Capstone Project

I. Definition

Project Overview

For computer scientist, physicists and mathematics Integer Sequences are a very important part of their daily work but historically, sequences have been an integral part of the scientific endeavor, as human beings have been always fascinated by patterns, and our ability to detect such patterns in sequences of numbers seems to be unlimited(1).

Even before Pythagoras mathematicians have been developing formulas and theorems that provide the theoretical basis for understand numerical patterns, modern computational technologies have developed different algebraic systems that perform symbolic and high-precision computations, but even though they're very advanced in terms both of computing power and graphical capabilities they are still no more than aides for the scientists(1).

The On-Line Encyclopedia of Integer Sequences (OEIS), also known simply as Sloane's, and the Journal of Integer Sequences are the main investigative effort in the area of Integer Sequences, both founded by Neil Sloane, are now collective efforts with the main objective of record new sequences as they are discovered and provide the theoretical support for those sequences. So, Machine Learning seems like the next logical step, where a deep level of pattern recognition can be used to find patterns without the inflexibility of mathematical rigor(2).

Integer Sequences are also the subject of an active research community, the main source for papers in this subject is the Journal of Integer Sequences (https://cs.uwaterloo.ca/journals/JIS/), also founded by Neil Sloane and now maintained by the School of Computer Science of the University of Waterloo, in Waterloo, Ontario.

For this problem the main datasets come from On-Line Encyclopedia of Integer Sequences (OEIS) since it's recognized as the biggest collection of integer sequences and it's a concerted effort that have been being developed since 1964, and there doesn't seem to be any other datasets that compares it to it in its completeness.

For this specific solution we have used a subset of the whole OEIS dataset obtained from the Kaggle platform, it contains the majority of the integer sequences from the OEIS. It is split into a training set, where the full sequences are given, and a test set, where the last number from the sequence has been removed. The datasets were downloaded from the following link: https://www.kaggle.com/c/integer-sequence-learning/data.

Problem Statement

For many years' scientists have been trying to discover new patterns in number sequences and developing formulas and theorems that describe the fundamentals of such patterns. Those sequences are not mere mathematical curiosities but powerful tools for many fields like number theory, combinatorics, and discrete mathematics(1).

Currently there are many software applications that provide computational algebraic analysis and graphical capabilities that help scientist if their quests of finding and defining new sequences, at first glance it may seem like this symbiotic relation between man and machine will continue to be the main source of new sequences, unless a deeper level of analysis of how patterns arise can be achieved, so new sequences can be discovered based on a fundamental understanding of their nature instead of a one-to-one basis(1).
In consequence there is a clear need of better tools for understanding how sequences behave and predict such behavior, and this is the problem I decided to tackle: predict the next number in a specific sequence based on how it behaves. For this I implemented a Logistic Regression using a dataset of integer sequences where I can create classifiers and features based on the behavior of the sequences, i.e. whether it contains zeroes or negatives, or the divisibility of the last digit. All this with the objective of generating an accuracy score to evaluate how good the algorithm predicts a subsequent digit in a sequence.

Evaluation Metrics

Note to the reviewer:

The reason I selected this problem was because it seemed a little cryptic and not mainstream, and I wanted to have the chance of looking for innovative solutions to not-so-popular problems. But I found not only that the problem was more complex than expected but that there's not a lot of public research in the era, ie. there doesn't seem to be a lot of researchers or machine learning engineers trying to tackle this specific problem.

Faced with this situation I tried to stay working on the same problem but with a more limited approach, for that reason the solution I proposed was *not* one that can universally predict the next number of any sequence but if under a specific condition a prediction can be made about the final digit in a sequence, in other words the question is not "which is the next number?" -that would have a continuous outcome: 0,1,2,3, -5, -3456, etc.- but "is the most common digit also the last digit in a sequence?" which has a binomial result: yes or no, and for this job Logistic Regressions and its ability to convert a binary variable into a continuous one that can take on any real value (negative or positive) seems to do the job very well.

