Feature Type Inference Capstone Tech Report

Tanveer Mittal
Halıcıoğlu Data Science Institute
La Jolla, California
tamittal@ucsd.edu

Andrew Shen
Halıcıoğlu Data Science Institute
La Jolla, California
anshen@ucsd.edu

1 INTRODUCTION

The first step AutoML software must take after loading in the data is to identify the feature types of individual columns in input data. This information then allows the software to understand the data and then preprocess it to allow machine learning algorithms to run on it.

Project Sortinghat of the ADA lab at UCSD frames this task of Feature Type Inference as a machine learning multiclass classification problem. Machine learning models defined in the original SortingHat feature type inference paper[4] use 3 sets of features as input.

(1) The name of the given column
(2) 5 not null sample values
(3) Descriptive numeric statistics computed from the given column

The textual features are easy to access, however the descriptive statistics previous models rely on require a full pass through the data which make preprocessing less scalable. Our goal is to produce models that may rely less on these statistics by better leveraging the textual features. As an extension of Project SortingHat, we experimented with deep learning transformer models and varying the sample sizes used by random forest models.

1.1 Previous Work

Project SortingHat produced the ML Data Prep Zoo which is a collection of publicly available datasets. The zoo also includes all the precomputed features defined above as labeled benchmark data for feature type inference. This data plays a role in this space similiar to ImageNet in computer vision allowing the benchmarking of existing tools on this task. For our investigation, we have used the original data from the ML Data Prep Zoo; both the raw csv's of data as well as the benchmark labeled data to train and test our models with.

The experiments observed in the original SortingHat paper produced models that outperform the accuracy of existing tools such as AWS's AutoGluon, Google's Tensorflow Data Validation, the Pandas python library, and more. The single best model produced by this paper was a random forest model that uses the column name and descriptive statistics that yielded an Accuracy of 0.9265.

2 METHODS

2.1 Transformer Models

To answer the first question, we applied deep learning transformer models to feature type inference. These transformers generate contextualized embeddings for words present in the column name and sample values of a column by analyzing them at a character level. We believe transformer models are able to perform well on feature type inference because of their ability to contextualize inputs. Because transformers are able to generate contextualized word embeddings this allows our model to better leverage the column names and sample values in context to each other. We experimented with the Bidirectional Encoding and Representation Transformer(BERT)[1] model pretrained by Google to generate embeddings and processed them using a Convolution Neural Network to produce classifications.

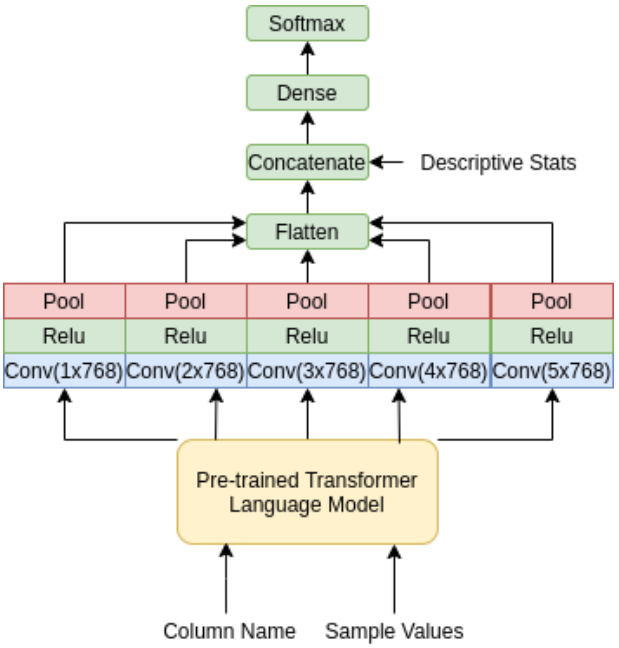

Figure 1: Diagram of transformer and full CNN architecture

To preprocess the column names and samples values we concatenated them and used separation([SEP]) tokens between them. These single strings are then tokenized using the HuggingFace transformers library. Our model architecture can be seen in Figure 12. The transformer receives the text and outputs a sequence of embeddings of size 768. We then use a convolution neural network architecture inspired from a paper that uses BERT with a CNN for offensive speech classification[3]. In this original model, the sequence of embeddings is fed into 5 separate convolutions that are processed with pooling in parallel. The intuition behind this operation is that the different convolution blocks can analyze different types of ngrams present in our data. Theoretically this model is able analyze individually words, bigrams, trigams, and more. The output of these operations are then concatenated and flattened. This vector is then concatenated with the descriptive statistics and fed into a softmax dense layer to output a classification. This architecture

