PharmaSUG 2019 - Paper BP-079

Performing Analytics on Free-Text Data Fields: A Programer's Wurst Nitemare

Michael S. Rimler, GlaxoSmithKline;
Matt Pitlyk, 1904labs

ABSTRACT

Case report forms (CRFs) often contain free-text fields for collecting patient information when standard responses do not apply, e.g. 'Other specify' or 'Reason for ...'. Furthermore, the analysis plan may require analyses to be performed on this non-standardized information. Free-text fields are renowned for their difficulty to be programmatically incorporated into analyses due to human nature injecting spelling/grammatical errors or differences in languages across global sites. These fields also tend to be very difficult to monitor. Requesting sites to 'modify' content to support cleaner analyses is time consuming, even if the site is responsive and agreeable.

We compare methodologies for classifying a record into a binary response based on information collected via a free-text field. For example, identifying all collected concomitant medication records taken for Chronic Obstructive Pulmonary Disease (COPD) when 'Reason for Therapy' is collected via free-text. Methodologies include typical techniques in SAS (brute force string search and fuzzy matching) and machine learning (clustering and classification algorithms). The basis of methodology comparison includes (i) degree of code complexity, (ii) measures of inaccuracy such as precision and recall, and (iii) the maintenance burden of code along the study life-cycle over iterative data cuts.

In our problem, machine learning algorithms using logistic regression and random forest classifiers exhibit the lowest incidence of false negatives on the test data. The brute force technique also exhibits a low incidence of false negatives, but code maintenance with this method is arguably more burdensome than the two machine learning algorithms as new data is observed.

INTRODUCTION

Using pooled and anonymized clinical data, we investigate a specific example of identifying concomitant medication records which are administered for chronic obstructive pulmonary disease (COPD) using 'Reason for Therapy', a free-text collection field on the CRF. Since most clinical programming (CP) departments report clinical trial results using SAS ® programming to generate analysis datasets, tables, listings, and figures, we consider typical techniques that may be implemented with a SAS programming infrastructure to classify records as COPD or not. As comparison, we also implement techniques available in the Python programming language, including *machine learning* (ML) algorithms.

A primary benefit of implementing a solution with SAS programming is that CP groups already possess the requisite skillsets. However, the code must be maintained after each new data cut to ensure that it executes properly. Within this paradigm, we discuss 4 methodologies:

- Importing a spreadsheet of flagged records
- SAS brute force string search
  - Low tech method
  - 100% accuracy method
- SAS fuzzy matching algorithms

Fuzzy matching algorithms are also available in Python, which we include as a comparison to similar methods in SAS. In addition, we apply ML algorithms available in Python, a common language used to perform such analyses. We apply *unsupervised* learning models to understand the data prior to applying *supervised* learning models for classification purposes. In summary, we discuss the following methodologies using Python:

- Python fuzzy matching algorithms
- Unsupervised learning
- Supervised learning
    - Logistic regression
    - Random forest

ML algorithms have been applied for parsing free-text fields within the clinical reporting before, for example, with respect to patient narratives [1]. Our problem differs from the patient narrative problem, however, in that our data has relatively fewer 'features'. A feature is a dimension of the data that can be leveraged to uncover insights into the data itself, analogous to covariates in a statistical model. Other examples of analyzing free-text fields with few features include:

- Classifying frequency of administration (CMFREQ) from 'Other, specify' fields
- Classifying RACE from an 'Other, specify' field
- Classifying lab specimen type (LBSPEC) from a comments field

As an example, suppose that the analysis requires identification of long-acting treatment regimens which should be identifiable simply through medication coding. However, a short-acting regimen which is administered with high frequency may, for the purposes of analysis, be considered long-acting. If, for example, 'high frequency' is considered at least 4 times a day, classifying free-text 'Other, specify' into more (or less) frequently than 4 times a day is programmatically challenging.

It is typical in the machine learning space, when training an algorithm, to choose parameter values which balances *precision* and *recall*. Precision is the percent of records flagged as COPD that are truly COPD. Recall is the percent of true COPD records that are flagged as COPD. You may recognize similarities to Type I error (false positives) and Type II error (false negatives). Also of interest is a third measure of performance, *accuracy*, which is the percent of records flagged correctly (true flagged as true, false flagged as false).

Another idiosyncratic characteristic of our problem, relative to other ML applications, centers around training the ML algorithms and the desired precision/recall balance. This balance is always determined by the context in which the classifier is being used. In our context, higher *recall* is preferred to higher *precision*, as it means that we have a higher incidence of correct COPD classification. Allowing for lower precision results in a wider net, i.e. flagging more non-COPD records incorrectly as COPD. We argue that this isn't as critical as failing to flag COPD as COPD. Given the small percentage of true COPD medications in the overall set, it is easier to unflag a record to 'non-COPD', than review a large set of non-flagged records and determine if they are truly COPD.

We could ensure 100% recall by flagging everything as COPD, but then a complete manual review of all records to 'unflag' non-COPD records would be required, mitigating any value of applying a classification methodology. Hence, a model which has higher *recall* relative to *precision* is preferred, but an incremental increase in recall may not be worth the incremental reduction of precision. The appropriate balance for a particular methodology is subjective, a decision we render in each case and present as part of our analysis. The decision is based on our understanding of the data and the sensitivity of the algorithms on performance on the training data from changes in underlying modelling parameters.

Finally, ML algorithms are not the magic bullet that removes the need for human intervention, particularly in pharmaceutical drug development. Any process applied to clinical data which contributes to the ultimate regulatory approval of an investigational product must be reproducible, validated, documented, and 100% accurate. Although it would be optimal to develop a classification algorithm that eliminates the need for human intervention when applied to new data, we contend some level of manual review will always be required to achieve 100% accurate classification. Even machine learning algorithms are imperfect classification methods but designed with the objective of minimizing the level of human intervention required to attain 100% accuracy.

However, a strength of ML algorithms, compared to standard SAS techniques, is that a well-trained ML algorithm generally requires minimal coding updates or retraining with every data cut. The initially trained model can be directly executed on new data with little expected deterioration of performance, assuming the data is sufficiently similar to the previous data. Significant changes in the input data may require a retraining of the algorithm to ensure it continues to perform at a high level, but we contend this is less impactful than the coding maintenance required with SAS implemented methods.

SOURCE DATA

The data used for the analysis leverages GSK's R&D Information Platform (RDIP), a platform which hosts many different data domains, among which is GSK's anonymized and pooled clinical trial data. At the time our data was extracted, the Concomitant Medications domain included over 700 individual studies and nearly 150,000 patients. The primary field of interest is concomitant medication indication (CMINDC), which is collected as a free-text field in the CRF.

