Research Log

Holden Project: Detecting Phony Data

October 12, 2018

Created a script, random-data-generator.R

This takes the CNA.cct file and creates 50 new phony samples.

If does this by finding the range of each protein in the real data and then generating new samples by randomly generating values from within each protein’s range.

The 100 real and 50 phony samples were then randomly split into train and test sets.

Created an organized file structure for the scripts and data

October 15 2018 & October 16 2018

Added the Classification\_Utils.py module to the analysis scripts folder I and an Ipython note book for test several of the learning models.

Naive Bayes and SVC both got 100% accuracies on the training set. Have not run the test set.

Logistic Regression got 93%

MLP (30,30,30,30): 85%

KNN: 72%

Multinomial Bayes: is being dumb and not working

Random Forest: 94%

Gradient Boosting: 97%

Also created script for doing a PCA plot of the random data

Added code to create bar plot of the classification accuracies and save the plot, at the end of Classification-Random-Data.ipynb

October 18, 2018

Modified the random analysis script so it test on the actual test data set and trains on the whole training set. The test set was run one once and no parameters were changed to adjust to it.

Created a script called distribution-data.R that creates phony samples/individuals based on the set of observed values for each protein and script them samples randomly from a distribution of those. \*\*The made up samples consist entirely of real protein expression values found in the original data.\*\* Similar to the random-data-generator.R script this creates 50 fake samples and makes test and train files.

My hypothesis is that since these are real values being used that it will be harder to classify the fake ones. But just visually I can spot the fakes samples because a real sample’s values often match for several consecutive proteins, but in the fake ones they do not. So I predict the machine learning should still pick up on the pattern and do well.

Ran the Dist data through the basic classifiers. Accuracies were not so good. RF has the highest with 71%, which is barely above base line. MLP took 30 minutes to train… Might be over fitting, there are 4 lays of nodes which may be over kill.

Ideas to increase accuracy.

* Feature selection
* Parameter optimization
* Feature creation
  + range and dist of a sample
  + Benford’s Law compliance

October 19, 2018

Made PCA plots of the Dist dataset

Tried doing a tSNE plot, run time for the first, solving for xx takes FOREVER!

Did feature selection on the data and re ran the models. GB and MLP were not run yet

TODO

Finish making a density plot of a random sampling

October 23, 2018

Started using a package for analyzing Bedford’s law, benford\_py

Created plots of the distribution of each sample individually

TODO: visualize the dist of the fake and the real data all at the same time instead of broken up

TODO: find a way to automate saying if something complies with Benford’s law or not

November 6, 2018

Added the information from benford\_py for first and second digit to the training data. It did not improve accuracy. KNN and SVC both scored only .65.

TODO next: look at distributions of the actual second digit, not just the second digit after the first non zero

Using the first digit after the decimal in combination with the other features it still goes poorly

SVM and KNN both getting .65

If using only the distribution of first digits KNN can do 89% while SMV still only gets .65

KNN

accuracy: 0.89 (+/- 0.12)

Runtime: 0.005766558647155762 minutes

SVC

accuracy: 0.65 (+/- 0.00)

Runtime: 0.006466662883758545 minutes

RF

accuracy: 0.94 (+/- 0.12)

Runtime: 0.02681665023167928 minutes

Gradient Boosting

accuracy: 0.94 (+/- 0.15)

Runtime: 0.0597168763478597 minutes

Niave Bayes

accuracy: 0.96 (+/- 0.08)

Runtime: 0.005083171526590983 minutes

LR

accuracy: 0.65 (+/- 0.01)

Runtime: 0.004949978987375895 minutes

MLP

accuracy: 0.65 (+/- 0.00)

Runtime: 0.172183620929718 minutes

Using the second digit plus the protein data of after the decimal place once again KNN and SMV get .65 percent but using just the decimal place we get the following results:

KNN

accuracy: 0.90 (+/- 0.13)

Runtime: 0.0051833430926005045 minutes

SVC

accuracy: 0.65 (+/- 0.00)

Runtime: 0.007800118128458659 minutes

RF

accuracy: 0.95 (+/- 0.10)

Runtime: 0.026099916299184164 minutes

Gradient Boosting

accuracy: 0.91 (+/- 0.12)

Runtime: 0.05738329887390137 minutes

Niave Bayes

accuracy: 0.98 (+/- 0.06)

Runtime: 0.004850411415100097 minutes

LR

accuracy: 0.65 (+/- 0.00)

Runtime: 0.005133358637491862 minutes

MLP

accuracy: 0.65 (+/- 0.00)

Runtime: 0.1698333740234375 minutes

New Idea, try using both the first and the second digit

November 15th 2018

On Tuesday I made another script that did classification based on the first and second digits distributions. It overall does better than either of the single sets of features.

Also I received imputation software (DreamAI) from Sam which he got from Pei Wang. I put it in the Holden folder for convenience but also git ignored it (listed it in the .gitignore file) so I am not publicly publishing someone else’s work.