Table 1: Convolution filter ablation experiments. The convolution blocks are represented as a list of integers $[x_1, x_2, ..., x_n]$ where x_i represents the a convolution block with a filter dimension of x_i x 768. These experiments are run using BERT with a fixed kernel size of 256. The change in accuracies from removing a single filter compared to the full model using all 5 filters are reported.

Convolution Filter Sizes	[1, 2, 3, 4, 5]	[2, 3, 4, 5]	[1, 3, 4, 5]	[1, 2, 4, 5]	[1, 2, 3, 5]	[1, 2, 3, 4]
Validation Accuracy	0.931	0.924	0.924	0.926	0.928	**0.930**
Testing Accuracy	0.930	0.928	0.931	0.929	**0.934**	0.932
Delta % Testing Accuracy	0.00%	-0.15%	+0.15%	-0.05%	**+0.35%**	+0.25%

Table 2: Additional Convolution Filter Ablation Experiments. These experiments are run using BERT with a fixed kernel size of 256. The change in accuracies from removing a single filter compared to the full model using the best 4 filters found in table 1 are reported.

Convolution Filter Sizes	[1, 2, 3, 5]	[1, 2, 3]	[1, 2, 5]	[1, 3, 5]	[2, 3, 5]
Valid Accuracy	0.928	0.927	0.931	0.930	**0.931**
Test Accuracy	**0.934**	0.926	0.930	0.929	0.931
Delta % Testing Accuracy	0.00%	-0.71%	-0.35%	-0.45%	-0.25%

has a lot of hyperparemeters and architeture that can be changed so we decided to run a series of experiments to identify the best combination of convolution blocks and kernel size our CNN can use for this task.

Table 3: Accuracies of the BERT transformer model with varying kernel sizes for the CNN. All these models use the best 4 filters reported in Table 1.

Kernel Size	64	128	256	384	512
Valid Accuracy	0.923	0.929	**0.931**	0.929	0.930
Test Accuracy	0.928	0.930	**0.934**	0.930	0.933

We first decided to run ablation experiment where we remove individual convolution blocks and then observe the difference in accuracy from the original model using 5 convolution blocks. From tables 1, we observe that the best performing combination of convolution blocks is [1, 2, 3, 5]. Once we identified this we also ran the additional ablation in table 2 to see if we could increase accuracy any further from removing certain blocks but we did not observe any more improvements. In addition to identifying the best architecture of convolution blocks for our model, these experiments have been helpful in identifying which filter dimensions perform best on our data. From table 1, we could observe that removing filters 1 & 3 decreased model performance where as every other filter's removal increased accuracy. This suggests that our model finds the most value from analyzing individual words and trigrams. Alongside these experiments and simple grid search of kernel sizes we determine that our best architecture uses the combination of blocks [1, 2, 3, 5] and a kernel size of 256 for all convolution blocks.

