Spring 3-14-2020

Treatment Effects of Modafinil for Cocaine Use Disorders: A Retrospective Analysis of Aggregated Clinical Trial Data From Three Cocaine Treatment Studies

Daniel Ruskin
daniel.ruskin@uconn.edu

Treatment Effects of Modafinil for Cocaine Use Disorders: A Retrospective Analysis of Aggregated Clinical Trial Data From Three Cocaine Treatment Studies

Author: Daniel Ruskin
Thesis Advisor: Professor Jinbo Bi

1. Introduction

1.1 Motivation and Objectives

Cocaine is a drug derived from the South African coca plant [1]. The drug is a powerful stimulant that, when exposed to the human body, causes an immediate buildup of the neurochemical dopamine [1]. This buildup induces strong feelings of pleasure and euphoria, which can lead to a desire to take the drug again [1]. Over time, this desire can develop into a biological dependence [1]. Long-term, repeated exposure can increase a user's risk of developing serious cardiac and neurological complications, such as heart attacks, seizures, and strokes [1]. There is also evidence that long-term exposure can induce lasting physiological effects, such as changes in the expression of genes associated with depressive symptoms [3]. The Diagnostic and Statistical Manual of Mental Disorders (currently version 5) (DSM-5) defines the various criteria to diagnose cocaine use disorder (CUD) [4]. However, when becoming dependent on cocaine, many people do not seek medical help, thus missing the opportunity for diagnosis and treatment.

According to the National Survey on Drug Use and Health (NSDUH), approximately 913,000 individuals met these criteria in the United States alone [2]. The widespread usage of cocaine, along with the negative cardiac and neurological effects associated with the drug, has made cocaine one of the top three drugs associated with overdose deaths in the United States, right after opioids such as oxycodone and heroin [5]. This epidemic has brought cocaine dependency into the public spotlight and has prompted extensive research into treatment strategies.

At this time, no drugs have been approved by the United States Food and Drug Administration (FDA) for use in treating CUD. Certain medications have been used to treat CUD, but there are no FDA-approved indications for such use at the time of writing. The purpose of this study is to examine the efficacy of one particular drug, modafinil, in treating CUD. Modafinil has been approved by the FDA to treat certain sleep disorders, including narcolepsy and shift work sleep disorder (SWSD) [20]. Retrospective analysis of longitudinal data aggregated from three clinical trials will be used to examine the efficacy of modafinil in treating CUD.

1.2 Clinical Trial Background

This study examines three clinical trials conducted at the University of Pennsylvania Perelman School of Medicine. The first clinical trial, hereafter referred to as Modafinil-1, aimed

to determine the efficacy of modafinil in treating cocaine dependent subjects [6]. Modafinil-1 included 62 subjects, each of whom was randomly assigned to receive a daily dose of either 400mg of modafinil or a placebo [6]. Modafinil-1 found that modafinil-treated patients had higher abstinence rate, as indicated by these patients returning fewer cocaine-positive urine samples [6]. This encouraging result suggested that modafinil may be efficacious in treating cocaine dependent subjects.

The results from Modafinil-1 encouraged another study at the University of Pennsylvania, referred to as Modafinil-2, with 210 active cocaine users [7]. During Modafinil-2, each subject was randomly assigned to receive a daily dose of 0mg (placebo), 200mg, or 400mg of modafinil [7]. The results of Modafinil-2 were inconclusive; no significant difference in cocaine abstinence was found between the modafinil and placebo groups [7].

The discouraging results of Modafinil-2 were tempered, however, by a separate, concurrent study hereafter referred to as Modafinil-3 [8]. Modafinil-3 included 94 cocaine dependent subjects, each of whom was randomly assigned to receive a daily dose of either 300mg of modafinil or a placebo [8]. Modafinil-3 found that modafinil significantly increased cocaine abstinence, decreased cocaine cravings, and made subjects more likely to rate themselves as “very much improved” on a clinical assessment tool [8]. These results lent strong support to modafinil as an efficacious drug.

