Data Science

Capstone Report - Fall 2022

Interpreting Database Columns using Feature Engineering and Machine Learning

Amanda Stuart

supervised by
Yacov Soloman and Li Guo

Preface

With big data becoming more and more prevalent in our daily lives, not only can it be helpful, but it can also be a pain to deal with. That is why I started this project; to tackle one of the many issues that large data sets can cause. While the solution of this project was aimed at helping Ketch, a data analytics company, it can benefit anyone using large unlabeled data sets. I'm Amanda Stuart, an NYU Shanghai senior who double majored in Data Science and Finance.

Acknowledgements

I would like to say a special thank you to Yacov Solomon and Ram Ben-David who I started a variation of this project with. They inspired the idea for me and were a lot of help in the progress of my learning both before and during this project. I would also like to Ketch, the company I worked with over the summer where I started this project. Lastly, I would like to thank Li Guo for helping me with my capstone throughout the semester and keeping me on track.

Abstract

With a lot of data, comes a lot of problems to solve, one of those problems being unlabeled or incorrectly labeled data columns. This can pose a lot of issues in analyzing database columns, or specifically for the company Ketch, how to assign privacy restrictions on specific labels. This is why we created a machine learning model, using a decision tree, that takes in data column samples, and then a function that outputs an assigned label. The function works by running each individual column sample through the prediction function, and then choosing the most prevalent label and assigning it to the column as a whole. Our approach runs with 51% accuracy and hopes to be improved further down the line.

Keywords

Capstone; Data Science; NYU Shanghai; Machine Learning; Decision Trees; Feature Engineering; Classification

Contents

1 Introduction

With big data being a thing of the present, the issue of sorting through and analyzing data has become a very strenuous task. One issue can arise from this, specifically in a database, is having uncategorized information. This can happen either because names aren't uniform, first_name and frst_nm mean the same this but should be FIRST_NAME, or because column names in a database do not indicate what is in their column (col1, col2, col3, etc.) Both pose issues when dealing with extremely large data sets, and would require work from the programmer to fix, time which could be better used elsewhere. That is where this program comes in as a solution. The goal of this project is to create a machine learning model that takes in unlabeled column names and outputs them with the correct labels. The inspiration for this project comes from the working with the company Ketch, a data control platform. Ketches focus is on "responsive infrastructure for data privacy, compliance, and security." [1] This problem arose at Ketch as they were trying to determine the security needed the databases of information that their clients had, but the security was not one solution fits all for the whole database. Some columns need to me more secure than others. For example, information such as first name, last name, and email address can be accessed by all employees at a company, but information such as credit card information or social security number should be more protected and therefore hidden from most employees, or only accessible to a select few. To address this issue however, we must first solve the issue of what is contained within each column. With that said, one solution to this problem is to create a machine learning model that takes in data column samples data and outputs the predicted labels. This way, there would be a uniformed title for each column which would allow the computer to categorize the data on its own and then implement personalized security measures per column. This project will work in two parts which are outlined below, and goes into more depth within the solutions section.

1. Use feature engineering to change the data column samples from string information into integer information.

2. Use machine learning to take in the features and output a labeled data column name

2 Related Work

These three papers are closely related to the approach taken below. They all three aim to solve the same problem of classifying data columns, and so it is helpful to understand how and why they take their own approaches.

2.1 LiveRamp

LiveRamp, a data enablement platform, aims to solve the same problem.[2] LiveRamp addresses a very similar problem to that of Ketch's, where they "receive thousands of large files each day from customers" and "need column type configuration to know how to interpret [those] files." [2] For their data they sampled 10,000 rows for a title of 2.3 million labeled strings from a few thousand column types. They used feature engineering, specifically count vectorization, to identify the frequency of different characters in their strings.