I propose the use of a classification metric as an evaluation metric for this problem, more specifically Area Under Receiving Operating Characteristic (ROC) Curve (or AUC for short) which is a performance metric for binary classification patters. The AUC represents the model's ability to predict the last number of a sequence. An area of 1.0 represents a model that made all predictions perfectly, and an area of 0.5 represents a model as good as random(3). Since we are using normalized units the area under the curve is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one (assuming 'positive' ranks higher than 'negative'). This can be seen as follows: the area under the curve is given by (the integral boundaries are reversed as large T has a lower value on the x-axis)

$$A = \int_{\infty}^{-\infty} \mathrm{TPR}(T)\mathrm{FPR}'(T)\,dT = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I(T' > T) f_1(T') f_0(T)\,dT'\,dT = P(X_1 > X_0)$$

where X_1 is the score for a positive instance and X_0 is the score for a negative instance(4). The calculation for our problem was done using this trapezoidal rule(5).

II. Analysis

Data Exploration

For this problem the main datasets come from On-Line Encyclopedia of Integer Sequences (OEIS) since it's recognized as the biggest collection of integer sequences and it's a concerted effort that have been being developed since 1964, and there doesn't seem to be any other datasets that compares it to it in its completeness.

For this specific solution we have used a subset of the whole OEIS dataset obtained from the Kaggle platform, it contains the majority of the integer sequences from the OEIS. It is split into a training set, where the full sequences are given, and a test set, where the last number from the sequence has been removed. The datasets were downloaded from the following link: https://www.kaggle.com/c/integer-sequence-learning/data.

The dataset consists of a Comma Separated Values (.csv) archive of around 40MB that contain 113846 digit sequences, divided in two files: a training set where full sequences are given, and a test set, where the last number from the sequence have been removed, the generation of both sets as well as the removal of the last digit of the test set was done by the Kaggle platform and was used as it was. Some sequences may have identical beginnings (or even be identical altogether), they have not been removed from the dataset and they're arranged in lexicographic order, indexed by the position of the first term that is greater than 1 in absolute value. Sequences that contain only 0's, 1's and -1's are in lexicographic order by absolute value at the beginning of the table

The datasets will be loaded in memory using Python *csv* libraries and processed to create features to be loaded into *pandas* dataframes. The data set consists only of sequences of integers, and the features are not explicitly defined on the dataset, but will be inferred once they're loaded into dataframes.

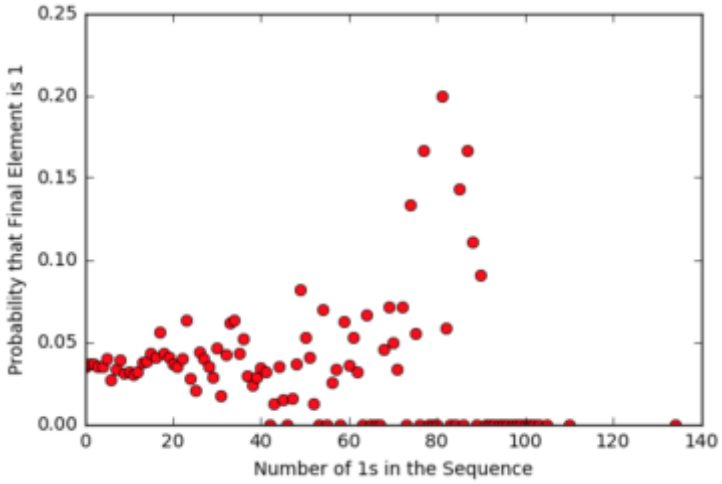
4, 6, 9, 10, 14, 15, 21, 22, 25, 26, 33, 34, 35, 38, 39, 46, 49, 51, 55, 57, 58, 62, 65, 69, 74, 77, 82, 85, 86, 87, 91, 93, 94, 95, 106, 111, 115, 118, 119, 121, 122, 123, 129, 133, 134, 141, 142, 143, 145, 146, 155, 158, 159, 161, 166, 169, 177, 178, 183, 185, 187
Sample of an Integer Sequences in the OEIS dataset – Sequence A001358 Semiprimes (or biprimes)

The following features were created:

- *final_element_count*: the number of times the final element is repeated in a sequence.
- *negatives*: indicates if there are at least one negative digit in the sequence.
- *zeroes*: indicates if there are at least one zero digit in the sequence.
- *max*: the digit with maximum value in the sequence.
- *even_odd_match*: it indicates if the divisibility by two of the last digit is the same as the majority of the digits.
- *features_class*: contains the most frequent final element.

