Alexandra DeGrandchamp
DSC 478 Final Project Submission
November 21, 2023

Towards Heuristic Handling of Free-Text Data

Introduction

Procurement analytics is a field that analyzes a corporation's purchasing data – the materials and services required to create the company's product – to optimize supply chains and create savings that can be realized as profit. The underpinnings of procurement analytics as a field relies on high-quality classifications to drive insight. Insights are driven by grouping and slicing data across a variety of metrics, and those groupings are often buoyed first by classifications (in this context, answering the question "what is this product?") of purchases. From expensive raw materials like metals to precision-crafted inputs like nuts and bolts to indirect services such as consulting and marketing, making sense of the immense volume of rows generated by a corporation typically requires strong master data management.

Some purchases, such as materials and inputs, can be more easily classified by relying on a hierarchical materials master lookup table. Others, such as indirect services and expense report data, can be much more tedious. Combine this tedium with a complex (typically four-level) classification hierarchy, and classification is often a labor-intensive process riddled with errors.

Project Landscape and Dataset

For this project, I was interested in data sets with a few useful categorical variables combined with free-text data. In the procurement space, data with this basic structure, such as credit card data, expense report system data, or purchasing data from service-based companies without a lot of raw material input into their product represents a potential pitfall in gaining valuable insights from procurement analytics. When most of the classifiable information is stored in free-text fields, heuristic decision trees built on categories of data points tend to fail. When classification must take place row-by-row, classification often doesn't take place at all, or produces poor-quality or high-level classifications.

However, with large language models and other natural language processing techniques readily available, more companies are turning to machine learning and AI to make sense of their poorly structured data. I was interested in combining several machine learning techniques to begin to model an accessible methodology for processing and classifying data mostly comprised of free-text fields.

Kaggle's "Large Purchases by the State of California"[1] dataset provided the perfect introductory dataset for this endeavor. The data set holds purchase order data for some 200,000 unique purchase orders comprised of approximately 346,000 rows of purchases made by government departments between 2012 and 2015. The set contains 31 total columns for consideration. Eight of these columns are classification columns – both the codes and the labels for the UNSPSC hierarchy (a brief description of the hierarchy appears below). I identified the following fields as pertinent to classifying each line to the four levels of the hierarchy:

FIELD NAME	BRIEF DESCRIPTION[2]
ACQUSITION TYPE	Type of Acquisition. Five possible values: Non-IT Goods, Non-IT services, IT Goods, IT Services, and IT Telecommunications
SUB-ACQUISTION TYPE	Depends on the type of acquisition type used. Data dictionary provides dependencies
DEPARTMENT NAME	Name of purchasing department. This field is normalized, not free text
ITEM NAME	Name of items being purchased
ITEM DESCRIPTION	Description of items being purchased

The remaining fields (for example, supplier zip code or acquisition method) are valuable for analyzing classified data, but do not point to the classifications themselves. Supplier Name was provided, but resulted in some 25,000 unique values. As this data set was already large and would need a significant number of features to accommodate dummified clustered data, I opted to exclude Supplier Name from the analysis. See Further Work below for a discussion of how Supplier Name may be incorporated into future development of this model.

UNSPSC Hierarchy

The United Nations Standard Products and Services Code (UNSPSC) provides a four-level classification hierarchy for product and services purchasing. It is considered an internationally recognized standard for classification, and while many corporations adopt classification hierarchies that reflect their unique business needs, UNSPSC is a reputable system for classification.

The hierarchy[3] used in this data set has four levels (UNSPSC can technically go to five levels):

1. Segment, which is the most general level, and coded as Level 0 in this model;
2. Family, which subdivides segment. This is coded as Level 1;

[1] Dane, S. (2017, August 29). *Large purchases by the state of CA*. Kaggle. https://www.kaggle.com/datasets/sohier/large-purchases-by-the-state-of-ca

[2] Taken from the DGS PURCHASING DATA DICTIONARY that accompanies the dataset; also available at the link above

[3] Wikimedia Foundation. (2023, October 20). *UNSPSC*. Wikipedia. https://en.wikipedia.org/wiki/UNSPSC

3. Class, which begins to discuss the specifics of the product or service (Level 2 in the model); and finally
4. Commodity, the deepest level. This is Level 3 in the model.

Each row is simultaneously classified to all four levels of the hierarchy when possible. The hierarchy is dependent on levels above; a Level 1 classification must be found only under one possible Level 0, etc.