Harshil Patel defines feature engineering as "the act of converting raw observations into desired features using statistical or machine learning approaches." [3] while Heavy.AI defines feature engineering as "the preprocessing steps that transform raw data into features that can be used in machine learning algorithms, such as predictive models."[4] There are a few processes that can be taken for feature engineering. One example is feature creation or extraction. Feature creation is creating a whole new feature that did not exist before[3] while feature extraction is taking important information from the data set and using it to create a feature.[3] These are done in the code below multiple times to create the features, one example is creating the feature of length for each string variable while another example is counting the number of times each alphanumeric character exists in a string to create those features. Another is transformations which are taking a feature and changing it from one representation to another [3], or exploratory data analysis which "can be used to improve your understanding of your data, by exploring its properties."[3] Feature engineering is important in a general sense because it helps reap better results for machine learning models, but in the case of this program, they are essential in order to convert string data into integer data to be fit into a machine learning model.

A few proposed ways of feature engineering that could help this case are One-Hot encoding or count vectorization (bag of characters). One-hot encoding "is a representation of categorical variables as binary vectors. This first requires that the categorical values be mapped to integer values. Then, each integer is represented as a binary vector that is all zero values except the

index of the integer, which is marked with a one." [5] While one-hot encoding is a good way to do feature engineering, the approached outlined above is instead of categorical data rather than for text data. One hot encoding for text data looks more like the image below. It represents each work and symbols as a unique one hot vector which contains binary data as the elements. [6] For the sentence "The cat sat on the mat" Each word becomes its own vector, for example "The" is [0100000] while cat is [0010000].

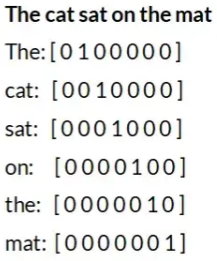

Figure 1: One Hot Encoding Example

Because the data is not all the same length (for example, some samples are 1 word such as "Australia" some are two "United States" and some because much longer, almost the length of a paragraph). This, along with the diversity of the words used, makes using one hot encoding extremely difficult for this data set as well as would make the results complicated in that it would be applying an extremely large number of categories to such a wide range of words. Another approach that is used by LiveRamp is count vectorization or bag of characters [2]. Count vectorization is the process where "we count the frequency of all characters in each string." [2] This is similar to the approach used by the model below. It works well because it assigns a uniform number of features to each string, which is required for a machine-learning model.

Through this approach, LiveRamp receives 11,363 features per string, which is far too many so in order to narrow the features down, they "apply univariate feature selection to select the top 1000 features that are individually correlated with distinguishing different labels as quantified by chi-squared statistics." [2] The top 20 features are number length, length, apostrophes, space length, vowel length, f, b, 9, 8, 5, 6, 7, 4, d, 3, 0, 1, 2, c, and special character length. This information is helpful and all of the features, except for the apostrophe, have been added to the model implemented below. LiveRamp uses a multilayer perceptron which consists of "a single hidden layer of 215 rectified linear units and a softmax output layer." [2] The model classifies 93.7% of strings correctly, and 95.3% of columns correctly. This was an essential reference to

work from and led to a lot of success in the model implementation, specifically the features that were used and the method of classifying individual strings and then grouping them into columns.

2.2 Sherlock

Another, more advanced, implementation of this problem is Sherlock. Sherlock aims to solve the problem of detecting semantic data types because "correctly detecting the semantic type of data columns is crucial for data science tasks such as automated data cleaning, schema matching, and data discovery"[7] and "existing data preparation analysis systems rely on dictionary lookups and regular expression matching to detect semantic types, however, these matching-based approaches often are not robust to dirty data and only detect a limited number of types" [7] Sherlock uses a multi-layer deep neural network for detecting semantic types. It trains on '686,765 data columns retrieved from the VizNet corpus by matching 78 semantic types from DBpedia to column headers." citeSherlock and characterizes "each matched column with 1,588 features describing the statistical properties, character distributions, word embeddings, and paragraph vectors of column values." [7]

Sherlock aims to tackle classification of semantic types because they, "are disproportionally more powerful [than atomic types] and in many cases essential. [They] provide finer-grained descriptions of the data by establishing correspondences between columns and real-world concepts.[7] Sherlock's feature extraction is slightly different from LiveRamps. They extract four categories: global statistics, aggregated character distributions, pretrained word embeddings, and self-trained paragraph vectors. LiveRamp, as well as this project, focus heavily on character distributions to train the model, which is why this model is more advanced. While the other 3 types of features Sherlock uses are not used in the model below, they are excellent in deciding where to go next in order to improve the model.