We compare different methods for classifying the verbatim entries of this field. Our focus is on identifying medications administered for Chronic Obstructive Pulmonary Disease (COPD). Therefore, we subset the broader Concomitant Medications domain with four conditions:

1. Studies in the Respiratory therapeutic area
2. Studies with an investigational product indicated for COPD
3. Exclusion of routes of administration which are unlikely to be used for COPD medications (e.g. include routes such as oral, inhalation, or nasal)
4. Exclusion of redacted values of CMINDC in the anonymized data

Restrictions on data availability and the anonymization process limit the ability to leverage other fields typically collected with concomitant medications such as anatomical therapeutic chemical (ATC) and preferred term (PT). The third exclusion allows for reduction in the number of records that are not COPD with little risk to losing true COPD medications. The anonymization process underlying the RDIP Clinical Trials Domain also results in a large number of redacted entries for CMINDC (approximately 33% of the data meeting the other subset conditions).

After additional mild data cleaning, the resulting analysis dataset includes 212,079 unredacted records across 106 studies and 18,481 unique values of CMINDC. In preparation for the analyses, we then manually labeled each unique value as COPD or not. Although this process was reviewed, we identified a few records which were arguably mislabeled after the analysis commenced. We contend that the low incidence of mislabeled terms, though positive, does not significantly impact the performance of the algorithms implemented.

Table 1 contains a description of the values of CMINDC within the analysis dataset, many of which appear more than once. Approximately 35.7% of the data are labeled as indicated for COPD (i.e., the 'truth'), compared to 3.1% of the unique terms. Among the unique terms, 2.3% contain 'COPD' as an exact string with 10 not labeled as COPD. For example, a CMINDC value of 'NON COPD' or 'PANIC ATTACK SECONDARY TO COPD' is not labeled as COPD. In addition, 48 unique records contain one of the following strings, treated as an exact proxy for COPD:

- CHRONIC OBSTRUCTIVE PULMONARY DISEASE
- CHRONIC BRONCHITIS
- EMPHYSEMA

For similar reasons to the string 'COPD', not all records containing a proxy string are labeled COPD.

The remaining unique records labeled as COPD (20.5%) contain misspellings of one of these four proxies, such as 'COIPD', 'CHRINIC OBSTUCTIVE PULMONARY DISEASE', 'EMPHYSIMIA', and 'CHRONIC BRONHITIS'. Correct classification of these records is the ultimate objective of this discussion.

	All Values (N=212,079)		Unique Terms (N=18,481)	
	Yes	No	Yes	No
Labeled as COPD medication	75,626 (35.7%)	136,453 (64.3%)	567 (3.1%)	17,914 (96.9%)
Contain 'COPD' exact string	72,570 (34.2%)	139,509 (65.8%)	428 (2.3%)	18,053 (97.7%)
Contain exact proxy of 'COPD'	2,182 (1.0%)	209,897 (99.0%)	48 (0.3%)	18,433 (99.7%)
Labeled as COPD and not contain exact string for 'COPD' nor proxy	935 / 75,626 (1.2%)		116 / 567 (20.5%)	

Table 1. Descriptive Statistics of CMINDC within the Analysis Dataset

We stratify the analysis dataset into two groups using Python's **train_test_split** from Scikit-Learn **model_selection**. After stratification, we have two datasets: a training dataset which contains 80% of the records (169,663 records) and a test dataset which contains the remaining 20%. From this point, we retain only unique values of CMINDC. The training dataset contains 15,988 such unique values, of which 498 (3.1%) are labeled as COPD (see Table 2). The stratification method works well, as the overall training dataset has 60,501 (35.7%) records labeled as COPD, consistent with the combined train-test data.

	Training Dataset (N=15,988)	Test Dataset (N=6,143)
Labeled as COPD medication	498 (3.1%)	195 (3.2%)
COPD labeled records appearing in only one dataset	372 / 498 (74.7%)	69 / 195 (35.4%)

Table 2. Unique Terms in Train vs Test Datasets

The test dataset contains 69 additional unique values of CMINDC that are labeled COPD and do not exist in the training dataset (see Table 2). This means that we have approximately 12.2% of our unique global list of labeled COPD indications which are not available to the training dataset (and will be included in the test phase). All algorithms discussed in this paper are calibrated on the training dataset prior to running on the test dataset.

CLASSIFICATION USING BRUTE FORCE METHODS

When faced with the issue of classifying records based on free-text collected values, there are typical models that clinical programmers implement to accomplish the objective.

MODEL

One classification model uses a data dump of all unique values of the free-text field to a spreadsheet. These values are manually reviewed and flagged as 'Y' or 'N' and then reimported during program execution. In some cases, perhaps a medical review of the free-text information would be required to ensure that the data is classified correctly. This model takes less novel programming to export and import a spreadsheet but requires more resources than other models due to the need for human review of every data point and maintenance of the spreadsheet. As new data comes in, the SAS code would not need any updating, but the spreadsheet would. There is also the potential for human error, either due to misclassification or due to mishandling of the updated data collated with previous iterations. Validation of the classification process may be time consuming and at risk for error. We do not include this model in our comparisons of models in this paper, but we mention it as it is a solution that programming teams implement.

A second classification model reads in the data and programmatically flags each record using SAS code with conditional processing. For example, one might develop SAS code with exact matching of strings such as flagging all records which contain a list of strings *except* for those that also contain a separate list of strings. Flagging all records with 'COPD' *except* those with 'NON-COPD' would be an attempt to accurately flag COPD records. This is the classification model that is applied in this section.

METHOD

We look at 2 methods that leverage programmatic and exact string searching. The first method is a relatively low-tech string search which looks for exact string matching of COPD and its 3 proxies. The second method expands upon the low-tech method, developed on the training dataset to provide 100% accuracy with minimal code.

Sample SAS code for the low-tech method is provided below. Sample SAS code developed for 100% accuracy on the training dataset is provided in the Appendix.

```
data train_method1;
  set train_unique;
  if find(cmindc,"COPD") +
     find(cmindc,"CHRONIC BRONCHITIS") +
     find(cmindc,"EMPHYSEMA") +
     find(cmindc,"CHRONIC OBSTRUCTIVE PULMONARY DISEASE") > 0
     then COPDYN_METH1=1;
  else COPDYN_METH1=0;
run;
```

RESULTS

Results on the training dataset using these methods on the unique values of CMINDC are summarized in Table 3. Focusing on unique values of CMINDC, the low-tech method is 99.3% accurate. Although this sounds quite high, the ability to identify records inaccurately flagged relies on already knowing the true labels. Indeed, the method correctly flags 392 records as COPD which are indeed COPD. Identifying the 12 of 404 flagged records which are not truly COPD (precision = 97.0%) takes little effort, due to the low volume of flagged records. However, an extensive review of the remaining unique values would be required to identify which 106 records of the other 15,584 are not flagged as COPD but should be (recall = 78.7%). In the context of reporting on clinical trial data, false positives are less problematic because they are more easily reviewed and remedied. As mentioned previously, this insight will drive the training of our machine learning algorithms to err on the side of higher recall at the expense of lower precision.