These inconsistent study results have muddled the initially positive expectations for modafinil. Modafinil-1 and Modafinil-3 both show some benefits for cocaine addicts to stop or cut cocaine use, suggesting that modafinil is an effective drug, whereas Modafinil-2 suggests the opposite. The current study attempts to reconcile these results by developing and applying a novel variant of Latent Class Analysis (LCA). Specifically, we first hypothesized that certain clusters of cocaine-dependent patients may be more likely to respond positively to modafinil. We then attempted to use our novel LCA method to classify patients into clusters based on both demographic covariates and longitudinal study data. Next, we analyzed the clusters to determine the characteristics of the clusters in terms of demographic features, longitudinal trajectory of treatment response, and the cocaine abstinence rate. This analysis revealed whether clusters significantly differed from each other in these respects.

If our hypothesis holds, this procedure would have allowed our team to discover meaningful clinical criteria for predicting the success of modafinil in treating cocaine dependence. Such a discovery would be impactful on the perception of modafinil as a viable treatment for cocaine use disorders.

2. Methods

Our overall approach consists of five steps: data preprocessing, feature selection, data normalization, model fitting, and latent trajectory analysis. Data preprocessing was an important step in this analysis and served to aggregate data from the three clinical trials. The present report is primarily focused on data preprocessing and statistical analysis. Brief summaries are provided on feature selection, data normalization, and model fitting.

2.1 Data Preprocessing

The University of Pennsylvania research team graciously provided us with de-identified data from the Modafinil-1, Modafinil-2, and Modafinil-3 studies, as well as additional studies that we did not analyze due to time constraints. This data came in the form of more than 100 IBM SPSS files. Combined, these files contained thousands of data points documenting each subject's demographics, laboratory tests, physical exams, and screening and clinical assessments. Our study methodology required us to analyze all of this data. However, we quickly realized that significant preprocessing would need to be performed before any meaningful analysis could take place. The raw data was simply not suitable for out-of-the-box analysis, because [9]:

1. The data files were in the proprietary SPSS format. Our analysis software was custom-written in Python, which cannot easily interpret SPSS files.
2. We intended to perform a multivariate analysis across hundreds of variables, but these variables were all located in different files.
3. Longitudinal variables did not have a consistent format. Some variables were represented in the "long" format, which included one measurement timestamp and one measurement value in each row. In the "long" format, a subject with multiple measurements of a single variable would have multiple rows in a single file. Other variables were represented in the "wide" format, which included many measurement timestamps and many measurement values in each row. In the "wide" format, a subject with multiple measurements would have only a single row per file. Due to human-error during data collection, certain variables were represented in a combination of both formats (hereafter referred to as the "long-wide format").

4. Non-longitudinal variables with multiple measurements also did not have a consistent format.

In order to address these issues and prepare our data for the proposed analysis, we implemented a reformatting pipeline in Python. The pipeline we developed comprised four stages:

1. The Export Data stage, which exported both data files and variable codebooks from SPSS to Excel. Data files contained measurement values, while variable codebooks contained free-form descriptions of each variable.
2. The Parse Variables stage, which converted Excel variable codebooks to CSV variable codebooks.
3. The Merge Data stage, which combined all CSV codebooks and Excel data files into a robust DataMatrix format.
4. The Series Processing stage, which converted all multi-measurement data into a consistent “long” format.

Each stage will now be described in detail.

2.1.1 Data Preprocessing - Export Data Stage

The Export Data stage served to convert the clinical trial data away from the proprietary IBM SPSS format. This task was achieved with the IBM SPSS Python API (the Python API) [10]. The Python API allowed our customized code to interact directly with the SPSS software. Specifically, we used the Python API to programmatically transmit commands to SPSS. The SPSS software then executed those commands and returned result codes back to our program.

The Export Data program accepted as input a set of study folders, each of which must contain one or more SPSS data files. The program then iterated through each input file. Two output files were generated for every input file: a data file and a variable codebook. The data file simply contained a direct export of the data from the SPSS file, while the variable codebook contained a human-readable description of each variable. Each column X in the data file directly corresponded to the X-th variable in the variable codebook. Both the data file and the variable codebook file were formatted as standard Excel files.

The Export Data program source code can be found in [11]. The program takes approximately 30 seconds to process each SPSS file. For reference, a pseudocode description of the Export Data algorithm is also included as Algorithm 1 below. In Algorithm 1, SpssClient refers to the SPSS API.