Sherlock trains "subnetworks for each feature category except the statistical features... These subnetworks "compress" input features to an output of fixed dimension. [They] chose this dimension to be equal to the number of types in order to evaluate each subnetwork independently. Then, [they] concatenate the weights of the three output layers with the statistical features to form the input layer of the primary network." [7] They use two hidden layers with rectified linear unit activation functions and use hidden layer sizes of 300, 200 and 400 for the character level, word embedding and paragraph vector subnetworks, as well as include drop out layers and weight decay terms to prevent overfitting. [7] Some concepts that come from Sherlock that are important

in understanding are Classification and then two classification models with they discuss which are Decision Trees, they decide not to use these, and neural networks.

"Classification refers to a predictive modeling problem where a class label is predicted for a given example of input data." [8] While there are four types of classification, binary classification, multi-class classification, multi-label classification, and imbalance classification, for this program, we are only interested in multi-class classification, where we have 78 possible labels. A classification model works by using "the training dataset and [calculating] how to best map examples of input data to specific class labels." [8] Sherlock also uses classification models, and given this program works on the same model, we do the same. Because the machine learning model used in this implementation is decision trees it is vital to understand what they are and how they work. "Decision trees are trees that classify instances by sorting them based on feature value. Each node in a decision tree represents a feature in an instance to be classified, and each branch represents a value that the node can be assumed." [9] There are two types of decision trees; classification trees which as explained in the name split the data based on different types, which is what is used in this implementation, and regression trees where the decision is a continuous variable. Given this is a classification problem, we will use classification trees over regression trees.

There are a few advantages and disadvatages in decision trees that are vital to understanding them, and in the decision to use them. The first advantage is that "compared to other algorithms decision trees require less effort for data preparation during pre-processing." [10] They also do not require normalization of data, or scaling of data. [10] Given this is a semester long individual project, this is an appealing advantage for this project. Also, "missing values in the data do NOT affect the process of building a decision tree to a considerable extend." [10] Because this program works off of real world data, it is likey that there will be missing values, and because of the size of our data, it is not possible to go through each column to find and correct them, therefore this is relevant to the project. As for disadvantages, "a small change in the data can cause a large change in the structure of the decision free causing instability," [10] Again, given this is real world data, there are not many changes made to the data because we want to keep the results as true as possible to other scenarios so this does not appear to be an issue. Decision trees sometime calculations become far more complex in relation to other algorithms. [10] "Decision trees often involve higher time to train the model" and "Decision tree training is relatively expensive as the complexity and time has taken are more." [10] These are things we have experienced and,

while incontinent, does not affect the results provided, so it is something we can and have deal with. And lastly, "the decision tree algorithm is inadequate for applying regression and predicting continuous values." [10] This program aims to ONLY classify the types, nothing else so this is not an issue.

Neural Networks are "computer programs that operate in a manner inspired by the natural neural network in the brain. The object of such artificial neural network is to perform such cognitive functions as problem solving and machine learning." [11] Neural networks consists of at least 3 layers, the input layer, hidden layers, and the output layer. The input layer, as the name implies, is what takes in the data. The hidden layers are the processing layers. It is where the data is "interpreted, processed and broken down into smaller components." [12] Lastly, the output layer is just that, the output. Some advantages of neural networks are they can interpret visual analysis such as images, however, for the project below this is not necessary. They can also process unorganized data, which again, is not important in our model given all the data is carefully organized in the preprocessing step. Another advantage of neural networks is that their structure is adaptive and "that for whatever purpose an ANN is applied, it alters its course of the structure according to the purpose." [12] As for disadvantages there are a few. First, that it requires a lot of computer power and requires, "heavy machinery and hardware equiptment to work for any application." [12] Next, is that "they can often create incomplete results or outputs. Since ANNs are trained to adapt to the changing applications of neural networks, they are often left untrained for the whole process." [12] Given we don't need a changing environment for our results, this might not be a good trade off for this project. Another is that they need large amounts of data. This would not be a problem for our application as we have hundreds of thousands of columns, that are turned into millions of their own data samples. Lastly, artificial neural networks leave very little control to the coder. While these negatives are too severe for the program we built, they don't necessarily outweigh the positives which is why we are using decision trees where the advantages are well suited for the project.