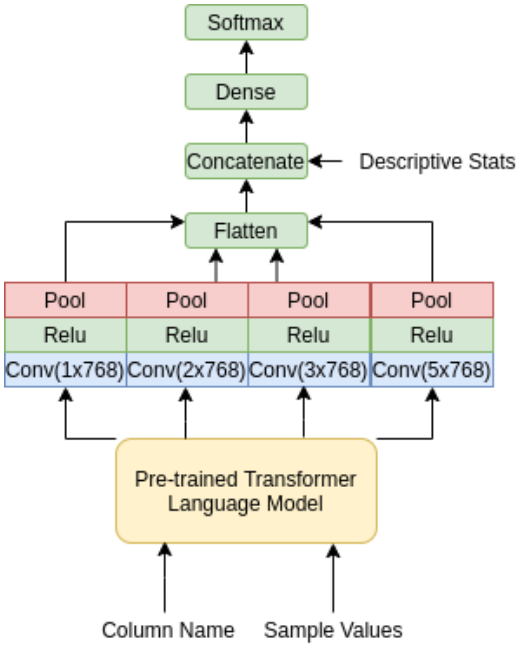

Figure 2: Diagram of transformer and best CNN architecture

In table 4 we can observe the importance of the different features used with our BERT CNN model. From the results we can see that BERT is significantly better at using column names and sample values than any of the models from the original SortingHat paper[4]. The sample values seem to be the single most important feature for our model. This supports our hypothesis by suggesting that BERT is effectively leveraging sample values and column names in the context of each other. Our best model from this experiment is still the one using all our features, however unlike the results from original SortingHat paper, the accuracy improvement of using the

Table 4: 9 class accuracies of our best model architecture using different feature sets. These experiments were run using the best BERT CNN model architecture found in Tables 1-3.

Feature Set	X_{name}	$X_{samples}$	X_{name}, X_{stats}	$X_{samples}, X_{stats}$	$X_{name}, X_{samples}$	$X_{name}, X_{samples}, X_{stats}$
Validation Accuracy	0.815	0.866	0.837	0.878	0.925	**0.928**
Testing Accuracy	0.813	0.858	0.841	0.871	0.929	**0.934**

descriptive statistics is very small. In fact our model that only uses column names and sample values still outperforms the best random forest model; this also scales better as it does not require a full pass through the data to generate descriptive statistics. As a result, we have decided to release these 2 models.

2.2 Sampling for Descriptive Statistics

To answer the last 2 questions, we would like to do an investigative analysis on summary statistics for feature type inference. We would like to propose the idea of generating summary statistics from a sample of input data to reduce runtime, especially on larger datasets. To do so, we would like to propose a truncated set of summary statistics as seen in Table 3 that do not require the entire dataset to compute and that we believe could be estimated from sampling.Using this truncated set of summary statistics, we would like to then compare our results with results from the original Sortinghat Feature Type Inference paper [4].

2.2.1 ***Data Sampling on Model Accuracy.*** To test the effects of taking a small sample of data instead of the entire dataset, we calculate the 9 class accuracy, precision, recall, and F1-score across sample sizes from 1-5 and 10. As seen in Table 4, numeric, categorical, and non-generalizable feature type accuracy improves with each additional sample. Datetime, sentence, url, embedded-number, list, and context-specific feature types did not see much benefit from taking additional samples of the data. This makes sense as these feature types follow a strict format and 1 sample is enough to match the format to the feature type. Confusion matrices for 1, 5, and 10 sample models can be found in the appendix.

2.2.2 ***Data Sampling on Model Runtime.*** To test the effects of increasing the sample size on the runtime of the model, we recorded the runtimes of the model across our 1 through 5 and 10 sample size benchmarks. All runtimes shown are the averages across 3 iterations. The times shown in Figure 2 are the recorded runtimes of training a Random Forest model, then running a test set on the trained model. For both the train and test set, the summary statistics and samples were generated prior to the recorded times.

