure consists of thousands or millions of paths that point to the files stored in the legacy servers. This information needs first to be 'featurized' such that it can be fed to an AI classifier. Our approach to achieve this was to consider each path as a natural language sentence. The assumption was that although pathnames are not exactly proper sentences, they do contain meaning pertaining to the files they point to and we hoped this meaning could be extracted using natural language processing (NLP) techniques.

The task for us then was to create listings of these paths with file copy tools such as robocopy and then process these listings to filter out slashes, numbers, and other non-alphabetic symbols. Next, these path-sentences would need to be tokenized and featurized (in NLP this is generally achieved using the bag-of-words or tf-idf representation). Once featurized, the data could then be fed to the NLP AI classifier for prediction.

At first, we tested out-of-the-box NLP classifiers to see if we could get better than random results. We just looked at binary classifiers to keep things simple. Also, these classifiers accepted basic (English) text input. That is, the out-of-the-box tool took care of featurizing the data prior to processing thus saving us this work.

The question asked was whether a given path contained art or not. For supervised models that accepted additional training data we provided ~1000 path-sentences (manually labelled: 500 'yes' and 500 'no'). The best results ranged between 60-65 % prediction accuracy. Not great but better than random (50%) which is all we wanted to determine with this exercise.

The main reason for the low accuracy results was that although English based, our pathnames contain truncations, abbreviations, or word and letter combinations that are not found in regular English. Also, certain words may have a very different meaning in our file structure context than in regular English. That is, there is jargon specific to the game industry and this is not captured in out-of-the-box classifiers. Therefore, we knew we had to train a model from scratch to improve accuracy and add more categories.

For this we needed data. Lots of it. Now it so happened that as part of a migration effort from on-prem storage to a 3rd party cloud our team generated thousands of robocopy logs for archives that we knew to be complete and well documented. We used this data to program a parser that produced lists of paths from those robocopy logs and labelled them as either pointing to audio, video/animation, static

art, source code or documentation. The latest dataset obtained from the parser contains around 10 million paths of which approximately 9 million were used for training and 1 million for testing.

To date, we obtained our best results with a Naïve Bayes Multinomial model. There are more approaches we plan to try, but current results are already quite promising. To measure the performance of our models we generated confusion matrices for randomly chosen test sets. Confusion matrices are commonly used for multi class problems since they display prediction performance for each individual category. Also, by looking at where the AI is giving us false positives or negatives we can determine where the model is getting "most confused".

The matrix below is one of many we have generated during our tests. Note that the results for a given model were very similar for each of the generated matrices regardless of how we varied our training and test sets:

	Audio	Video	Static	Code	Docs	Other
Audio	**0.921** *(29410)*	0.020 *(639)*	0.020 *(642)*	0.01 *(343)*	0.003 (104)	0.025 *(811)*
Video	0.035 *(328)*	**0.758** *(7529)*	0.069 *(686)*	0.040 *(397)*	0.012 *(119)*	0.086 *(851)*
Static	0.002 *(312)*	0.027 *(4030)*	**0.813** *(121541)*	0.067 *(10060)*	0.013 *(1982)*	0.077 *(11512)*
Code	0.004 *(551)*	0.003 *(428)*	0.032 *(4430)*	**0.824** *(114618)*	0.022 *(3101)*	0.115 *(15935)*
Docs	0.005 *(62)*	0.014 *(165)*	0.040 *(460)*	0.289 *(3337)*	**0.580** *(6701)*	0.071 *(823)*
Other	0.030 *(4365)*	0.018 *(2620)*	0.156 *(22551)*	0.295 *(42513)*	0.026 *(3806)*	**0.473** *(68155)*

The rows correspond to true values, while the columns show the predicted values. Truth and prediction meet at the diagonals. The above matrix is "true" normalized, meaning we are dividing samples by all true values for a given category. This tells us the closer the value is to 1.000 at the diagonals the better our model's sensitivity (1.000 being a perfect score). Another option is to normalize by dividing samples by all the predicted values. In that case the diagonals would be giving us the precision or "positive predictive value" for a given category.

Our conclusion from our overall results is that the model is performing reasonably well when predicting the location of ‘audio’ data and gets respectable results for ‘static art’. Sensitivity is good for ‘code’ but precision is not so good. Sensitivity is passable for ‘video’ but precision is not great. Results are rather poor for ‘documentation’ and ‘other’ categories. Various tweaks of our model do give better numbers for the more poorly performing categories although so far it has been at the expense of lowering performance of other categories. However, we are still in the early stages of our experimentation. Other approaches may work better than Naïve Bayes for the ‘bad’ categories such that in the end we may use a combination of predictors to do the full categorization.

Additionally, we have calculated F1 scores to assess the overall performance of the model. The three scores we looked at are the F1-micro for all categories, the F1-micro for combinations of ‘audio’, ‘video’, ‘static’, and ‘code’ categories, and the F1-macro for the same. Explanations on how the F1 scores are calculated and what they measure exactly is beyond the scope of this document, but we would provide those details in our demonstration.

We have also applied additional filters to improve performance. One such filter is to discard predictions made below a certain probability (confidence). Setting a confidence threshold achieves better results at the expense of filtering out some true positives with the false ones. Precision-sensitivity graphs were used to determine what the optimal threshold might be for filtering out results. More details on all this work would be provided in our demonstration.

Finally, we have also done some tests around training an AI on subcategories of the categories mentioned above. Specifically, we have trained a model to predict subcategories of 'audio' (music, special effects, speech, and ambient sounds). Results here have also been quite promising and would be shared as part of a demonstration.