2.3 Sato

Sato is an even more advanced program than Sherlock, taking a lot of the elements of Sherlock and expanding on it. They aim to solve the same problem because, "detecting the semantic types of data columns in relational tables is important for various data preparation and information retrieval tasks such as data cleaning, schema matching, data discovery, and semantic search"

[13] with the same complaints that, “existing detection approaches either perform poorly with dirty data, support only a limited number of semantic types, fail to incorporate the table context of columns or rely on large sample sizes for training data.” [13] Sato aims to correct for some of the downfalls of Sherlock which it states are that it, “under-performs for types that do not have a sufficiently large number of samples in the training data” [13] and it “uses only the values of a column to predict its type, without considering the columns context in the table.” [13] For the second point Sato provides the example of a column containing ‘Florence,’ ‘Warsaw,’ and ‘London.’ These could be determined as LOCATION, CITY, or BIRTH PLACE, but it is unclear given the data. However, if they were within a table that held biological information, then it would more accurately be declared BIRTH PLACE. [13]While the first issue appears to be a problem in our data set as well with types with smaller supports performing poorly, the second issue is not one we are able to solve with our data set. This is because each column stands on its own and is therefore not placed within a table we could use as additional information. Because of the advanced level of this model, it will not be too helpful in the report given the other sources found, however, it is a great reference for what to do next.

3 Solution

3.1 Data set

The dataset used for this project is the same dataset used in Sherlock: A Deep Learning Approach to Semantic Data Type Detection. First to select the number of types they used they “adopted the types described by the T2Dv2 Gold Standard, the result of a study matching DBpedia properties with columns from the Wed Tables wed crawl corpus.” [7] and then “these 275 DBpedia properties, such as country, language, and industry, represent semantic types commonly found in datasets scattered throughout the web.” [7] Sherlock took this approach, which we found appropriate because it collects the most popular data types and provides us with a few important things for data collection. First, because these are the most widely used, Sherlock had no problem finding a significant enough amount of data to be used in the machine learning model, and having a significant amount of data is needed to train a machine learning model. Also by working with the most popular data types, the program created will be the most useful for later implementations because it will most likely include all the important columns needed to be sorted. This makes the model more significant for later use, rather than being a one-use case machine learning

model. Next, to collect data column samples, Sherlock turns to the Viznet repository [14], "which aggregates and characterizes data from two popular online visualization platforms and open data portals, in addition to the Web Tables Corpus." [7] Their matching process resulted in 6,146,940 columns matching 275 types, however, the data section used in this program is a smaller subset of that with roughly 400,000 data columns used for training spread across 78 different types, and 100,000 data columns used for testing.

3.2 Feature Engineering

Once the data was collected, next, we had to translate the string data into integer data somehow. This is because a machine learning model such as Decision Trees cannot take in string information but rather requires integer data to complete the model. Within the program, we have a function which takes in a string and outputs data on it. The data collected starts by obtaining information such as string length, the number of integers present, the number of spaces present, the number of special characters present, and the number of vowels present, and then the number of each alphanumeric character, a-z and 0-9, and assigns it to a variable. This then provides us with 41 different features per data column sample. The inspiration for these data column collections comes from the Medium article by LiveRamp which is mentioned above. [2]

3.3 Machine Learning – Decision Trees

With the data and features decided, the next step was creating the machine learning model and running the data on it. The model used is a decision tree because of its simplicity in preprocessing the data. The methodology to run the program goes as followed. First, the training data is loaded in as two different files, train_values and train_labels where each list has the same index, and train labels contain the data labels such as COUNTRY, ADDRESS, AFFILIATION, and train values contains the same index with data column values attached to it. For example, the index associated with COUNTRY, might have the data "Australia", "United States", "Canada", "China" associated with it. Below is an image to illustrate this concept.