Classification Method	Accuracy	Precision	Recall	False Positives	False Negatives
Brute Force – Low Tech	0.993	0.970	0.787	12	106
Brute Force – 100% Accuracy	1.000	1.000	1.000	0	0

Table 3. SAS Exact String Matching

Of course, since the second method is developed for 100% accuracy, there are no false negatives nor false positives. This means that both precision and recall are 100%. The focus here is the complexity of the code required to generate these results, with a forward-looking perspective on the code maintenance required when additional (new) data is added to the database (see Appendix). While this code may not be the most efficient method for coding to 100% accuracy, it is effective.

There are a few points to highlight with respect to the 100% Accuracy code. First, the ability to generate this code quickly was heavily dependent on already knowing the truth. In practice, it would require an iterative process of flagging records, reviewing the records flagged as COPD or not, and updating code until the developer was satisfied with the results. Second, note the elements of conditional processing that are unique to the data. In some cases, we have added an 'else' statement for a general class of records that are either COPD or not. In other more specific cases, we have extracted the last few records that would otherwise be flagged incorrectly to flag them correctly by brute force. This tailoring of

conditional string searching to the training dataset reduces confidence in the performance of the same code on the test dataset, particularly because we know that there are 69 unique 'true' terms in the test data which do not exist in the training dataset. Finally, we note at least 3 records that were misclassified as true COPD in our initial manual labeling of CMINDC terms ("ANXIETY AFTER HER HUSBANDÂŽS DEATH", "HYPERTENSIÃ‘N", and "RESCUE FOR"). Although we do not expect that this significantly impacts the results of machine learning algorithms, it is important to call out.

CLASSIFICATION USING FUZZY MATCHING IN SAS

MODEL

We consider 4 scoring methods for fuzzy matching in SAS. One popular technique uses the COMPGED function which calculates a generalized edit distance based on the well-known Levenshtein edit distance [3]. Another SAS function is SPEDIS, which measures the difference in spelling of two strings. A third technique uses the SOUNDEX function, which implements the Odell-Russel algorithm (US Patents 1261167 (1918) and 1435663 (1922)). Details on the algorithm can be found in Knuth (1973) [7] and SAS documentation [13]. In Kevin Russell's SAS blog, SOUNDEX is used in combination with COMPGED to produce a distance metric between two strings [11]. In what follows, we refer to this model as the COMPGEDX approach. We apply a similar combination method with the SPEDIS function for a fourth fuzzy matching approach.

METHOD

For each of the fuzzy matching methods, we calculate the score of a free-text field value with each of the 4 proxy terms and retain the lowest score, i.e., the best match for that scoring method. We then linearly normalize the range of the scores calculated across all unique terms on a 0 to 1 scale. For any score X, the normalized score is Z = (X-Min)/(Max-Min). Then, for a threshold Z* such that 0 < Z* < 1, any record with a normalized score Z < Z* would be flagged as COPD based on the distance metric and the 4 proxy terms.

RESULTS

For classification algorithms that produce a score, a threshold must be chosen that converts each score into a class. Once a threshold is chosen, any input value that produces a score greater than or equal to the threshold is labeled as a positive class. Conversely, any input value that produces a score less than the threshold is labeled as a negative class. Typically, a Precision and Recall vs Threshold plot is used to determine the desired threshold. This threshold can be adjusted to be higher to favor precision or lower to favor recall. Commonly, it is desirable to choose a threshold value that balances precision and recall. However, the balance is always ultimately determined by the context in which the classifier is being used and, as discussed, our context favors higher recall at the expense of lower precision.

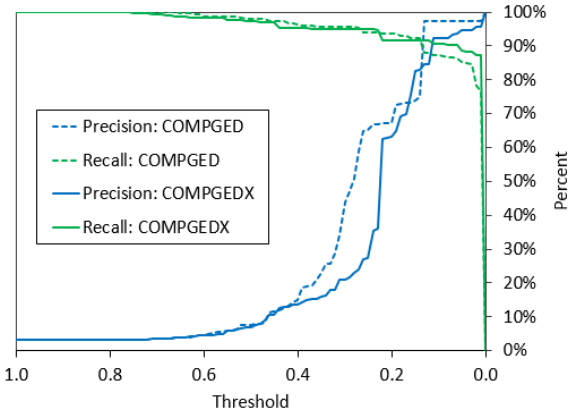

Figure 1. SAS Fuzzy Matching with COMPGED and COMPGEDX scoring methods

Precision and Recall vs Threshold plots for both COMPGED and COMPGEDX are presented in Figure 1 and Table 4. We reverse the direction of the horizontal axis to be consistent with results presented on machine learning algorithms. In the ML comparative statics on 'threshold', a higher threshold is interpreted as higher *similarity* to the proxy terms, whereas in this context a higher threshold is interpreted as more *dissimilarity.*

The two scoring methods performed similarly on our training dataset. Based on the analysis, thresholds of Z*=0.26 for COMPGED and Z*=0.22 for COMPGEDX are considered the best calibration of the SAS fuzzy matching methods using the training dataset. Increases in threshold increase recall and reduce precision, but the tradeoff is not a net gain. Precision drops to 0.589 and 0.361, respectively, but recall increases to 0.954 and 0.948, respectively.

Classification Method	Z*	Accuracy	Precision	Recall	False Positives	False Negatives
COMPGED	0.26	0.982	0.650	0.940	252	30
COMPGEDX	0.22	0.980	0.626	0.918	273	41

Table 4. Results on Calibrated SAS Fuzzy Matching Algorithms

Although we also considered the SPEDIS function, it did not perform well on the training data, with or without passing the terms through SOUNDEX. Therefore, we exclude all results from the presentation.

CLASSIFICATION USING FUZZY MATCHING IN PYTHON

MODEL

We consider five scoring methods for fuzzy matching in Python. The first utilizes the Damerau-Levenshtein edit distance metric [6], a well-known variety of the Levenshtein distance [8]. The Damerau-Levenshtein metric counts the number of character additions, subtractions, and transpositions that are necessary to transform one string into another.

The remaining four methods are included in the Python **fuzzywuzzy** package, created and open sourced by the company SeatGeek [12]. The four methods are based on the Levenshtein distance metric but manipulate the two input strings in a variety of ways before applying the metric. A detailed explanation is available on the SeatGeek blog [2].