1. Open the SPSS program via the following API call: SpssClient.StartClient.
2. Direct the SPSS API to open up a new Syntax Document via the following API call: SpssClient.NewSyntaxDoc. Set this value to the variable SDoc.
3. Direct the SPSS API to open up a new Output Document via the following API call: SpssClient.NewOutputDoc. Set this value to the variable ODoc.
4. Set variable O as the current output document. This can be accomplished with the following API call: O.SetAsDesignatedOutputDoc.
5. Set variable PrevDoc as null.
6. Repeat for each study S:
    a. Repeat for each SPSS file in S with path P:
        i. Sleep for a pre-set number of seconds S to allow IBM SPSS to complete pending operations[1].
        ii. Direct the SPSS API to open file path P via the following API call: SpssClient.OpenDataDoc. Set the returned value to the variable DDoc.
        iii. If PrevDoc is non-null, direct the SPSS API to close PrevDoc via the following API call: PrevDoc.CloseDocument.
        iv. Wipe all output from the previous iteration via the following API calls: ODoc.SelectAll and ODoc.Delete. In addition, activate the output document by calling ODoc.SetAsDesignatedOutputDoc.
        v. Export the variable codebook file via the following procedure:
            1. Prepare a command to generate the variable codebook file by calling SDoc.SetSyntax with the following string:
                a. "CODEBOOK ALL /VARINFO LABEL TYPE FORMAT VALUELABELS /OPTIONS VARORDER=VARLIST SORT=ASCENDING MAXCATS=200 /STATISTICS NONE."

[1] On our machine, S = 2 seconds. This was a sufficient amount of time to allow SPSS to complete pending operations.

2. Execute the prepared command via the following API call: SDoc.RunSyntax. The exported data will be contained in the output document ODoc.
3. Call ODoc.SelectAllTables and ODoc.ExportDocument to save the generated data to an Excel file.

vi. Export the data file via the following procedure:

1. Prepare a command to export the data file by calling SDox.SetSyntax with the following string:
    a. "SAVE TRANSLATE OUTFILE='FILE_NAME.xls' /TYPE=XLS /VERSION=12 /MAP /FIELDNAMES VALUE=NAMES /CELLS=VALUES /REPLACE."
2. Execute the prepared command via the following API call: SDoc.RunSyntax. The exported data will be automatically saved to an Excel file.

Algorithm 1: The Export Data algorithm.

2.1.2 Data Preprocessing - Parse Variables Stage

The raw codebook files generated in the Export Data stage were not suitable for immediate processing because these files contained a great deal of extraneous spacing and formatting. For example, each variable description was spread out over several rows. The Parse Variables program served to address this problem. Specifically, the Parse Variables stage accepted a set of Excel codebooks as input, programmatically parsed each codebook, and created a set of simplified CSV codebooks that could be used by the rest of the Preprocessing pipeline.

The Parse Variables program source code can be found in [12]. The program takes approximately one second to process each codebook file. For reference, a pseudocode description of the Parse Variables algorithm is also included as Algorithm 2 below. In Algorithm 2, XLRD refers to an open-source Python library that facilitates the programmatic manipulation of Excel files [13]. Algorithm 2 also makes use of the built-in Python CSV library [14].

1. Repeat for each study S:
    a. Repeat for each Excel codebook file F in S:
        i. Load file F into a Python matrix M via the XLRD library.

ii. Initialize an empty array V.

iii. Iterate through each row R in worksheet M:

1. Determine if row R represents a variable by analyzing its format. A row R represents a variable if:
    a. The first column in R is non-nil
    b. The second column in R is nil
    c. All columns in row R - 1 are nil
2. If R represents a new variable, extract the variable's name and description into V. The name can be accessed as follows: M[R][0]. The description can be accessed as follows: M[R + 2][2].

iv. Export V to a CSV file via the Python CSV library.

Algorithm 2: The Parse Variables algorithm.

2.1.3 Data Preprocessing - Merging Data Stage

The next stage of the Data Preprocessing pipeline is the Merge Data stage. The Merge Data stage ran immediately after the completion of the Export Data and Parse Variables stages. These preceding two stages generated a plethora of Excel and CSV files. We quickly determined that, in order to facilitate efficient data analysis, these files would need to be aggregated into a single data structure. The Merge Data stage was designed to serve this purpose.