	values
55030	['Global', 'United States', 'Australia']
167000	['Fiction, Adult - Non-Floating', 'Fiction, Ad...
638282	['', '', 'University of Puerto Rico - Rio Pied...
232298	['Laughology', 'MTV', 'With Intent to Kill', '...
316158	['Mare', 'Gelding', 'Gelding', 'Gelding', 'Gel...
...	...
344035	['Management of companies and enterprises', 'L...
167866	['Fiction, Adult', 'Fiction, Adult', 'Fiction,...
256742	['This document presents strategies practition...
253456	['0-1ft', '5-6ft', '8-10ft', '17-20ft']
491858	['Norwegian Cod Liver Oil Cherry', 'Norwegian ...

412059 rows × 1 columns

Figure 2: Raw Data

Next, each list identified with a training index is broken down to one element and tagged as its label. For example instead of having COUNTRY associated with the list [“Australia”, “United States”, “China”, “Canada”], each element will have its own association with COUNTRY. For example, [“Australia”] is labeled as COUNTRY, [“United States”] is labeled as COUNTRY, and so on. This concept is portrayed below as well.

```
[[['Global', 'area'],
  ['United States', 'area'],
  ['Australia', 'area'],
  ['Fiction', 'collection'],
  ['Adult - Non-Floating', 'collection'],
  ['Fiction', 'collection'],
  ['Adult', 'collection'],
  ['Fiction', 'collection'],
  ['Adult', 'collection'],
  ['Fiction', 'collection'],
  ['Adult', 'collection'],
  ['Fiction', 'collection'],
  ['Adult', 'collection'],
  ['', 'team Name'],
  ['', 'team Name'],
  ['University of Puerto Rico - Rio Piedras', 'team Name'],
  ['University of Puerto Rico - Rio Piedras', 'team Name'],
  ['University of Puerto Rico - Rio Piedras', 'team Name'],
  ['University of Puerto Rico-Rio Piedras', 'team Name'],
```

Figure 3: Split Data

Once the data is organized in this way, the list of column elements is run through the preprocessing method which translates the element, such as [“United States”] into the features mentioned above. At this point in the code, the program is working with two lists, one that contains the

label information, and the other that contains lists of the features for each element. These two lists are run through a decision tree model and used to fit the model.

Once this is completed, the testing data must be transformed into a similar format in order to run predict method. A function was created to run the predict method and predict each set of sample data. Because our Decision Tree model was run on individual elements of the data column we must run the predict on a similar format in order for it to run correctly. In order to do this, the function takes in the test values as well as the indexes that it should be run through. For each index, the function takes the list of elements and splits them and cleans them as is done to the training data. It then runs predict on each individual cleaned sample and adds each prediction to a list. Finally, for the specific index it chooses the most prevalent predicted title for the sample. For example, using the COUNTRY example, if "Australia", "United States", and "China" are predicted as COUNTRY but "Canada" is predicted as AFFILIATION, the label COUNTRY will be assigned to the grouping. The method loops through each index that was run through the function and does this for each sample, outputting a list of predicted labels associated with the sample.

4 Results and Discussion

To test the accuracy of training on 300,000 data samples, we used 80,000 testing columns to get a returned accuracy of 51%. To observe the results I printed out a classification report below to observe which types do better than others, and observe where improvements need to be made in the program. First three definitions to better understand results.

Precision: The number of positive class predictions that actually belong to the positive class.[15]

Recall: The number of positive class predictions made out of all positive examples in the dataset. [15]

F1-Score: Provides a single score that balances both the concerns of precision and recall in one. [15]

Overall, we get a macro average precision of .45, recall of .59, and F1-score of .46 and a weighted average precision of .62, recall of .51, and F1-score of .51. One issue that becomes clear viewing the results is that types that have smaller support sizes do poorly compared to types with larger support sizes. For example, the six types with less than 50 support samples do not receive a precision over 0.05 while the nine types with support size of over 2000 samples, except for gender,

receive a precision score of at least .50, and six of those are above 0.70.