METHOD

To compare the five methods, we normalize each metric to a 0-1 scale. We convert the Damerau-Levenshtein distance metric into a scaled similarity metric by calculating the Damerau-Levenshtein distance, dividing by the length of the longer string (to normalize), and finally subtracting that value from one (to transform it into a similarity metric). The score from each of the remaining four metrics is divided by 100 to normalize the score to the 0-1 scale.

For each of the fuzzy matching methods, we again calculate the score of a free-text field value with each of the 4 proxy terms and retain the highest score, i.e., the best match for that scoring method. For any normalized metric score Z, and for any threshold $0 < Z^* < 1$, we flag any record with a normalized score $Z < Z^*$ as COPD based on the distance metric and the four proxy terms.

RESULTS

Precision and Recall vs Threshold plots for all five Python methods are presented in Figure 2. Two of the four **fuzzywuzzy** package methods (Partial Ratio and Partial Token Sort Ratio), along with the Damerau-Levenshtein custom approach, yield similar positive results. These methods are applied on the test data, while the other two methods are dropped.

Given our desire to optimize recall to ensure that as many true positives were eventually identified, either by the model or with a concentrated manual review of the outputs of the model, thresholds for all five models are chosen accordingly and presented in Table 5. At the chosen thresholds, the three selected

fuzzy matching methods produce the most desired results: high accuracy and recall, with moderate precision. This ensures that most of the true positives are identified while minimizing the false positives.

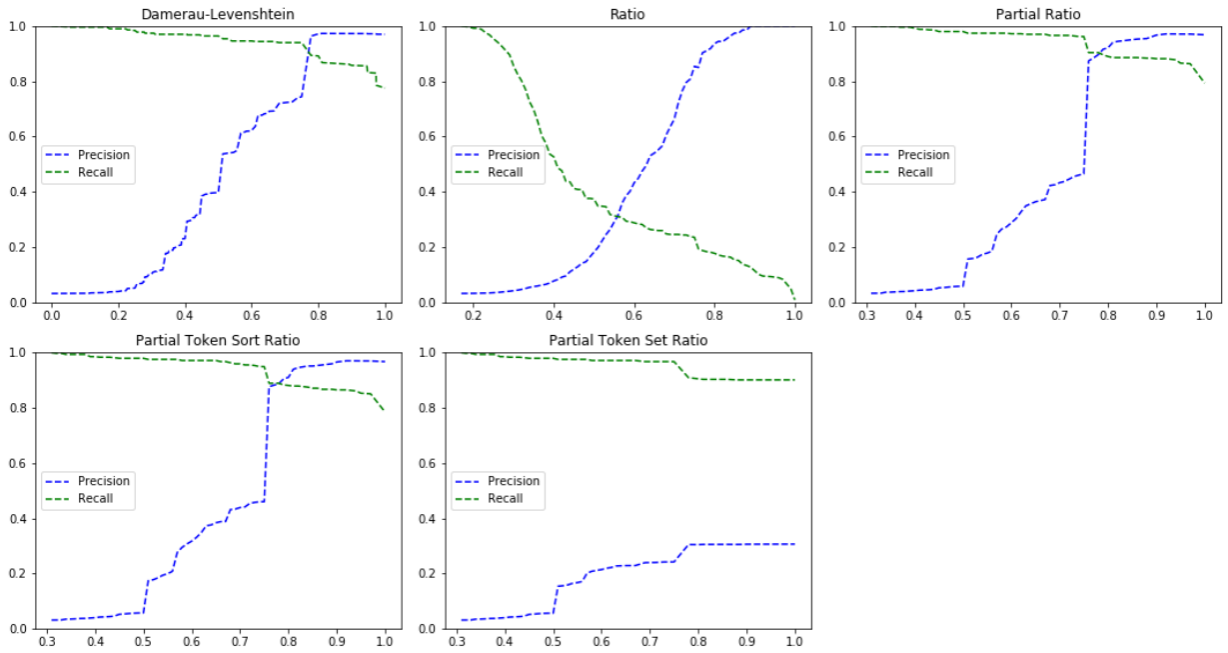

Figure 2. Python Fuzzy Matching with various scoring methods

Classification Method	Z*	Accuracy	Precision	Recall	False Positives	False Negatives
Damerau-Levenshtein based custom approach	0.74	0.988	0.744	0.939	161	30
Ratio	0.55	0.957	0.314	0.309	336	344
Partial Ratio	0.78	0.994	0.915	0.895	41	52
Partial Token Sort Ratio	0.77	0.993	0.887	0.889	56	55
Partial Token Set Ratio	0.50	0.832	0.154	0.975	2663	12

Table 5. Results on Calibrated Python Fuzzy Matching Algorithms

Interestingly, the three chosen methods all experience a dramatic shift in precision around a threshold value of 0.75. Investigation into the data reveals that this is largely due to data points containing the word "COLD." These three methods take different approaches to what is essentially the same process: given two input strings, compare the shorter string to substrings of the larger string and compute a ratio of matching characters. Since "COPD" (one of the four proxy words) and "COLD" share 75% of the characters in the same order, they end up with roughly a 0.75 similarity score from all three methods.

EXPLORING THE DATA USING UNSUPERVISED LEARNING TECHNIQUES

Unsupervised learning algorithms are algorithms that identify relationships and patterns in data without knowledge of categories or target characteristics. This type of data is called *unlabeled* data and is prevalent in many industries, thus making unsupervised learning a critical tool in understanding data. Typically, unsupervised learning techniques accept a vectorized data set as input, and output a way of grouping the data points, either visually or with numerical labels. In our context, we use unsupervised learning algorithms to explore the data only. We do not convert the algorithms into classifiers to make

predictions on data outside the training dataset, though this is often an application of unsupervised learning.

DATA VECTORIZATION

Machine learning models take as input arrays or matrices of numbers, therefore textual datasets must be converted into meaningful numeric representations. This process of converting text (or other non-numeric data such as images or sounds) to numerical representations is called *data vectorization*.

For this paper we applied a standard text vectorization technique which

- First tokenizes each data point (referred to as a "document") into separate words and multiword "n-grams" (1-4 words per token)
- Counts the number of occurrences of each token for each document (a process referred to as the "bag of words" technique)
- It then adjusts the weights of the counts by the overall frequency of each token in the entire dataset (a process called "Term Frequency - Inverse Document Frequency" or TF-IDF)

This technique vectorizes each document into an array where each array index represents a token, and the value at that index is a weight for that token specific to that document. The resulting document-term matrix is a meaningful numerical representation of the documents in the dataset and can be used as input into ML models. Applying this technique to our training set resulted in 40,729 unique tokens which are represented by the columns in the document-term matrix.