The Merge Data stage accepted two file sets as input: the Excel data files generated in the Export Data stage and the CSV variable codebook files generated in the Parse Variables stage. The Merge Data stage then iterated through the files, loaded them into memory one line at a time, and imported the pertinent data into a customized data structure we called DataMatrix. After completing the import process, the Merge Data stage persisted the DataMatrix to a file via the built-in Python serialization library, Pickle [15]. This application of Pickle to the DataMatrix allowed all of the clinical trial data to be saved to a single binary file. Accordingly, future pipeline stages simply accepted DataMatrix files as input and no longer required raw codebooks or raw data files.

The DataMatrix design closely mirrored a key-value storage data structure (e.g. a Python dictionary). Specifically, the DataMatrix stored data by subject identifiers. Each unique subject was allocated a distinct Python dictionary, which could then be populated with an arbitrary number of variable names and values. Canonical variable names were generated for

this purpose by appending source file names to raw variable names[2]. Each variable value was represented by an array to facilitate the storage of longitudinal data.

Note that the Merge Data stage did not merge data across studies. Instead, the Merge Data algorithm simply ran on each study separately. Accordingly, separate DataMatrix files were generated during this stage for the Modafinil-1, Modafinil-2, and Modafinil-3 studies.

The Merge Data program source code can be found in [16]. The program takes approximately 10 seconds to process each codebook file. For reference, a pseudocode description of the Merge Data algorithm is also included as Algorithm 3 below. Note that the XLRD [13], CSV [14], and Pickle [15] libraries were employed in Algorithm 3.

1. Repeat for each study S:
    a. Initialize an empty DataMatrix D.
    b. Repeat for each data file F in S:
        i. Load file F into a Python matrix M via the XLRD library.
        ii. Load the variable codebook associated with F into a Python array V via the CSV library[3].
        iii. Iterate through the codebook V to find the index of the subject ID field. Set this index to variable SubjectIDFieldIndex. The name of this field must be passed in as an argument to this procedure.
        iv. Iterate through each row R in W, where a row R represents a single subject:
            1. Set the SubjectID variable to the following value: M[R][SubjectIDFieldIndex].

---

[2] Canonical variable names were employed to ensure global variable name uniqueness. This measure was necessary because raw variable names were *not* guaranteed to be unique across study files. For example, several study files contained generic variable names such as “Date,” “Week,” or “Value.” Canonical variable names were generated by simply prepending each raw variable’s name with the variable’s source file name. For example, a “Date” variable from the “ASI.sav” file would have a canonical variable name of “”ASI.sav_Date.” This logic successfully ensured global variable name uniqueness within each study. This guarantee was sufficient for the Merge Data stage, as each study’s data was still maintained separately throughout this stage (i.e. separate DataMatrix files were saved for Modafinil-1, Modafinil-2, and Modafinil-3).

[3] Each data file has an associated variable codebook file. Each variable codebook is assumed to have a filename that is deterministically related to the data file’s filename. Throughout this stage, the relationship between data files and codebook files is analogous to the relationship between their representations in IBM SPSS.

2. Iterate through each column C in W, where a column C represents a single variable:
    a. Append the following datapoint to D:
        i. Subject ID=SubjectID
        ii. File Name=F
        iii. Variable Name=V[C]
        iv. data_value=M[R][C]
c. Export the filled DataMatrix D to a file via the Python Pickle serialization library

Algorithm 3: The Merge Data algorithm.

2.1.4 Data Preprocessing - Series Processing Stage

The Series Processing stage served to reformat all raw data into a consistent format. In this study, we aimed to analyze three clinical trial datasets and identify relevant subject subgroups. In order to achieve this goal, we determined that we would need to transform all three study datasets into formats that were mutually consistent; these new data structures could then be efficiently combined and analyzed with our custom machine learning methods. The Series Processing stage served to achieve this goal.

In more practical terms, the Series Processing stage first accepted as input the three DataMatrix files outputted by the Merge Data stage, along with a manually populated configuration spreadsheet for each study. The Series Processing stage then used custom rules defined in the configuration spreadsheet to extract, interpret, reformat, and aggregate the data contained within each DataMatrix. Once the script processed every row in a given input DataMatrix, a new output DataMatrix was saved to a new Pickle file. This new DataMatrix represented the final, processed data for the pertinent study. Accordingly, upon completion of the Series Processing stage, we were left with three final DataMatrix files (i.e. one for each of the three original clinical trial datasets).