Some of the best performing types were Isbn with a precision score of .96, recall of .99 and F1-score of .98, jockey with .83, .98, .90, and industry with .81, .96, and .88.

Some of the worst performing types were plays with .0, .0, and .0 for precision, recall, and F1-score respectively, however, plays had 0 support samples, followed by birthplace, with .0, .06, and .01 and then elevation with 0.01, 0.08, and 0.01.

	precision	recall	f1-score	support
address	0.08	0.55	0.13	238
affiliate	0.02	0.40	0.03	5
affiliation	0.53	0.97	0.69	573
age	0.55	0.80	0.65	1183
album	0.07	0.70	0.14	191
area	0.53	0.96	0.68	647
artist	0.32	0.87	0.47	639
birth Date	0.01	0.08	0.02	36
birth Place	0.00	0.06	0.01	16
brand	0.27	0.60	0.37	150
capacity	0.34	0.96	0.51	81
category	0.24	0.79	0.36	542
city	0.21	0.58	0.30	607
class	0.14	0.83	0.23	284
classification	0.50	0.72	0.59	243
club	0.23	0.95	0.37	437
code	0.76	0.87	0.81	1508
collection	0.44	0.94	0.60	125
command	0.47	0.96	0.63	297
company	0.54	0.87	0.66	1095
component	0.42	0.93	0.58	311
continent	0.03	0.33	0.06	12
country	0.41	0.61	0.49	1203
county	0.33	0.59	0.42	938
creator	0.23	0.86	0.36	51
credit	0.17	0.85	0.28	103
currency	0.64	0.97	0.77	159
day	0.66	0.37	0.47	3139
depth	0.18	0.09	0.12	1129
description	0.24	0.47	0.32	878
director	0.10	0.21	0.14	57
duration	0.42	0.93	0.58	799
education	0.23	0.60	0.33	70
elevation	0.01	0.08	0.01	66
family	0.76	0.94	0.84	347
file Size	0.61	0.95	0.74	136
format	0.78	0.96	0.86	1409
gender	0.09	0.02	0.03	3252
genre	0.64	0.65	0.64	659
grades	0.79	0.96	0.87	816
industry	0.81	0.96	0.88	1457
isbn	0.96	0.99	0.98	820
jockey	0.83	0.98	0.90	1376
language	0.42	0.89	0.57	410

Figure 4: Results for labels

language	0.42	0.89	0.57	410
location	0.43	0.47	0.45	1561
manufacturer	0.25	0.58	0.35	252
name	0.25	0.24	0.24	1858
nationality	0.16	0.43	0.23	89
notes	0.52	0.16	0.24	4476
operator	0.23	0.19	0.21	267
order	0.73	0.97	0.84	636
organisation	0.31	0.28	0.29	170
origin	0.60	0.40	0.48	1271
owner	0.65	0.58	0.61	1099
person	0.45	0.08	0.14	1928
plays	0.00	0.00	0.00	0
position	0.42	0.40	0.41	1850
product	0.70	0.65	0.68	1664
publisher	0.73	0.42	0.54	868
range	0.55	0.74	0.63	240
rank	0.29	0.27	0.28	1869
ranking	0.16	0.73	0.26	52
region	0.58	0.46	0.51	1973
religion	0.16	0.25	0.19	125
requirement	0.68	0.52	0.59	241
result	0.87	0.79	0.83	1920
sales	0.04	0.22	0.07	36
service	0.84	0.77	0.80	1455
sex	0.94	0.61	0.74	2710
species	0.74	0.35	0.48	1067
state	0.86	0.36	0.51	4262
status	0.88	0.33	0.48	4815
symbol	0.89	0.54	0.67	1716
team	0.71	0.55	0.62	2232
team Name	0.66	0.46	0.54	1291
type	0.82	0.38	0.51	3775
weight	0.15	0.82	0.25	318
year	0.92	0.46	0.61	3420
accuracy			0.51	80000
macro avg	0.45	0.59	0.46	80000
weighted avg	0.62	0.51	0.51	80000

Figure 5: Results for labels

Birthplace, while it has a low support size, was most likely miscategorized with place. While it would be ideal to check this with a confusion matrix on all samples, given the time restraint, and the amount of time that it takes for the code to run, I was only able to run it on a sample of 50,000 training samples and 10,000 testing samples, however, running it on this sample size produced a similar accuracy at 47% and a macro avg of precision at .41, recall at .52 and F1-score and weighted avg of precision at .63, recall at .47 and F1-score at .49. Therefore it is acceptable to use the smaller sample size given the time restraint and similar scores.