To account for misspellings of the two main terms of interest, "COPD" and "CHRONIC OBSTRUCTIVE PULMONARY DISEASE", we added four features that capture similarity to those terms using two different string similarity metrics. The two string similarity metrics, Jaro-Winkler and Damerau-Levenshtein, measure edit distances between two strings in different ways. They were used to create features of similarity metrics between the two terms of interest and each data point.

VISUALIZING DATA

To gain a better understanding of the data during the data science process, often the data is visualized using various transformations and plotting techniques, many of which are unsupervised learning techniques. Visualizing data allows data scientists to see patterns that may be harder to find through purely analytical methods.

Visualizing high dimensional datasets is extremely difficult. Thus, dimensionality reduction techniques are used to compress the data into fewer dimension while preserving as much of the variation and correlation in the dataset as possible. A standard dimensionality reduction technique is Principle Component Analysis (PCA). However, because our dataset had 40,729 features, PCA was impractical to run in our analysis environment. Therefore, we chose another technique called Truncated Singular-Value Decomposition (SVD) because it achieves similar results as PCA but requires far few resources. We applied SVD to the training set and reduced it to 10 features. We then inputted those 10 features into an unsupervised learning embedding technique called t-distribution Stochastic Neighbor Embedding (t-SNE) to further reduce the data set down to two features, which we use to visualize each data point in Figure 3. In the figure:

- Each dot represents one data point from the training set
- The red dots are true COPD related data points
- The black dots on non-COPD related data points
- Dots that are close together on the plot are similar to each other across the 40,729 features of the dataset
- The x and y coordinates of the dots are the outputs of the t-SNE embedding and have no direct interpretation

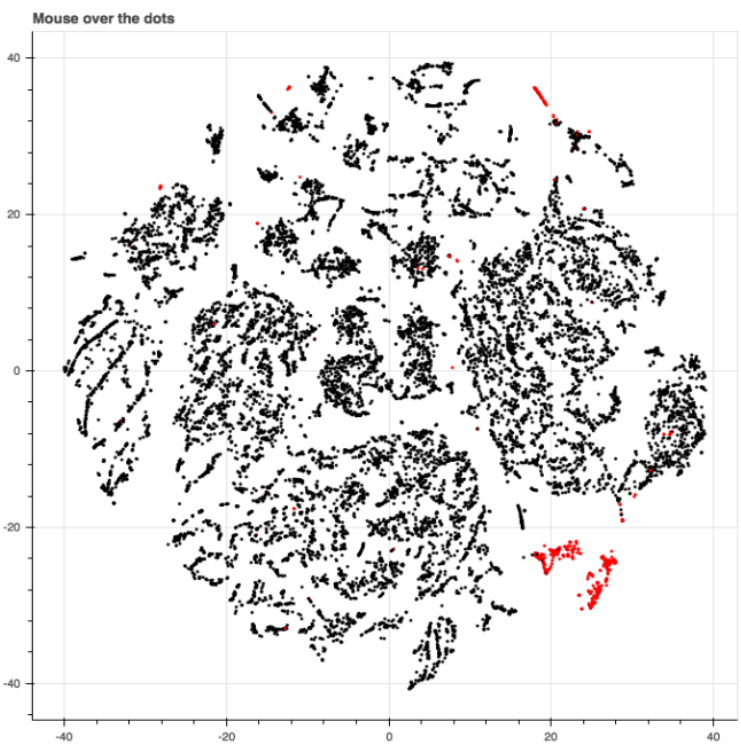

Figure 3. Truncated Singular-Value Decomposition and t-SNE of Training Set

There are two main groups of red dots. Those in the lower right contain "COPD" or a misspelling in the text. Not all data points containing "COPD" are in that group, but the majority are. The smaller group in the upper right contain "CHRONIC OBSTRUCTIVE PULMONARY DISEASE " or slight misspellings in the text. These two groups are the data points most likely to be correctly classified by techniques discussed in later sections when we use the same feature set.

The red dots that appear in smaller groups or isolated from other red dots represent data points that have fundamental differences from the rest of the data set and will be more difficult for techniques to correctly classify.

CLASSIFICATION USING SUPERVISED LEARNING

Supervised Learning algorithms are algorithms that learn relationships between input characteristics about the data and a target characteristic that needs to be predictive on new (previously unseen) data points. Data that has the value of the target characteristic already identified is called *labeled* data and is available much less frequently than unlabeled data, in part because often humans are required to label the data which is costly and time consuming. But when labeled data is available, supervised learning techniques are powerful tools for predicting information about data.

LOGISTIC REGRESSION CLASSIFIER

Model

Logistic Regression Classifier (as known as logit and Max-Entropy) is a linear regression model that estimates the probability that a data point belongs to a certain class, and if that probability is higher than the probability threshold (0.5 by default), then the classifier predicts the data point belongs to the class. Thus, it is a binary classifier. Logistic Regression has a regularization hyperparameter "c" which helps guard against overfitting.

Method

We use scikit-learn's `LogisticRegressionCV` estimator class which uses cross-validation to choose the best value for the hyperparameter "c". [9] We also use scikit-learn's set of cross-validation functions `cross_validate` and `cross_val_predict` to calculate average accuracy, recall, and precision scores for various values of the probability threshold. [4, 5] Once the best model is selected, the probability threshold can be adjusted to tune the Recall/Precision trade-off.

Training Results

Initial training was with 5-fold cross validation within `LogisticRegressionCV` and 3-fold cross validation with `cross_validate` and gave the following promising results in Table 6:

Classification Method	Threshold	Accuracy	Precision	Recall	False Positives	False Negatives
Logistic Regression	0.50	0.995	0.969	0.876	14	62

Table 6. Training Results with Logistic Regression Classifier

In addition to the metrics above, an additional metric used to evaluate classification models is the Area Under the Curve of the Receiver Operating Characteristic, abbreviated ROC AUC or just AUC. This curve is created by plotting the true positive rate against the false positive rate for various thresholds. Thus, it acts as a summary statistic for the model across all potential threshold values. The range of AUC values is 0-1, with 1 being a perfect classifier. Figure 4 presents the ROC for our Logistic Regression model with an AUC of 0.991, an exceptionally high value for typical classification problems.