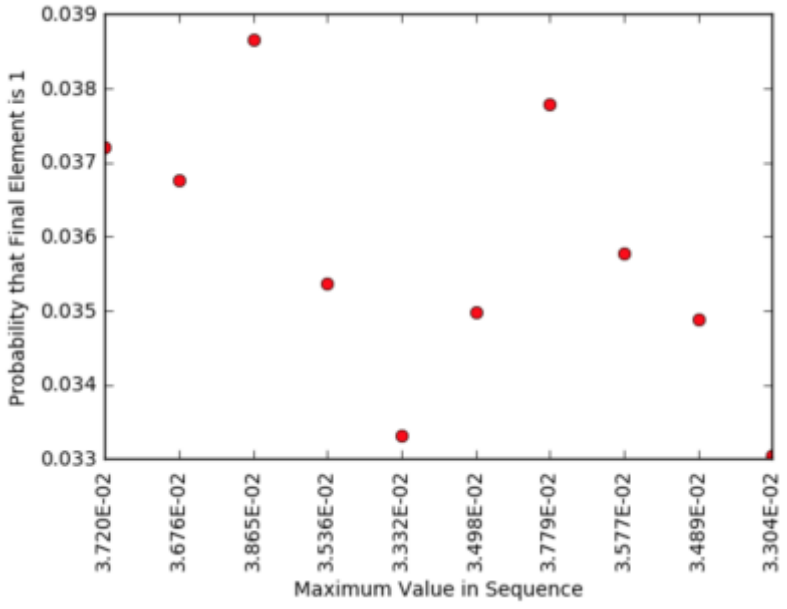
Exploratory Visualization

In order to decide which features should be engineered I decided to plot each feature using *matplotlib.pyplot* in order to evaluate its predictability, some of them performed very well, but some of the didn't. One of the most predictive features was how common the final element in the sequence was, i.e. how many repetitions of that element could be found in the sequence.



As the plot demonstrates the more times the element is in the sequence, the more likely it is that the it will be the last element.

There's also an interesting relationship for the maximum element in the sequence.



This visualization makes clear the necessity of categorize this feature.

Algorithms and Techniques

The solution for this problem is to utilize machine learning techniques to use a classifier that predicts the last number of an integer sequence. In general, the classifier is created by engineering six features (does the sequence has negatives numbers; the maximum value of a sequence; are there zeroes in the sequence; does the final element in the sequence divisibility by two matches that of most of the elements in the sequence) to predict if the most common number in the sequence is the final element, then we use a logistic regression to create the classifier, then I fit the data and a prediction is made, finally a score is calculated using an AUC algorithm (the details of this are on the Methodology section of this report).

For defining the features the following set of methods were developed, including: *how_frequent_is_final_element_in_sequence, maximum_value_in_sequence, are_there_negative_values_in_sequence, are_there_any_zeroes_in_sequence,* and *maximum_value_in_sequence*. These methods take a sequence as input and return a value depending of its intended functionality.

Based on the visualization of the maximum element in the sequence feature, it was necessary to categorize it. The categorization was done based on quantiles in four levels and its named 'maxCategorized'. For this I created the *create_categorized_version_of_feature* method that creates a categorized version of any feature. Once calculated, all the values are stored in a *pandas dataframe* (called 'features').