The configuration spreadsheets were critical to the aggregation process. Each spreadsheet was designed to rigorously describe an accurate transformation from “Source” (raw) variables to “Output” (processed) variables. Specifically, each spreadsheet contained a single definition row for each Output variable. Each definition row included the following parameters:

1. “Name”: a name for the Output variable.
2. “Source File”: a source datafile for the Source variables. This field was used to instantiate canonical variable names, as described in the previous section.
3. “Source Type”: a field to specify the formatting of the Source variables. Specifically, this field categorized the Source variables as representing long-format data, wide-format data, or long-wide-format data. This field also specified whether the Source variables were temporal in nature (i.e. whether they contained timestamps).
4. “Source Variables”: a comma-separated list of Source variables[4]. This field, along with the “Source File” field, was used to instantiate canonical variable names. The Series Processing script then used the canonical variable names to fetch the source data that would be aggregated into the Output variable.
5. “Timestamp Variables” or a “Timestamp Function”: a field to define the timestamps associated with the source data. This field was only populated for temporal data. If Timestamp Variables were provided, the Series Processing script would generate a one-to-one mapping between Source Variables and Timestamp Variables. Then, whenever the script extracted a Source Variable datapoint for a subject, the associated Timestamp Variable would be accessed to determine the datapoint’s timestamp. Naturally, the configuration spreadsheet was expected to contain an equivalent number of Source Variables and Timestamp Variables. The kth Source Variable was then mapped to the kth Timestamp Variable.

The Timestamp Variables field was used to generate the majority of Output variables. However, a substantial number of temporal Source Variables did not contain explicit timestamps for each measurement and instead used alternative labeling methods. For example, some data was represented by a base timestamp followed by a plurality of measurements. Each measurement had a unique timestamp, but this timestamp was

[4] The Series Processing script referred to the “Source Type” field when interpreting the “Source Variables.” Specifically, if the source type was set to “wide,” the script expected only a single Source variable. This expectation follows intuitively from the definition of wide-format data, as wide-format data represents repeated data points via multiple rows (as opposed to multiple variables).

If the source type was set to “long” or “long-wide,” on the other hand, the script expected more than one Source variable. This expectation is also intuitive, as long-format data represents repeated data points via multiple variables. The Series Processing script must reference all of these variables in order to populate a comprehensive dataset.

not explicitly included in the data. Instead, measurement timestamps could be calculated by a predetermined formula applied to the base timestamp.

These fields taken together precisely defined transformations from Source Variables to Output Variables. These transformations were executed by a custom algorithm which will be provided later. The algorithm served to extract data from each Source Variable, label each temporal datapoint with an accurate timestamp, transform all data into a consistent storage format, and export the aggregated result into a single, multi-study DataMatrix. Before we rigorously describe this algorithm, we must first clarify several details relating to its implementation.

First, observe that the output of this algorithm contained timestamped data. As such, each datapoint contained two values: a measurement and a timestamp. To accommodate this requirement, we created a new DataMatrixEntry data structure. All datapoints in the output DataMatrix were stored as DataMatrixEntry objects. Each DataMatrixEntry contained the following fields:

1. A “type” field to describe the nature of the data. The type field was set to any of the following five values:
    a. TYPE_TEMPORAL_SERIES represented a series of timestamped, longitudinal measurements (e.g. a survey administered once per week).
    b. TYPE_NON_TEMPORAL_SERIES represented a series of non-timestamped measurements (e.g. a list of concomitant medications for a subject).
    c. TYPE_TEMPORAL_SERIES_MULTIPLE_SETS represented a series of timestamped measurements originating from a long-wide format source data.
    d. TYPE_NON_TEMPORAL_SERIES_MULTIPLE_SETS represented a series of non-timestamped measurements originating from a long-wide format source data.
2. A set of timestamp values.
3. A set of measurement values, where each measurement value was associated with a timestamp value.

This DataMatrixEntry storage format allowed for the efficient, memory-conscious storage of both temporal and non-temporal data.

Next, an astute reader may notice that the configuration spreadsheet is somewhat complex. Specifically, recall that the input DataMatrix for the Series Processing stage was a substantially direct copy of the raw SPSS study datafiles. Each variable in the SPSS files had a distinct, direct counterpart in this DataMatrix instance. Thus, if we manually prepared the full configuration spreadsheet, we would need to manually assign thousands of Source Variables and Timestamp Variables. This process would be error-prone, time-consuming, and inefficient.