Level	Code	Description
Segment	**44**000000	Office Equipment, Accessories and Supplies
Family	44**12**0000	Office supplies
Class	4412**19**00	Ink and lead refills
Commodity	441219**02**	Lead refills

Figure 1: A sample UNSPSC classification (courtesy cited Wikipedia article).

Data Cleaning

I benefitted from a largely clean data set; from 346,000 rows I ended up with just fewer than 300,000 complete cases (complete cases defined as non-null values in Acquisition Type, Item Name, Item Description, and Commodity Title). Sub-Acquisition type was often null, so I chose to render those values as the categorical 'N/A' instead of NaN. I also filtered rows to eliminate very small groups in the final model. For example, I limited rows to those belonging to a Department Name with at least 20 instances, and each Commodity was also required to have at least 20 instances. Finally, to avoid narrow scope from the top of the hierarchy, I limited rows to Segments containing at least 1,000 instances. Finally, I stripped Total Price of its '$' character to render the variable numeric.

Model Development

The final model combines several machine learning techniques that can be applied to datasets similarly structured to this example. I tuned text algorithms, tf-idf vectorization, singular value decomposition, k-means clustering, and a decision tree classifier on the final data set.

Text Algorithms and TF-IDF Vectorization

Since so few columns were available for classification with this dataset, I spent most of the model development time fine-tuning the text processing algorithms. The intent was to turn messy, free-text data into usable grouped clusters suitable for a decision tree or other heuristic model. I chose to join both Item Name and Item Description into one concatenated column since many rows featured identical information in both columns. I felt the repetition of this free-text would be beneficial in the tf-idf vectorization described in further detail below.

The text algorithms were developed from the Natural Language Toolkit[4] algorithms for Python, primarily tokenization/detokenization and the Snowball Stemmer. For each row of concatenated text data, I stripped numeric values using regex (I ignored punctuation since the tf-idf vectorizer described below also ignores punctuation), tokenized each word, stemmed the word according to the English stemmer, and detokenized the words to create a single string. Stemming was crucial to feature reduction with this size data set. The algorithms from NLTK are effective as-is for an initial model, so no major tuning was performed at this stage. The stemming algorithm also converts all text to lower case.

I used scikit[5]-learn's TfidfVectorizer to compute and normalize term-document frequencies, with each row comprising a "document" for this model's purposes. I set lower to false when creating the matrix, as the NLTK algorithms had already performed this normalization. I limited terms to three+ letters, and employed the algorithm's basic stop words exclusion list as an additional feature reduction mechanism, despite the known limitations to stop words functionality.[6] The resulting tf-idf matrix featured some 43,000 terms, which is quite a lot, but a significant improvement from using free-text data as is (where the number of features would be almost indistinguishable from the number of rows in the data set), and an improvement from an early tuning without reliance on NLTK, which boasted over 70,000 features.

Singular Value Decomposition and K-Means Clustering

Still, 43,000 features are too many for further algorithm development. I chose to employ scikit-learn's TruncatedSVD algorithm to reduce the number of components. When used in conjunction with the TfidfVectorizer, this method is called Latent Semantic Analysis and is a common technique to tease out important word groupings in a sea of text. I needed to balance using a meaningful subset of components from the tf-idf vectorizer while also keeping mindful of run times. I chose potential components between 100 and 1500 (1500 representing some 75% of the variance in the data set) to test.

I performed a k-means clustering on the SVD datasets at various component levels. For each component level (100, 200, 300, 500, 1000, and 1500), I tested between 10% and 100% of the components in terms of k, divided evenly in 10 splits. I then calculated the overall silhouette score of the model.

Appendix 1 contains the results of this tuning exercise, with SVD components forming the vertical axis and tested k clusters forming the horizontal. I was attempting to maximize silhouette scores, and stopped tuning components once I observed consistently smaller

[4] Bird, Steven, Edward Loper, and Ewan Klein (2009), *Natural Language Processing with Python*. O'Reilly Media Inc.
[5] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.
[6] J. Nothman, H. Qin and R. Yurchak (2018). "Stop Word Lists in Free Open-source Software Packages". In *Proc. Workshop for NLP Open Source Software*.

numbers by increasing both components and k. There were clear diminishing returns as explained variance increased the complexity of the model.