With the features dataframe in place we can fit the model using a Logistic Regression using the *statesmodel* ***glm*** api ***fit*** method, I decided to use this specific linear regression (logistic) because it allowed me to model the relationship between a dependent variable (features_class) that is categorical instead of scalar and multiple explanatory variables. Finally, a prediction is made using the *statesmodel api* ***predict*** method and a score is calculated using the and Area Under ROC Curve score using *sklearn metrics* library, that is a general function that takes point of a curve and uses the trapezoidal rule for computing the are under the ROC curve.

About Logistic Regressions

Logistic regression is used to predict the odds of being a case based on the values of the independent variables or predictors. The odds are defined as the probability that a particular outcome is a case divided by the probability that it is a noncase and they're used for predicting binary dependent variables rather than a continuous outcome as it expected of a regression model.

Logistic Regression provides a way to convert a binary variable into a continuous one that can take on any real value (negative or positive). To do that logistic regression first takes the odds of the event happening for different levels of each independent variable, then takes the ratio of those odds (which is continuous but cannot be negative) and then takes the logarithm of that ratio. This is referred to as logit or log-odds) to create a continuous criterion as a transformed version of the dependent variable. The logit of success is then

fitted to the predictors using linear regression analysis. The predicted value of the logit is converted back into predicted odds via the inverse of the natural logarithm, namely the exponential function. Thus, although the observed dependent variable in logistic regression is a zero-or-one variable, the logistic regression estimates the odds, as a continuous variable, that the dependent variable is a success(12).

From a mathematical perspective the logistic regression can be understood simply as finding the β parameters that best fit:

$$y = \begin{cases} 1 & \beta_0 + \beta_1 x + \varepsilon > 0 \\ 0 & \text{else} \end{cases}$$

where ε is an error distributed by the standard logistic distribution instead of a Gaussian or Poisson distribution β_0 is the intercept from the linear regression equation (the value of the criterion when the predictor is equal to zero) and $\beta_1 x$ is the regression coefficient multiplied by some value of the predictor. It differentiates from a regression in that fact that β is not a parameter of x or y, i.e. cannot be expressed as a direct function of those variables, instead β is a maximum likelihood estimation that's a function of all observed x and y in all the data, it's thus a heavy computational iterative estimation process (12).

There are many ways of perform this estimation process of this maximum likelihood, but for binary logistic regressions it's being calculated using *iteratively reweighted least squares (IRLS)* that uses a likelihood function called log-likelihood of a Bernoulli distributed process (that makes a distribution binary) using Newton's method that finds successively better approximations to the roots (or zeroes) of a real-valued function (12).

In short, the Logistic regression return a binary value, 1 when the outcome of a complex likelihood function of all the occurrences of x and y in the data is greater than 0, and 0 when it's not.

Benchmark Model

Since the basic approach is to make a prediction of what a subsequent number in a sequence will be, the measure of success is how accurate the prediction was made. I was initially aiming for levels of accuracy over 85% or a score of 0.85 in the specifically Area Under Receiving Operating Characteristic Curve, so the result could be deemed to be at a safe distance of being the same as random. This specific threshold is based on other approaches to this problem as documented in the kaggle platform (https://www.kaggle.com/c/integer-sequence-learning/kernels), there many different approaches were taken, with varied results, but in general they used simple classifiers that resulted in prediction scores around 0.90.

III. Methodology

Data Preprocessing

For this problem no preprocessing of the raw data is needed, since the available datasets were already processed to the point it can be easily loaded into the program an analyzed.

Nevertheless, the data was processed into features as follow:

- *final_element_count*: contains the result of applying ***how_frequent_is_final_element_in_sequence*** to all the sequences.
- *negatives*: contains the result of applying ***are_there_negative_values_in_sequence*** to all the sequences.
- *zeroes*: contains the result of applying ***are_there_any_zeroes_in_sequence*** to all the sequences.
- *max*: contains the result of applying ***maximum_value_in_sequence*** to all the sequences. This feature will later be categorized.
- *even_odd_match*: contains the result of applying ***even_odd_match_in_sequence*** to all the sequences.
- *features_class*: contains the most frequent final element.