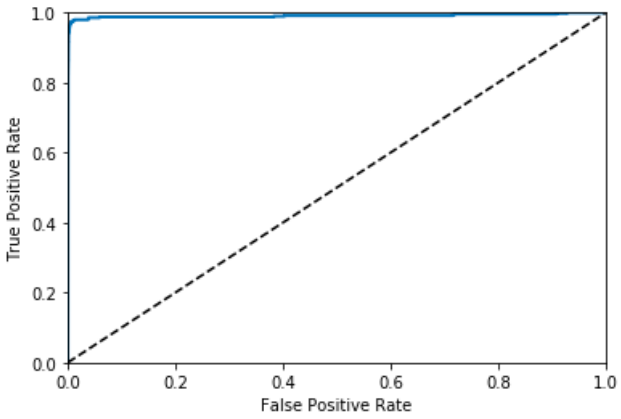

Figure 4. ROC Area Under the Curve for Logistic Regression Classifier

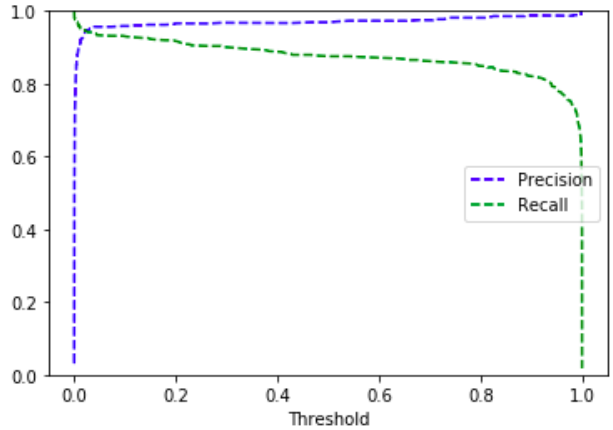

Figure 5. Precision-Recall Plot for Logistic Regression Classifier

Figure 5 is the Precision and Recall vs Threshold plot, which illustrates that the default threshold of 0.5 favors precision at the cost of a lower recall. This threshold can be adjusted to be higher to favor precision or lower to favor recall. In our paper it is desirable to calibrate recall to ensure that as many true positives are identified, either by the model or with a concentrated manual review of the outputs of the model. Tuning the threshold value can be done by calculating the precision and recall for various threshold values. Table 7 presents results for this logistic regression model with a threshold value of 0.005.

Classification Method	Threshold	Accuracy	Precision	Recall	False Positives	False Negatives
Logistic Regression	0.005	0.995	0.872	0.970	71	15

Table 7. Results on Calibrated Logistic Regression Classifier

RANDOM FOREST CLASSIFIER

Model

The Random Forest Classifier is an ensemble model that trains several Decision Tree Classifiers (a forest of decision trees) on different subsamples of the dataset and aggregates the results to produce class prediction probabilities for data points. Similar to Logistic Regression, if the probability is above a threshold value (0.5 by default), then the classifier predicts the data point belongs to the class. This model has several hyperparameters including the number of decision trees to build, how deep trees are allowed to grow, and a minimal number of data points that must be in each leaf in order to split. These hyperparameters help improve prediction performance and guard against over-fitting.

Method

We used the `RandomForestClassifier` class in scikit-learn and used the default values for all hyperparameters [10]. We again use scikit-learn's set of cross-validation functions `cross_validate` and `cross_val_predict` to calculate average accuracy, recall, and precision scores for various values of the probability threshold. Once the best model is selected, the probability threshold can be adjusted to tune the Recall/Precision trade-off.

Training Results

Initial training was with 3-fold cross validation 3-fold cross validation with `cross_validate` gave the following promising results in Table 8:

Classification Method	Threshold	Accuracy	Precision	Recall	False Positives	False Negatives
Random Forest	0.50	0.996	0.982	0.876	7	62

Table 8. Training Results with Random Forest Classifier

The ROC for the Random Forest Classifier (Figure 6) has an AUC of 0.984, which is extremely high and similar to the Logistic Regression. Figure 7 presents the Precision and Recall vs Threshold plot which shows that the default threshold of 0.5 favors precision at the cost of a lower recall. Once again, the threshold must be chosen with the problem context in mind. In our paper it was desirable to optimize recall to ensure that as many true positives were eventually identified, either by the model or with a concentrated manual review of the outputs of the model.

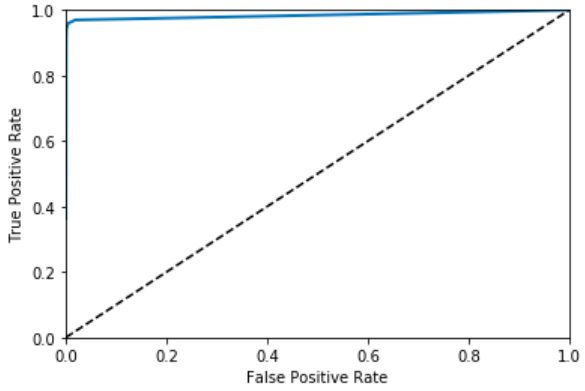

Figure 6. ROC Area Under the Curve for Random Forest Classifier

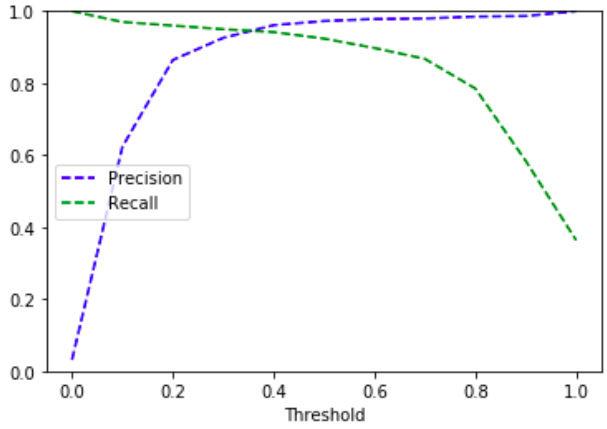

Figure 7. Precision-Recall Plot for Random Forest Classifier

Table 9 presents results for this random forest model with a threshold value of 0.26.

Classification Method	Threshold	Accuracy	Precision	Recall	False Positives	False Negatives
Random Forest	0.26	0.996	0.910	0.960	47	20

Table 9. Results on Calibrated Random Forest Classifier

COMPARISON OF CLASSIFICATION METHODS ON TEST DATA

Ultimately, models are developed to be applied to previously unseen data. This includes data which is generated or collected once the model is deployed to production, and thus the data is not part of the training set. To simulate this, the dataset that is available for training is typically split into a "training" set and a "test" set. All previous results in the paper were based on the training set. Only now, after we have calibrated our models the best we can on the training set, do we use them to classify medications as COPD on the test set. The results on the test set will hopefully mimic results on production data.

The results of the models on the test data set are compiled in Table 10. All models performed well on the test set. Often data elements that exist in the test set but not in the training set (simulating a possible production situation) will cause models to have lower, and sometimes substantially lower, metrics on the test set - that is not the case here. All models performed well on the test set, even with the 69 new unique true COPD terms existing in the set.