Similar results were observed for k values between 300-800 and SVD components between 300-1000. Ultimately, I chose 500 SVD components forming 350 clusters as a balance between model complexity and completeness.

Though the resulting silhouette score – 0.27 – is considered low for a clustering model on its own, I argue that capturing over 50% of the variance in the free-text data is a significant contributor to an eventual heuristic model. As the free-text data won't stand alone in the final model, the silhouette value does not need to reach levels closer to 1. The Further Work section will further discuss how silhouette scores may be improved in further iterations of the model.

The resulting cluster numbers were cleansed to reflect a concatenated string of the top three stemmed terms found in the cluster. This cluster label replaced the Item Name and Item Description columns in the final data set.

Decision Tree Classifier

The resulting clustered dataset was converted to 480 dummy variables (though there were 350 clusters, the remaining categorical variables all had several values). I used scikit-learn's DecisionTreeClassifier model to create a decision tree.

With a large data set and highly dimensional data, I conducted three separate tunings to consider various inputs in a more runtime-friendly manner. I created two control models using default parameters (one with the clustered set and one with the categorical variables without any text data, mirroring how this type of data structure is often treated in practice in procurement analytics). I then varied the following parameters before settling on a final model based on the evaluation criteria:

PARAMETER NAME	VALUES TESTED	RUNS TESTED
CRITERION	Entropy, Gini	All runs
MAX DEPTH	[1,5,10,15,20] [20,28,37,46,55,64,73,82,91,100]	1 & 2
MIN SAMPLES (LEAF)	[1,8,15,22,30]	1
MIN SAMPLES (SPLIT)	[2,6,11,15,20]	1
MAX FEATURES	[0.05, 0.156, 0.261, 0.367, 0.472, 0.578, 0.683, 0.789, 0.894, 1]	2 & 3

Model Evaluation

Procurement data is unique because it is associated with actual spend values, and the classification hierarchy deployed to make sense of the data must cover all spend exhaustively. That means the classifications are heavily imbalanced – the Accounting Department's monthly paper clips order need to be classified in the same taxonomy as the new $2M telescope for the

R&D lab. Furthermore, classification hierarchies can be incredibly granular – while some industries may use broad lowest-level categories for inconsequential purchases, others, such as non-profits, universities, and, yes, governments, can develop incredibly granular hierarchies. Therefore, while accuracy as a measure of accurately classified rows was considered when developing the final model, I also evaluated accuracy in terms of percent of overall spend classified accurately.

As mentioned in the Decision Tree section above, two control models were developed to compare the final model. The (rounded) results of the three models are described in the tables below:

No Text Model

LEVEL	% ACCURATE ROWS	% SPEND ACCURATE
0	45%	88%
1	30%	84%
2	26%	83%
3	22%	82%

Default Model Parameters

LEVEL	% ACCURATE ROWS	% SPEND ACCURATE
0	60%	89%
1	51%	86%
2	46%	85%
3	40%	85%

Tuned Model Parameters

LEVEL	% ACCURATE ROWS	% SPEND ACCURATE
0	60%	88%
1	51%	86%
2	46%	84%
3	39%	84%

The text processing model significantly improved the results of the decision tree classifier, indicating that the data available in the free text fields has value and should be further explored to improve model accuracy. Even at the most granular level, results jumped from 22% to 40% accurate for number of rows.

Notably, the tuned model did not have significant improvement over the default model, suggesting that improvements need to come from sources outside of the tree. Potential improvements will be discussed in Further Work below.

All models showed remarkable accuracy in terms of the spend. This suggests that the model is not handling tail spend (the 10-15% of low-dollar purchases made by corporations) with precision, and likely, this is disproportionately impacting some of the classification's hierarchy more than others. There are also likely several categories with few rows but high spend dollars driving up the spend % accuracy. Measuring spend value accuracy is usually the top priority for procurement analytics teams, but as demonstrated above, overreliance on this metric can mask significant accuracy issues.

Unfortunately, the time constraints of this project did not lend to further evaluation criteria for the model, such as precision and recall of the final model versus the controls. For imbalanced data sets, these are critical checks and will be considered alongside the recommendations in Further Work for future model improvements.

Further Work

My intent for this project was to propose a preliminary algorithm for considering free-text data in data sets that are mostly categorical, limited in the ability of the categorical variables to result in accurate classifications, and rich in free-text data that is inherently unusable in its raw state. I believe this framework provides rich opportunities to continue to develop this model to successfully capture the hidden patterns in free-text data that point to accurate classifications in complex hierarchies.