The sheer size of the confusion matrix makes it almost impossible to view the whole table at once given it is 78 columns by 78 columns. Therefore I am unable to add an image of it below, so I will summarize some important results instead. (The confusion matrix has been turned into a numbers file and added to the code google drive link below for further exploration.) An important note within this confusion matrix is that the labels down are the true labels and the labels across

are the assigned labels.

For the sake of simplicity, we will not review all 78 labels, but instead 3. First, notes, because notes has a support size of 534, but a precision of 0.42, recall of 0.13, and F1-score of 0.20. This is the highest support size, with a relatively low accuracy so it is worth it to take a look into. Notes are falsely assigned address: 1, age: 1, club: 1, day: 1, description: 4, duration: 1, location: 1, name: 2, owner: 1, person: 6, position: 5, product: 3, publisher: 4, range: 2, region: 2, result: 2, service: 6, sex: 10, species: 6, state: 4, status: 9, team: 1, team name: 6, type: 10, weight: 1, and year: 5, while only classifying notes correctly 70 times. Due to the nature of a type such as note, this can be because it varies greatly, but taking a look at the notes information would provide us with much more information and allow us to correct this and possibly suggest new features.

Another type that might be helpful to look at is gender because it also receives a low precision and a high recall with 0.07 and 0.60 respectively. Given we also have the type sex the assumption here is that these two get confused with each other, however, sex has a precision of 0.96 and recall of 0.41. Gender is inaccurately labeled as age: 2, notes: 4, sex: 75, and symbol: 1, but only correctly labeled as gender 6 times. In this case our assumption was correct that gender and sex get confused with each other. This begs the question of how to deal with this. We have two choices: do further digging into the two types and learn how the columns are similar or different to aid the training process or combine them given they appear to be the same column.

Lastly, let us look at city which, as the example in SATO can be a confusing type to identify in comparison to others. City gets a precision score of 0.23, recall of 0.52, and F1-score of 0.32. City is incorrectly classified as address: 2, classification: 1, company: 1, country: 1, county: 15, gender: 1, location: 5, manufacturer: 2, name: 3, notes: 11, person: 28, position: 5, product: 3, publisher: 1, region: 28, result: 3, service: 1, sex: 5, species: 1, state: 22, status: 5, symbol: 3, team: 9, team name: 7, type: 10, and year: 1 but only correctly identified it as city 53 times. 53. Here the confusion matrix is extremely helpful in understanding where and how misclassifications happen in order to improve the model later on.

5 Conclusion

Given the time restraint and that this is an individual project a 51% accuracy on a 78 way multi-classification problem appears to be good. In comparison to the related approaches I discussed,

these results, however, do not compare, but they provide a good base to continue later work. LiveRamp recieved an overall accuracy of 95% while Sherlock received a support-weighted F1-score of 89%, and SATO receives a support-weighed F1-score of 92.5%. All of these projects were conducted by experienced teams over a longer time period than I have been allotted and therefore it a given that their accuracy and F1-scores would be higher. However, there are several ways to improve this project going forward to improve results.

A few approaches for future improvement:

5.1 Larger Sample Size

One way this methodology could possibly improved would be to run it on a larger sample size. Sherlock runs their model on over 600,000 samples while our model is only run on 300,000. The reason ours is decreased is because of time restrictions and computer capability. When running the model on over 400,000 samples, after a few hours of running the kernel dies and cannot continue the process. While inconvenient, this problem was easily solved by decreasing the data sample sizes, however, as known with any machine learning model, larger training sizes tend to contribute to better results.