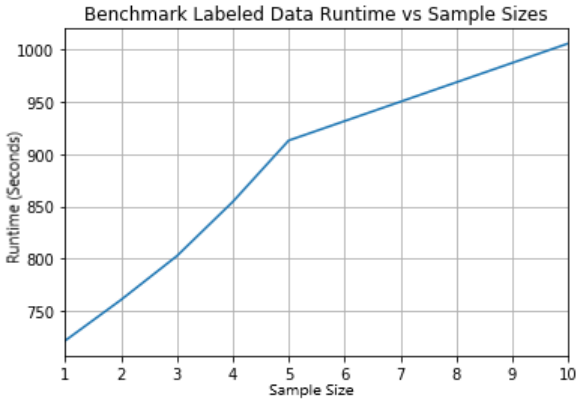

Figure 3: Benchmark Labeled Data Runtimes

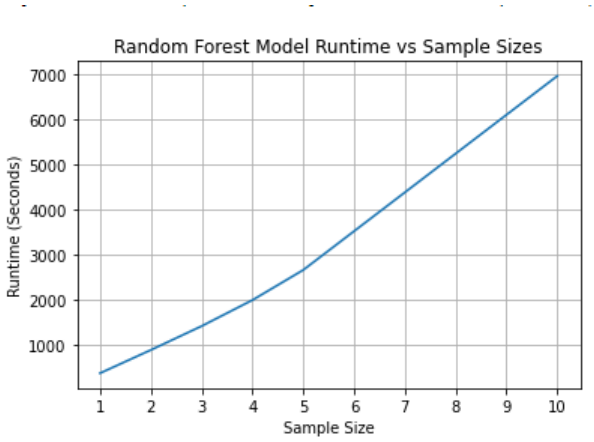

Figure 4: Random Forest Model Runtime vs Sample Size

Table 5: Class specific Accuracy, Precision, and Recall on testing data of our best BERT CNN model.

Data Type	numeric	categorical	datetime	sentence	url	embedded-number	list	not-generalizable	context-specific
Accuracy	0.983	0.972	1.0	0.986	0.999	0.997	0.994	0.968	0.967
Precision	0.959	0.935	1.0	0.849	0.969	0.989	0.960	0.848	0.870
Recall	0.996	0.943	1.0	0.859	0.969	0.949	0.842	0.856	0.762

Table 6: Accuracy, Precision, Recall, and F1-Score Across all 9 Classes Using Increasing Sample Sizes

Feature Type	Sample Size / Metric	1	2	3	4	5	10
numeric	accuracy	0.96	0.96	0.96	0.96	0.96	0.96
	precision	0.92	0.92	0.93	0.93	0.93	0.92
	recall	0.97	0.97	0.97	0.98	0.97	0.97
	f1-score	0.94	0.94	0.95	0.95	0.95	0.95
categorical	accuracy	0.92	0.92	0.92	0.92	0.92	0.92
	precision	0.79	0.80	0.80	0.81	0.80	0.80
	recall	0.88	0.87	0.89	0.88	0.88	0.89
	f1-score	0.83	0.83	0.84	0.84	0.84	0.84
datetime	accuracy	1.00	1.00	1.00	1.00	1.00	1.00
	precision	0.97	0.97	0.98	0.98	0.98	0.98
	recall	0.97	0.99	0.99	0.98	0.99	0.97
	f1-score	0.97	0.98	0.98	0.98	0.99	0.98
sentence	accuracy	0.98	0.98	0.98	0.98	0.98	0.99
	precision	0.82	0.81	0.83	0.85	0.82	0.87
	recall	0.76	0.79	0.78	0.80	0.82	0.83
	f1-score	0.79	0.80	0.81	0.82	0.82	0.85
url	accuracy	1.00	1.00	1.00	1.00	1.00	1.00
	precision	0.94	0.93	0.93	0.95	0.95	1.00
	recall	0.91	0.84	0.94	0.95	0.86	0.88
	f1-score	0.92	0.89	0.93	0.95	0.90	0.93
embedded-number	accuracy	0.99	0.99	0.99	0.99	0.99	0.99
	precision	0.94	0.96	0.95	0.94	0.95	0.94
	recall	0.85	0.87	0.89	0.88	0.88	0.86
	f1-score	0.89	0.91	0.92	0.91	0.92	0.90
list	accuracy	0.99	0.99	0.99	0.99	0.99	0.99
	precision	0.96	0.98	0.95	0.98	0.98	0.98
	recall	0.75	0.75	0.74	0.76	0.75	0.72
	f1-score	0.84	0.85	0.83	0.86	0.85	0.83
not-generalizable	accuracy	0.93	0.93	0.94	0.94	0.94	0.94
	precision	0.73	0.71	0.73	0.73	0.74	0.77
	recall	0.63	0.64	0.66	0.66	0.64	0.70
	f1-score	0.67	0.67	0.69	0.69	0.68	0.73
context-specific	accuracy	0.95	0.95	0.95	0.95	0.95	0.95
	precision	0.76	0.75	0.78	0.77	0.79	0.79
	recall	0.63	0.62	0.63	0.64	0.66	0.62
	f1-score	0.69	0.68	0.70	0.70	0.72	0.69