The only abnormality that may be to be addressed was that there were some duplicated sequences, but due to the nature of the features it can be ignored, first because the number of duplicates is statistically insignificant (less that 0.2%) and the reason they are repeated is because they technically two different sequences that happen to have the same digits, so from that point of view they *must* be included.

Implementation

The implementation is divided in four main stages: setup, features, visualization, and classification.

Setup.

During the setup stage the dataset .csv files are loaded into memory and the number of registries are counted.

For loading the data, I used Python Pandas library *read_csv* method, that takes a well formed .csv file and loads it into a Pandas *dataframe* object, then using the split method is applied to the dataframe and a Pandas *series* object is created with all the values of all the sequences. This '*sequences*' is the main data object of el the project.

For counting the number of registries, the *countLines* method was created, it takes a string with a relative location, the opens the file on that location using the Python's native *open* method and then returns the number of lines in the file.

Features.

During the classification we will need a set of features to create the classifier, the methods to create such features are the following:

- ***how_frequent_is_final_element_in_sequence***. Returns the number of times the final element is repeated in the sequence. It takes a sequence as an input and returns an integer.
- **maximum_value_in_sequence**. Returns the digit with the maximum value in the sequence. It takes a sequence as an input and returns an integer.
- **are_there_negative_values_in_sequence**. Looks for negative values in the sequence, it takes a sequence as an input and returns true if it finds at least a negative number and false if it doesn't.
- **are_there_any_zeroes_in_sequence**. Looks for zero values in the sequence, it takes a sequence as an input and returns true if it finds at least a zero and false if it doesn't.
- **maximum_value_in_sequence**. It calculates if the majority of the elements in the sequence are even or odd and then compares with final element in the sequence, it takes a sequence as an input and returns true if they match and false if they don't.

As mentioned in the Analysis section of this document, the maximum element in the sequence feature need to be categorized based on quantiles in four levels. The new feature is named and i 'maxCategorized' and is computed using the following method:

- **create_categorized_version_of_feature.** It takes a dataframe of features, a string with the name of the new categorized feature, and a string with the name of feature we want to categorize. The categories are four levels: the first with the 0 to .25 quantile, the second with the .25 to .50 quantile, the third with the .50 to .75 and the fourth the .75 to 1 quantile.

Visualization.

In the visualization stage the features are plotted to evaluate its predictability. For visualization purposes some of the features are categorized and then plotted using the plotPredictibility method. It takes an array with quantiles and a Pandas *series* object for the categories in the x and y axis respectively. The plots are created using the matplotlib.pyplot *plot* method, a MATLAB-like plotting framework that combines pyplot with numpy into a single namespace.

Classifier.

In this stage the model a classifier is created to predict of the most common element in the sequence is also the final element. I used several libraries for the classifier as follows:

- For the classifier I used a Logistic Regression using the *statesmodel* api *glm* method, it takes a string with a formula that defines the dependent and the explanatory variables, in our case, it also takes a family that basically defines the type of regression we are applying and the data. The return value is a Pandas *series* with the classifier data.

  In our case, the dependent variable is “features_class” and the explanatory variables “final_element_countt”, “maxCategorized”, “negatives”, “zeroes”, “even_odd_match” and used the binomial family (sm.families.Binomial) that specifies a binomial error distribution and indicates that it should use a logistical regression function.

  This classifier uses a binary family, this implies that a binomial exponential error distribution and a logistic regression link function is used, also it uses the previously described formula (*"features_class ~ final_element_count + max_categorized + negatives + zeroes + even_odd_match"*), these are the only to parameters than can be tuned, and in our case it required some tuning in defining the independent variable and in consequence the link function, besides that the classifier didn’t require any other tuning. In my opinion it performed fairly well as it resulted in a better than expected score (~0.95).

- Then, the data is fitted using the *statesmodel* api *fit method,* it takes the classifier dataframe as a parameter and returns a dataframe with the fitted data.