5.2 More Features

Another approach would be to add features to the model. Our model ran on 41 different features as mentioned above, however LiveRamp and Sherlocks ran on over 1000 features each. More features for this approach also poses the issue of time restriction and machine capabilities because while the model took hours to create and run on 41 features, were there even more features, specifically if it were run on the 1000 that both implementations took, it would be incapable of running on my machine.

5.3 Other encoding methods

While I had selected and created the features, there are many other encoding methods that exist in machine learning. One mentioned above was One Hot Encoding and another is Bag of Characters.

Overall, this project provided better results than I had expected and much better than the starting program which predicted labels on data sample lists instead of individually (this model

only correctly identified 2% of column labels) but that does not mean that there isn't further progress to be made.

6 Code

Click here to link to google drive where you can download the Code and Raw Data

References

[1] Ketch. [Online]. Available: https://www.ketch.com/

[2] LiveRamp, "Using machine learning to auto detect column types in customer files," Medium, 2018, last accessed 16 December 2022. [Online]. Available: https://medium.com/liveramp-engineering/using-machine-learning-to-auto-detect-column-types-in-customer-files-80413c976a1e

[3] H. Patel, "What is feature engineering — importance, tools and techniques for machine learning," Medium, August 2021.

[4] "Feature engineering," HEAVY.AI. [Online]. Available: https://www.heavy.ai/technical-glossary/feature-engineering

[5] J. Brownlee, "How to one hot encode sequence data in python," Long Short-Term Memory Networks, July 2017. [Online]. Available: https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/#:~:text=A%20one%20hot%20encoding%20is,is%20marked%20with%20a%201.

[6] Jeet, "One hot encoding of text data in natural language processing," Medium, August 2020. [Online]. Available: https://medium.com/analytics-vidhya/one-hot-encoding-of-text-data-in-natural-language-processing-2242fefb2148

[7] M. Hulsebos, K. Hu, M. Bakker, E. Zgraggen, A. Satyanarayan, T. Kraska, c. Demiralp, and C. Hidalgo, "Sherlock: A deep learning approach to semantic data type detection," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2019.

[8] J. Brownlee, "4 types of classification tasks in machine learning," August 2020. [Online]. Available: https://machinelearningmastery.com/types-of-classification-in-machine-learning/

[9] a. Z. I. Kotsiantis, Sotiris and P. Pinetalas, *Artificial Intelligence Review*. [Online]. Available: https://www.researchgate.net/profile/P-Pintelas/publication/226525180_Machine_learning_A_review_of_classification_and_combining_techniques/links/0fcfd5119227feb83c000000/Machine-learning-A-review-of-classification-and-combining-techniques.pdf

[10] D. K, "Top 5 advantages and disadvantages of decision tree algorithm," May 2019. [Online]. Available: https://dhirajkumarblog.medium.com/top-5-advantages-and-disadvantages-of-decision-tree-algorithm-428ebd199d9a#:~:text=Compared%20to%20other%20algorithms%20decision,scaling%20of%20data%20as%20well.

[11] V. Zwass, *Encyclopedia Britannica*. [Online]. Available: https://www.britannica.com/technology/neural-network

[12] S. Rawat, "Advantages and disadvantages of neural networks," July 2022. [Online]. Available: https://www.analyticssteps.com/blogs/advantages-and-disadvantages-neural-networks

[13] D. Zhang, M. Hulsebos, Y. Suhara, c. Demiralp, J. Li, and W.-C. Tan, "Sato: Contextual semantic type detection in tables," *Proc. VLDB Endow.*, vol. 13, no. 12, p. 1835–1848, sep 2020. [Online]. Available: https://doi.org/10.14778/3407790.3407793

[14] K. Hu, N. Gaikwad, M. Bakker, M. Hulsebos, E. Zgraggen, C. Hidalgo, T. Kraska, G. Li, A. Satyanarayan, and Ç. Demiralp, "Viznet: Towards a large-scale visualization learning and benchmarking repository," in *Proceedings of the 2019 Conference on Human Factors in Computing Systems (CHI)*. ACM, 2019.

[15] J. Brownlee, "How to calculate precision, recall, and f-measure for imbalanced classification," January.