In order to avoid this tedious process, we decided to implement two improvements. First, we implemented a default configuration for Source Variables. The default configuration was fairly straightforward. Consider a source variable V_S that was *not* explicitly included in the manually-prepared configuration spreadsheet. The Series Preprocessing stage processed this variable based on the following default mapping:

Variable Name: V_s.name
Variable Source File: V_S.source_file
Source Data Type: Long format, timestamped
Source Variables: V_S only
Timestamp Variables: A constant value that is assigned in advance for each study (i.e. “Date” for Modafinil-3).

We designed this default configuration after a careful review of the input DataMatrix. We determined that the default configuration outlined above was appropriate for the majority of the Source Variables. As such, our use of the default configuration allowed us to reduce the size of our configuration spreadsheet by over 50%.

The second improvement related to the identification of Source Variables in the configuration spreadsheet. While preparing the configuration spreadsheet, we noticed that our initial method for identifying Source Variables was inefficient. We required each configuration row to contain the full names of all pertinent Source Variables, even when those names differed only by a single letter or number. This requirement meant that our team had to manually append dozens of effectively redundant Source Variable names to many configuration rows.

In order to address this issue, we implemented support for templated Source Variable names. This functionality allowed our team to identify an arbitrary number of Source Variables with four parameters: a start value, an end value, and a numeric range. These parameters were

then expanded by the Series Preprocessing script to generate a full list of Source Variable names. This expansion was performed via Algorithm 4, as reproduced below and implemented in [17].

1. Repeat for each configuration spreadsheet row R:
    a. If R contains templated Source Variable names:
        i. Extract the following values from R: TemplateStart, TemplateEnd, TemplateRange.
        ii. Create an empty array A.
        iii. Repeat for all integer values I in TemplateRange:
            1. Set string S to the concatenation of the following values: TemplateStart || I || TemplateEnd
            2. Append string S to array A
        iv. For the remainder of the script, interpret array A as the explicit Source Variable list for row R.

Algorithm 4: Template Expansion Algorithm

The transformations were executed by a custom Python script, which can be found in [18]. For reference, a pseudocode description of the Series Processing algorithm is also included as Algorithm 5 below.

1. Repeat for each study S:
    a. Load the input DataMatrix $D_{S,I}$
    b. Initialize an empty output DataMatrix $D_{S,O}$
    c. Load the configuration spreadsheet file C_S
    d. Repeat for each Output Variable V_O in configuration file C_S:
        i. Repeat for each subject X in $D_{S,I}$:
            1. Switch on the variable type V_O.type:
                a. If V_O.type indicates that the Source Variables are in the timestamped wide format:
                    i. Initialize a new DataMatrixEntry E and set the subject ID to X.id, the type to V_O.type, the source file to V_O.source_file, and the name to V_O.name.

ii. Extract the source variable list V_O.source_variables. For each variable v in V_O.source_variables:
    1. Set Y to the value of v for subject X.
    2. Append Y to E.data.
iii. If Timestamp Variables are provided for V_O:
    1. Extract the timestamp variable list V_O.timestamp_variables. For each variable v in V_O.timestamp_variables:
        a. Set Y to the value of v for subject X.
        b. Append Y to E.timestamps.
iv. If a Timestamp Function is provided for V_O:
    1. Calculate the Timestamp Function for each data value. Append the timestamps to E.timestamps.
v. Append E to $D_{S,O}$.

b. If V_O.type indicates that the Source Variables are in the non-timestamped wide format:
    i. Complete the same procedure as in step 1(d)(i)(1)(a), but skip steps 1(d)(i)(1)(a)(iii) through 1(d)(i)(1)(a)(iv).

c. If V_O.type indicates that the Source Variables are in the timestamped long format:
    i. Initialize a new DataMatrixEntry E and set the subject ID to X.id, the type to V_O.type, the source file to V_O.source_file, and the name to V_O.name.
    ii. Obtain the value of Source Variable V_S from the input DataMatrix $D_{S,in}$ for subject X and set it on the data field of E. This value will be an array, with one entry for each row in the original long format data.
    iii. Complete steps 1(d)(i)(1)(a)(iii) through 1(d)(i)(1)(a)(iv).
    iv. Append E to $D_{S,O}$.