- The prediction is made *using statesmodel api* ***predict****,* this method takes a dataframe with the features from the test set and returns a *pandas series.*

- Finally, a score is calculated using and Area Under ROC Curve score using *sklearn metrics* library, that is a general function that takes point of a curve and uses the trapezoidal rule for computing the are under the ROC curve. This function takes two arrays, one with true binary labels in binary label indicators and other with the target scores, it then returns a float number with calculated score.

Refinement

An initial solution was found using only three (3) features, how frequent is the final element in a sequence, are there any negative numbers and are there any zeroes, with a AUC score of ~0.91. From there a process improvement was started first by adding new features, maximum value in the sequence and divisibility by two match, each feature added to the accuracy until it reached a score of ~0.95.

In addition, several type of error distribution were tested, including negative binomial and Gaussian exponential families, both distributions fitted the model less efficiently than a binomial exponential distribution, since they all produced AUC scores under 0.92.

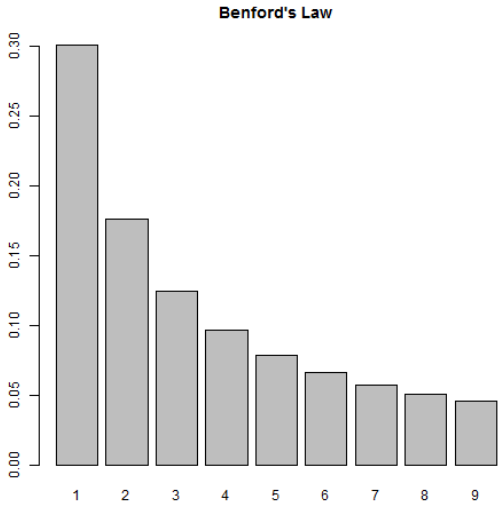
Also, two additional link functions, logarithmic regression and the power transformation, again, they made the ***glm*** function to underperform logistic regression link function and binomial error distribution. Multiple combinations of the aforementioned distributions and link functions, and all of them worked less efficiently than the selected combination.

Another area where improvement was possible is the categorization of features. There, the selection of both the number of levels and the range of values were progressively tuned until it resulted in the best AUC score possible, but in no case the positive impact over the score was over ~0.02 when they scored over 0.9, there were of course many combinations that scored very poorly.

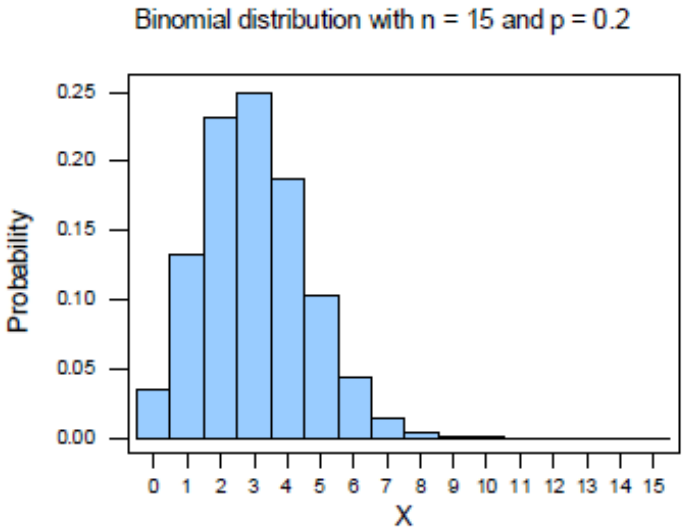
IV. Results

Model Evaluation and Validation

The final model is based on a classifier that predicts if the final element is the most common element final element among the sequences in the dataset. This model was chosen based on the concept of Benford's Law(7) that makes predictions about the distribution, and how its prediction of the n*th* digit in the sequence seems to resemble a binomial distribution as the position of that digit increases in order (see graphs below).