3 CONCLUSION

From the experiments that we have ran, we can see that transformer models are very effective at Feature Type Inference. Our models now outperform all existing tools and models benchmarked against the ML Data Prep Zoo. Furthermore we can see there is great potential in applying more state of the art natural language processing techniques to this task to increase performance and rely less on descriptive statistics to produce scalable models. This project produced great results but we have only scratched the surface of what transformers could be capable of in this space. Further work in this area can involve experimenting with more CNN architectures than the one we defined and trying other state of the art language models such as RoBERTa[2] and XLNet[5]. We will be releasing the best models produced by our experiments that use and do not use descriptive statistics to Torch Hub to allow for easy integration of our models into AutoML platforms or other applications of automated data preparation.

REFERENCES

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. (October 2018). https://arxiv.org/abs/1810.04805

[2] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *International Committee for Computational Linguistics* (July 2019). https://arxiv.org/abs/1907.11692

[3] Ali Safaya, Moutasem Abdullatif, and Deniz Yuret. 2020. KUISAIL at SemEval-2020 Task 12: BERT-CNN for Offensive Speech Identification in Social Media. *International Committee for Computational Linguistics* (2020). https://aclanthology.org/2020.semeval-1.271.pdf

[4] Vraj Shah, Jonathan Lacanlale, Premanand Kumar, Kevin Yang, and Arun Kumar. 2021. Towards Benchmarking Feature Type Inference for AutoML Platforms. *ACM SIGMOD 2021* (June 2021). https://adalabucsd.github.io/papers/TR_2021_SortingHat.pdf

[5] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. (June 2019). https://arxiv.org/abs/1906.08237

4 APPENDIX

Table 7: Random Forest 1 Sample Confusion Matrix

	numeric	categorical	datetime	sentence	url	embedded-number	list	not-generalizable	context-specific
numeric	301	0	0	0	0	0	0	406	0
categorical	0	30	3	1	0	1	10	412	0
datetime	0	0	114	0	0	0	3	24	0
sentence	0	0	0	7	0	0	15	70	0
url	0	0	0	0	24	0	1	7	0
embedded-number	0	0	1	0	0	64	7	27	0
list	0	0	0	0	0	2	47	8	0
not-generalizable	1	2	1	0	1	0	1	209	0
context-specific	7	0	0	0	0	0	5	169	4

Table 8: Random Forest 5 Sample Confusion Matrix

	numeric	categorical	datetime	sentence	url	embedded-number	list	not-generalizable	context-specific
numeric	604	9	1	0	0	0	0	93	0
categorical	15	269	1	16	0	1	2	150	3
datetime	0	2	130	0	0	2	0	7	0
sentence	0	2	0	69	0	0	1	20	0
url	0	0	0	1	27	0	0	4	0
embedded-number	0	6	1	0	0	71	1	20	0
list	1	2	0	2	0	3	7	41	1
not-generalizable	4	20	1	1	1	1	0	186	1
context-specific	44	20	1	1	0	2	2	89	26

Table 9: Random Forest 10 Sample Confusion Matrix

	numeric	categorical	datetime	sentence	url	embedded-number	list	not-generalizable	context-specific
numeric	618	13	1	0	0	0	0	75	0
categorical	23	298	0	16	0	2	2	112	4
datetime	0	1	121	0	0	2	0	17	0
sentence	0	1	0	68	0	1	1	21	0
url	0	0	1	1	26	0	0	4	0
embedded-number	0	6	1	0	0	71	1	20	0
list	1	2	0	2	0	5	4	41	2
not-generalizable	2	16	2	1	0	1	0	192	1
context-specific	56	22	1	3	0	2	0	71	30