d. If V_O.type indicates that the Source Variables are in the non-timestamped long format:
    i. Complete the same procedure as in step 1(d)(i)(1)(c), but skip step 1(d)(i)(1)(c)(iii).
e. If V_O.type indicates that the Source Variables are in the timestamped long-wide format:
    i. Initialize a new DataMatrixEntry E and set the subject ID to X.id, the type to V_O.type, the source file to V_O.source_file, and the name to V_O.name.
    ii. Repeat for each row R for subject X in V_O.source_file:
        1. Complete steps 1(d)(i)(1)(a)(i) through 1(d)(i)(1)(a)(v). Append all data and timestamps to E, as initialized in 1(d)(i)(1)(e)(i).
    iii. Append DataMatrixEntry E to the output DataMatrix D_O
f. If V_O.type indicates that the Source Variables are in the non-timestamped long-wide format:
    i. Complete the same procedure as in step 1(d)(i)(1)(e), but while executing steps 1(d)(i)(1)(a)(i) through 1(d)(i)(1)(a)(v), skip steps 1(d)(i)(1)(a)(iii) through 1(d)(i)(1)(a)(iv).

e. Export the populated DataMatrix D_O to a file via the Python Pickle serialization library

Algorithm 5: The Series Processing algorithm.

2.1 Data Preprocessing - Conclusion

The preprocessing pipeline described above served to fully transform the raw SPSS datafiles that were provided to our team into a single DataMatrix file for each study that contained well-formatted, easily-parsable data. The DataMatrix files were provided as a direct input to our new machine learning algorithm. This algorithm, along with its implementation, is described in the subsequent sections.

2.2 Feature Selection

Our analysis began with feature selection. The original SPSS datafiles contained thousands of distinct variables in aggregate. Feature selection was carried out to identify features that are most relevant to cocaine abstinence; these features were then selected for use in our LCA. If we applied LCA to all of the variables, each LCA run would likely take too long to complete. In addition, our analysis would be prone to overfitting. To prevent both of these issues, we began our analysis process with feature selection.

We used a feature selection method called Temporal Minimum Redundancy-Maximum Relevance (TMRMR) [19]. TMRMR serves to select an optimal set of features such that the inter-correlation between those features is minimized, while the correlation between those features and a target variable is maximized. In our case, the target variable indicated whether the subjects were abstinent from cocaine at various points during the screening, treatment, and follow-up periods. Abstinence was based on the results of Urine Benzoylecgonine tests (UBT).

Additional information regarding the application of this method can be found in [19]. This portion of the project was led by Tan Zhu, with implementation help by Daniel Ruskin.

2.3 Data Normalization

After we selected features from the original set, we proceeded to normalize both our longitudinal and covariate data. Longitudinal data was normalized with the Proportion of Maximum Scaling (POMS) method [19]. This method worked as follows.

For each selected longitudinal feature F, the minimum and maximum values for F were obtained and recorded as F_{min} and F_{max}. Then, all values f were normalized via the following formula (Formula 1) [19].

$$f_{norm} = (f - F_{min}) / (F_{max} - F_{min})$$

Formula 1: Proportion of Maximum Scaling (POMS) feature normalization formula.

Covariates were normalized differently. Age was “normalized into the range [0,1]” [19]. Race was represented via the use of “dummy variables.” Specifically, we included the following two variables for each subject: D_{black} and D_{white}. Both variables could be set to either zero or one. This implies the following race options:

1. D_{black} = 1 and D_{white} = 0: The subject is black.
2. D_{black} = 0 and D_{white} = 1: The subject is white.
3. D_{black} = 0 and D_{white} = 0: The subject is neither white nor black.

A rigorous description of our feature normalization process can be found in [19]. This portion of the project was led by Tan Zhu, with implementation help by Daniel Ruskin.

2.4 Model Fitting

After normalizing our study data, we proceeded to conduct our LCA by fitting a custom model. We used a novel Parallel Latent Growth Mixture (PLGM) for this purpose. Our PLGM model accepted an arbitrary number of longitudinal and covariate features as input. The model then performed trajectory analysis with the goal of discovering "homogeneous subgroups of the aggregated sample" [19]. For our study, we provided the PLGM model with the normalized version of the features selected via the TMRMR process. The output of this model comprised a matrix of the size N by K, where N represented the number of subjects and K represented the number of subgroups[5]. We utilized the open-source OpenMx software package to define and construct our model.

A rigorous definition of our model is provided in [19], along with a description of the method used to simplify and optimize the model. This portion of the project was led by Tan Zhu, with implementation help by Daniel Ruskin.