Benford's Law example (8)



Binomial Distribution example(9)

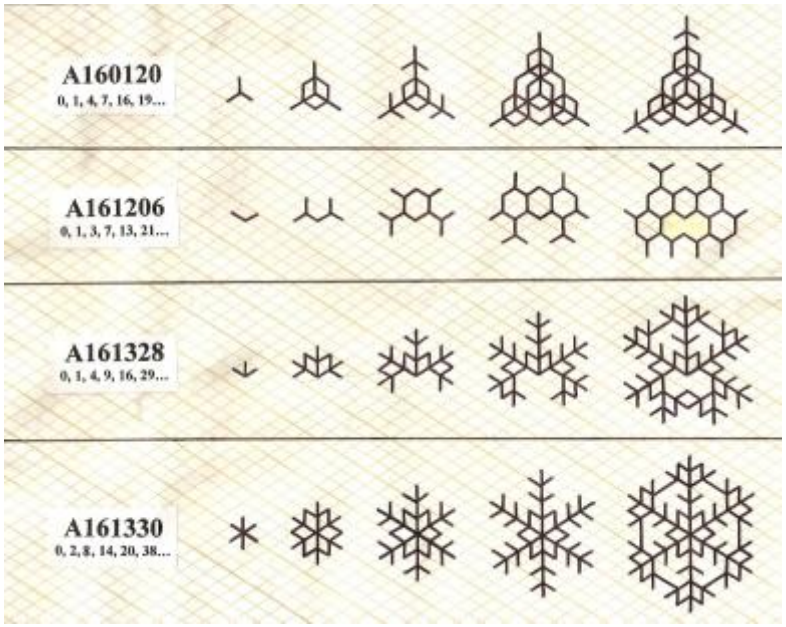
The final model was trained and tested using a technically complete version of the OEIS dataset, that is the most complete compilation of sequences available, therefore the only possible way of testing with different inputs is by using cross-validation, ie. we partitioned the dataset differently several times, so the training and test sets were different every time. I found no significant difference in the final prediction score when using distinct partitions, even when the new subsets were reduced in size, only after a significant reduction in the size of the dataset (greater of 20%) the prediction scores started to fall under 0.90, for this reason I consider the model to be robust and the result trustable.

Justification

Initially a benchmark of a 0.85 AUC score was set as an ideal threshold, as it was far from a 0.50 score that'd be consider random guessing. Since the model consistently scored over that benchmark, we can say that the results we found are stronger than the benchmark, *and from this perspective* it can be said that the final solution is significant enough to have solved the proposed problem i.e. to be able to predict a subsequent digit in a sequence with some level of consistency.

V. Conclusions

Free-Form Visualization



Sequences of numbers are not just ordered sets of digits, they are material or abstract representation of natural phenomena, therefore the ability of predicting the behavior of different sequences can be observed in other ways besides its numerical representation, the implications of this are powerful, especially when there are sequences that describe physical processes, like for example the flow of bodies of water, the reflection of light or even how plants and trees grow, and this is in the very core of each and every of the sequences of the dataset.

In the image we can see several versions of Toothpick Sequences(2):

A160120 Y-Toothpick sequence.
A161206 V-Toothpick (Honeycomb) sequence.
A161328 E-Toothpick sequence.
A161330 E-Toothpick Snowflake sequence.

All of them follow a single pattern of growth in which the number of segments oscillates with a fractal pattern(11). Fractals can be seen in theoretical constructions like cellular automata, or in natural phenomena like honeycombs and snowflakes, they're very elegant examples of the close relation between sequences and less abstract objects and processes, and highlights the potential of any technique that provides the ability of predicting the behavior of sequences of digits.

Reflection

This solution implemented an approach that can be considered as standard for any supervised learning solution: A classifier was created using a set of features, then a model was fitted and a prediction made, nevertheless it required a deeper understanding of the nature of the dataset in a theoretical/mathematical level, here the finding and subsequent analysis of Benford's Law was critical as it allowed me to set the parameters for the *statesmodel glm* function that is extensive in the number of parameters that can be set for the error distribution and the link functions, and even though is very well documented from the programming perspective it gives little information about the theoretical meaning of them.