Classification Method	Accuracy	Precision	Recall	False Positives	False Negatives
Brute Force – Low Tech	0.993	0.993	0.769	1	45
Brute Force – 100% Accuracy	0.999	1.000	0.974	0	5
COMPGED	0.983	0.665	0.918	90	16
COMPGEDX	0.982	0.657	0.892	91	21
Damerau-Levenshtein	0.988	0.763	0.908	55	18
Partial Ratio	0.993	0.901	0.882	19	23
Partial Token Sort Ratio	0.993	0.883	0.887	23	22
Logistic Regression	0.987	0.709	1.000	80	0
Random Forest	0.998	0.937	0.995	13	1

Table 10. Comparison of Results on Test Data with Calibrated Classification Methods

Figure 8 presents false positive and false negative results on the test data, ordered and labeled by the incidence of false negatives. While all models performed well, the two ML models, Logistic Regression and Random Forest, performed the best, achieving perfect and near perfect recall. As explained previously, recall is the metric most important to the context of this paper, as we need to identify as many COPD data points as possible.

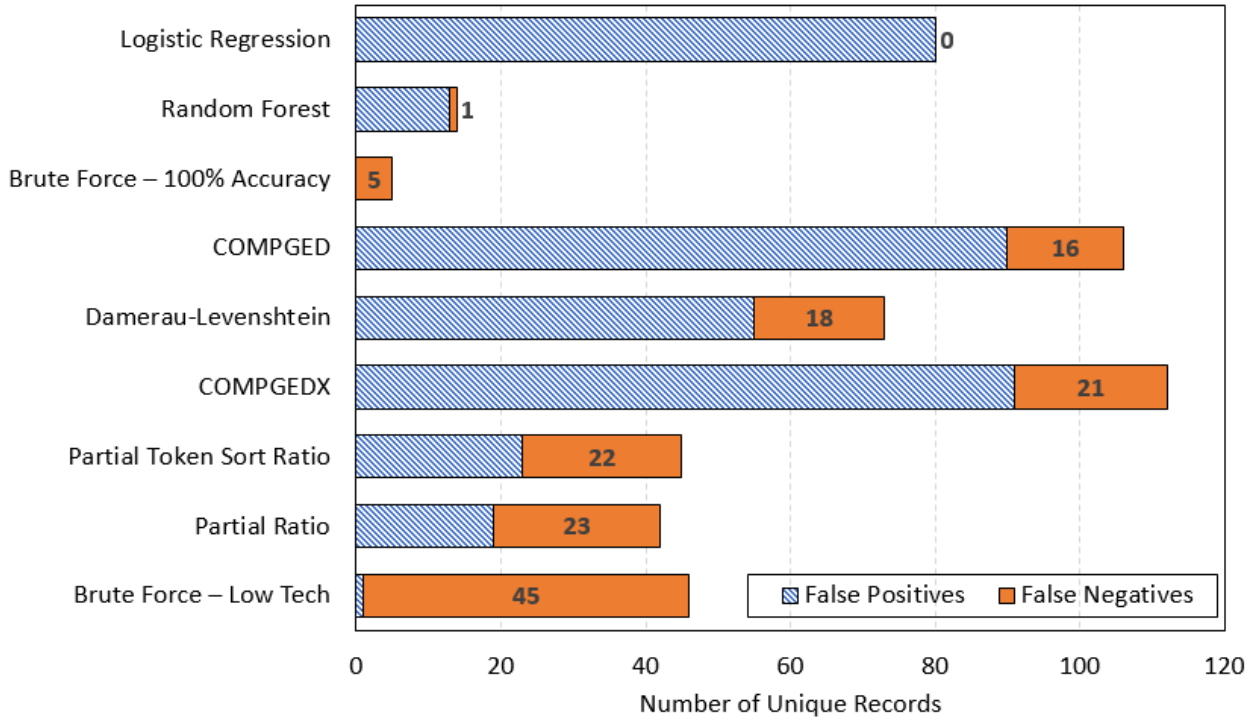

Figure 8. Incidence of False Positives and False Negatives on Test Data with Calibrated Classification Methods

The Brute Force – 100% Accuracy method also performed well. There was no incidence of any new unique terms being falsely flagged as COPD (false positive). However, 5 of the 69 new terms known to be COPD were incorrectly flagged as non-COPD. In practice, a complete review of the new data would be required to identify these new terms and update the SAS code appropriately. The 5 new values are:

- ACUTE EXACERBATION CHRONIC BROCHITIS
- ACUTE EXACERBATION OF CHRONIC BRONCHITS
- CHRONIC OBSTRUCTIVE LUNG
- CHRONIC OBSTRUCTIVE LUNG DISESE
- RESCUE BRONCHODIALATOR

Modification of the code in the Appendix is possible to attain 100% accuracy, but continued maintenance of such code (and communication across study team) can be challenging to an organization.

An important distinction between the string comparison methods and the ML methods is the reliance on the four proxy terms. The string comparison methods rely on those proxy terms representing all the true COPD data points, and none of the non-COPD data points. Conversely, while the ML methods do have features derived from the proxy terms, the majority of their features are unrelated to the proxy terms, allowing these methods to learn relationships between their features and the true COPD data points. This ability to learn from the data alleviates the programmer or analyst from having to identify the appropriate proxy terms for each problem. Instead, the ML methods simply need a sufficiently large set of labeled data (something also necessary for determining the appropriate proxy terms for string comparison methods) on which to train.

CONCLUSION

Our investigation into various fuzzy matching and ML techniques shows ML techniques in Python can perform as well or better than standard SAS and custom fuzzy matching approaches without the need to have a curated set of proxy terms. Noting that the field under analysis (CMINDC) typically contains few vocabulary words which can be leveraged, it can be challenging to improve ML techniques without additional variables that help classify the truth. However, moving forward, there are opportunities for improvement. Additional steps to potentially improve the ML models include:

- Expanding the analysis dataset to include additional features from collected data in the case report forms such as medication coding (PT and ATC)
- Performing hyperparameter tuning to find model parameters which optimize results
- Performing additional text processing to the input data, such as stemming algorithms, before applying the TF-IDF technique
- Fixing the labeling inaccuracies of COPD medications in the analysis dataset, which may misinform supervised learning algorithms

Another area of ‘next steps’ would investigate the value of converting our unsupervised learning techniques into classifiers, as well as developing a hybrid method of unsupervised and supervised techniques. Given the performance of ML techniques on our data, it would also be interesting to assess the performance in other use cases, such as those mentioned in the Introduction. Finally, one could operationalize one of the trained ML algorithms and deploy it for use on an ongoing study to assess true time and cost savings.

REFERENCES

[1] Cicconetti, Greg, David Wade, and Shani Sampson. 2015. “In-stream Text Mining of Patient Narratives: Searching for Dangling Endpoints.” *New England Statistical Symposium*.