The first avenue for future work stems from the text processing part of the model. For simplicity, this model concatenated two similar text fields into one. Further work could explore if leaving these fields separate, and performing separate tf-idf, SVD, and clustering mechanisms on them as individuals would help the decision tree final model develop more nuance in its predictions.

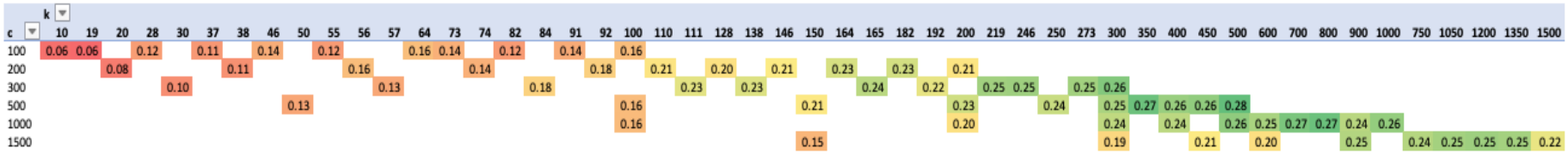
A second opportunity is with the omitted Supplier variable. While this project focused on the more readily known material free-text data, supplier classification is a field rife with possibilities. Immature data systems such as expense report platforms or credit card data feeds often broadly classify suppliers, so a supplier like Amazon, where one could feasibly buy anything, is associated with the immutable category of "Bookstores". Supplier parenting mechanisms are often fallible as well, with complex business relationships reduced to simple single-value lookup fields that need to be constantly maintained to reflect current business conglomerates. Perhaps using portions or even the entirety of the text processing module on the Supplier field, normalized but to some 25,000 unique values, would lend itself to better classification in the final module.

In addition to free text data modifications, the clustering itself would benefit from more time spent on its tuning. K-Means was selected for its speed and ease of use. However, this highly dimensional data set is likely a far cry from the spherical forms favored by K-Means, and thus alternative clustering algorithms, such as DBSCAN, may be used in future iterations. I should note that an attempt was made to test a DBSCAN model, but I personally lacked the computing

power to run the model to the tuning parameters I was looking for, so a smaller dataset may be better suited for this modification.

Finally, the final model deployed was a decision tree algorithm. This was intentional, as the typical end users of classified procurement data are not necessarily technically skilled in advanced model building. Decision trees are machine and human readable; they are easy to build and easy to understand. However, with this sparse data set, a more sophisticated final model, such as a random forest or neural network, may be better suited to the hidden patterns in the data.

Appendix 1



c \ k	10	19	20	28	30	37	38	46	50	55	56	57	64	73	74	82	84	91	92	100	110	111	128	138	146	150	164	165	182	192	200	219	246	250	273	300	350	400	450	500	600	700	800	900	1000	750	1050	1200	1350	1500
100	0.06	0.06		0.12		0.11		0.14		0.12			0.16	0.14		0.12		0.14		0.16																																
200			0.08				0.11				0.16				0.14				0.18		0.21		0.20		0.21		0.23		0.23		0.21																				
300					0.10							0.13					0.18					0.23		0.23				0.24		0.22		0.25	0.25		0.25	0.26														
500									0.13											0.16						0.21					0.23			0.24		0.25	0.27	0.26	0.26	0.28										
1000																				0.16											0.20					0.24		0.24		0.26	0.25	0.27	0.27	0.24	0.26					
1500																										0.15										0.19			0.21		0.20			0.25		0.24	0.25	0.25	0.25	0.22

Appendix 2: Bibliography

Bird, Steven, Loper, Edward & Klein, Ewan (2009), *Natural Language Processing with Python*. O'Reilly Media Inc.

Dane, S. (2017, August 29). *Large purchases by the state of CA*. Kaggle. https://www.kaggle.com/datasets/sohier/large-purchases-by-the-state-of-ca

Nothman, J., Qin, H. & Yurchak, Y. (2018). "Stop Word Lists in Free Open-source Software Packages". In *Proc. Workshop for NLP Open Source Software*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Wikimedia Foundation. (2023, October 20). *UNSPSC*. Wikipedia. https://en.wikipedia.org/wiki/UNSPSC