Being able to predict the behavior of a sequence can have many practical uses, whether in theoretical or in real-life applications, but it seems to me that its utility is ingrained in the understanding of the mechanisms that govern each sequence, more specifically the equations that can generate each and every one of the numbers in a sequence, so it's seems that even though my solution score better than a defined benchmark, but that threshold is defined on a theoretical level and it may be not enough, so it's hard to know if it should be use in a general setting for similar types of projects, this or any solution that is not very close to a score of 1.

On the other hand, this approach, even if it may not be good enough to be full-blown solution of the greater general problem of describing the nature of any sequence of integers, it may serve as a tool for other set of problems, for example it may serve as a measure of randomness, as a sequence with very low scores can be deemed as completely random, a feature than can be useful in areas like information security.

In short, I believe the algorithm behaves better than expected, but I'm not sure of the real life applications of the solutions if the predictions scores are not taken to very high levels, ie. ~0.99.

Improvement

Even though there are plenty of room for improvement, there's an area in particular I think requires improvement since in my opinion it's the main obstacle to overcome if we want to get prediction scores greater than 0.98.

Logistic regressions are a well proven supervised learning method, but it lacks the sophistication of other approaches like random decision forests that are based on decision trees as predictive model instead of linear models, expanding the range of datasets that can be fitted and integrate complex relationships between multiple features and outputs$_{(10)}$. So this is the main improvement I'd do, but for this solution it proven very hard to implement, as the learning curve was steep and very time consuming, but I think a better solution is possible by using random forests that can easily improve prediction scores over the 0.95 benchmark set by this solution.

(1) Staff. "AT&T Researchers — Inventing the Science Behind the Service." *AT&T Labs Research - The Achievement of The Online Encyclopedia of Integer Sequences*. Research ATT, 6 Mar. 2012. Web. 1 Dec. 2016. <http://www.research.att.com/articles/featured_stories/2012_03/201203_OEIS.html?fbid=eb3d4qlYcul>.
(2) "On-Line Encyclopedia of Integer Sequences." *Wikipedia*. Wikimedia Foundation, 2 Mar. 2004. Web. 12 Dec. 2016. <https://en.wikipedia.org/wiki/On-Line_Encyclopedia_of_Integer_Sequences>.
(3) Hallinan, Jennifer. "Assessing and Comparing Classifier Performance with ROC Curves - Machine Learning Mastery." *Machine Learning Mastery*. Machine Learning Process, 04 Sept. 2016. Web. 12 Dec. 2016. <http://machinelearningmastery.com/assessing-comparing-classifier-performance-roc-curves-2/>.
(4) "Receiver Operating Characteristic." *Wikipedia*. Wikimedia Foundation, 15 Sept. 2003. Web. 12 Dec. 2016. <https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curve>.
(5) "Sklearn.metrics.auc." *Sklearn.metrics.auc — Scikit-learn 0.18.1 Documentation*. Scikit Learn, n.d. Web. 12 Dec. 2016. <http://scikit-learn.org/stable/modules/generated/sklearn.metrics.auc.html>.
(6) P. McCullagh and J. A. Nelder (1992). Generalized Linear Models. Chapman & Hall.
(7) "Benford's law." Wikipedia. Wikimedia Foundation, n.d. Web. 10 Dec. 2016.
(8) "Earthquake data and Benford's Law." Statistics you can probably trust. N.p., n.d. Web. 09 Dec. 2016.
(9) "Effect of n and p on Shape." Effect of n and p on Shape | STAT 414 / 415. N.p., n.d. Web. 13 Dec. 2016.
(10) "Decision tree learning." Wikipedia. Wikimedia Foundation, n.d. Web. 13 Dec. 2016..
(11"Toothpick sequence." Wikipedia. Wikimedia Foundation, n.d. Web. 14 Dec. 2016.
(12) "Logistic regression." Wikipedia. Wikimedia Foundation, n.d. Web. 19 Dec. 2016...