[2] Cohen, Adam. "FuzzyWuzzy: Fuzzy String Matching in Python" *ChairNerd Blog*, July 8th, 2011. Accessed April 24, 2019. Available at https://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/

[3] COMPGED Function. *SAS 9.4 Documentation: Functions and CALL Routines.* Accessed April 10, 2019. Available at

https://documentation.sas.com/?docsetId=lefunctionsref&docsetTarget=p1r4l9jwgatggtn1ko81fyjys4s7.htm&docsetVersion=9.4&locale=en

[4] cross_validate. *scikit-learn.* Accessed April 24, 2019. Available at https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html

[5] cross_val_predict. *scikit-learn.* Accessed April 24, 2019. Available at https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_predict.html

[6] Damerau-Levenshtein distance. *Wikipedia.* Accessed April 24, 2019. Available at https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance

[7] Knuth, D.E. 1973. *The Art of Computer Programming*. Volume 3. Sorting and Searching, Reading, MA: Addison-Wesley.

[8] Levenshtein distance. *Wikipedia.* Accessed April 24, 2019. Available at https://en.wikipedia.org/wiki/Levenshtein_distance

[9] LogisticRegressionCV. *scikit-learn.* Accessed April 24, 2019. Available at https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegressionCV.html

[10] RandomForestClassifier. *scikit-learn.* Accessed April 24, 2019. Available at https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[11] Russell, Kevin. "How to perform a fuzzy match using SAS functions." *SAS Blogs*. January 27, 2015. Accessed April 10, 2019. Available at https://blogs.sas.com/content/sgf/2015/01/27/how-to-perform-a-fuzzy-match-using-sas-functions/

[12] seatgeek/fuzzywuzzy repository. *GitHub.* Accessed April 24, 2019. Available at https://github.com/seatgeek/fuzzywuzzy

[13] SOUNDEX Function. *SAS 9.4 Documentation: Functions and CALL Routines.* Accessed April 10, 2019. Available at https://documentation.sas.com/?docsetId=lefunctionsref&docsetTarget=n1i9a3o4kciemhn1kpgutl20e4i0.htm&docsetVersion=9.4&locale=en

ACKNOWLEDGMENTS

The authors would like to acknowledge the following individuals who were integral to making the data available for the analyses:

- Graeme Archer (Clinical Trial Domain)
- Marc Harris (Business Planning Outsourcing)
- Nick Locantore (Clinical Trial Domain)
- Heath Roberts (External Access Services)
- Shani Sampson (RDIP)
- Jennifer Van Ekelenburg (Human Subject Research Governance)

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Michael S. Rimler
GlaxoSmithKline
Email: michael.s.rimler@gsk.com

Matt Pitlyk
1904labs
Email: matt.pitlyk@gmail.com

APPENDIX

All code used in our analyses is available at the following GitHub repository:

https://github.com/MattPitlyk/PharmaSUG-2019--BP-079

For ease of reference, we include the SAS code used in the Brute Force for 100% Accuracy method.

SAS BRUTE FORCE CODE FOR 100% ACCURACY

```
data train_method2;
  set train_unique;
  if find(cmindc,"COPD") +
    find(cmindc,"CHRONIC BRONCHITIS") +
    find(cmindc,"EMPHYSEMA") +
    find(cmindc,"CHRONIC OBSTRUCTIVE PULMONARY DISEASE") > 0 then do;
      if find(compress(tranwrd(cmindc,"-"," ")),"NONCOPD")>0 then
         COPDYN_METH1=0;
      else if cmindc in
         ("ANXIETY DUE TO COPD"
          "DESATURATION DURING EXCERCICE (COPD)"
          "HYPERGLYCEMIA DURING COPD EXACERBATION"
          "PANIC ATTACK SECONDARY TO COPD"
          "SECONDARY INFECTION OF EMPHYSEMA"
          "SPUTUM WITH CHRONIC OBSTRUCTIVE PULMONARY DISEASE")
        then COPDYN_METH1=0;
      else COPDYN_METH1=1;
    end;
  else if find(compress(tranwrd(cmindc,"-"," ")),"COPD") +
          find(compress(tranwrd(cmindc,"."," ")),"COPD") >0 then
     COPDYN_METH1=1;
  else if (min(find(cmindc,"CHRONIC"),1)     +
           min(find(cmindc,"OBSTRUCTIVE"),1) +
           min(find(cmindc,"PULMONARY"),1)   +
           min(find(cmindc,"DISEASE"),1)
           ) > 2 and
          find(cmindc,"HEART") = 0 then COPDYN_METH1=1;
  else if min(find(cmindc,"BRONCHITIS"),1)   +
          min(find(cmindc,"BRONHITIS"),1) +
          min(find(cmindc,"CHRONIC"),1) >= 2 then COPDYN_METH1=1;
  else if length(CMINDC)<=4 and
          (min(find(cmindc,"C"),1) + min(find(cmindc,"O"),1) +
           min(find(cmindc,"P"),1) + min(find(cmindc,"D"),1)) > 2 and
           cmindc not in ("APOC" "BPCO" "BPOC" "CORP" "DROP"
                          "EPOC" "MPOC" "PBCO" "POCD") then COPDYN_METH1=1;
  else if find(cmindc,"SYMBICORT")>0 then COPDYN_METH1=1;
  else if find(cmindc,"AIRWAY OBSTRUCTION")>0 and
          cmindc ne "UPPER AIRWAY OBSTRUCTION" then COPDYN_METH1=1;
  else if find(cmindc,"OPD")>0 then COPDYN_METH1=1;
  else if find(compress(tranwrd(cmindc,"-"," ")),"RESCUEMED")>0 and
          find(cmindc,"ASTHMA") = 0 then COPDYN_METH1=1;
  else if cmindc in ("EMFYSEMA"
                     "EMPHYSIMIA"
                     "EMPHYSYMIA"
                     "CHOPN"
                     "COIPD"
                     "COPOD"
                     "COPPD"
```

```
                    "CHRINIC OBSTUCTIVE PULMONARY DISEASE"
                    "CHRONIC AIRWAYS DISEASE"
                    "CHRONIC OBSTRUCTIVE LUNG D/O"
                    "CHRONIC OBSTUCTIVE PULMONARY DISESE"
                    "COBD EXECERBATION"
                    "COPE ACUTE EXACERBATION"
                    "COPT EXECERBATION"
                    "EXACERBATION OF BRONCHITIS"
                    "RELIEVER INHALER"
                    "RESCUE DRUG FOR SHORTNESS OF BREATH"
                    "RESCUE INHALER"
                    ) then COPDYN_METH1=1;
  else if cmindc in ("ANXIETY AFTER HER HUSBANDÂŽS DEATH"
                    "HYPERTENSIÃ“N"
                    "RESCUE FOR"
                    ) then COPDYN_METH1=1;
  else COPDYN_METH1=0;
run;
```