2.5 Statistical Analysis

After fitting our PLGM model, we proceeded to conduct several statistical tests to determine whether our discovered clusters were meaningful. The following tests were performed:

1. We performed χ^2 tests to determine whether subjects' demographics and baseline measures significantly differed between subgroups. Independent χ^2 tests were performed for discrete variables and Generalized Estimating Equation (GEE) Wald χ^2 tests were performed for continuous variables [19].
2. We performed GEE Wald χ^2 tests to determine whether subjects' longitudinal trajectories significantly differed between subgroups [19].

[5] The number of subgroups is a model parameter that must be set before the optimization process.

3. We performed independent χ^2 tests to determine whether the specific dosage of modafinil significantly affected the cocaine abstinence rate of subjects within each subgroup [19][6]. A separate GEE model was also fitted for subgroups for which a significant effect was discovered with respect to dosage; this model took longitudinal data into account [19].

A data summarization pipeline was created to perform the first two tests described above. The data summarization pipeline was written in MATLAB; the source code can be found in [21]. The pipeline comprised three stages: a DataMatrix Export Stage, a MATLAB Import Stage, and a Table Generation Stage. The DataMatrix Export and MATLAB Import stages were straightforward; they simply served to convert DataMatrix files into a format that could be interpreted by the MATLAB software. The Table Generation Stage was more complex and will now be discussed in detail.

The Table Generation Stage began by iterating through each variable that was provided to the PLGM model. Each variable was then classified into one of three buckets: longitudinal continuous variables, non-longitudinal continuous variables, and non-longitudinal discrete variables. Each variable was then analyzed via the appropriate tests as described earlier in this section. Specifically, GEE Wald χ^2 tests were performed on all continuous variables, while independent χ^2 tests were performed on all discrete variables[7]. GEE Wald χ^2 tests were performed using the GEEQBOX software suite.

The Table Generation Stage aggregated all test results into a single table. The table contained the following values for each test:

1. The name of the tested variable.

---

[6] Subjects were assigned to receive varying doses of modafinil.

[7] Several strategies were used to perform χ^2 tests on discrete variables. For binary variables - variables with only two possible values - we first attempted to perform a GEE Wald χ^2 test. If this test failed to return meaningful results, we then performed a Generalized Linear Model (GLM) Wald χ^2 test instead.

For variables with more than two possible values, we performed a Multinomial Logistic Regression (MNR) Wald χ^2 instead.

Note that the general analysis process was identical between all variables; we first fit a model to the variable data, then analyzed the model coefficients with Wald χ^2 tests to determine statistical significance. Tests differed only by the fitted model.

2. Statistical summaries of the tested variable with respect to each cluster (i.e. mean and standard deviation)
3. Statistical significance values (i.e. χ^2 values and P values)

Additional information regarding the statistical analysis process can be found in [19]. This portion of the project was led by Tan Zhu, with substantive contributions from Daniel Ruskin (i.e. performing many of the above tests with Matlab).

3. Results

Because our study comprised five stages - data preprocessing, feature selection, data normalization, model fitting, and statistical analysis - the study results will be discussed with respect to each of these stages.

The data preprocessing pipeline successfully produced valid DataMatrix files for each study. The datapoints within the DataMatrix were formatted in a consistent manner and were generally suitable for analysis. Accordingly, the DataMatrix files were provided as direct inputs to our custom analysis software.

The feature selection, model fitting, and statistical analysis processes were successfully carried out. The TMRMR process eliminated 4 longitudinal trajectories, which left 13 longitudinal variables for our new PLGM method to analyze [19]. The PLGM model accepted these longitudinal variables and divided the study subjects into three subgroups with distinct trajectory patterns. Our statistical analysis tests revealed that the subgroups had meaningful, significant properties. Specifically, we discovered that the subgroups could be accurately labeled as follows:

Group 1: Subjects with high cocaine use and high alcohol use
Group 2: Subjects with decreasing cocaine use and low alcohol use
Group 3: Subjects with light cocaine use and low alcohol use" [19].

We further discovered that subjects assigned to receive 300mg/day or 400mg/day of modafinil experienced significantly higher improvements in weekly cocaine abstinence rates, as compared to subjects assigned to the placebo group [19].

4. Discussion and Conclusion

Our study helps explain the mixed results from previous studies. Our results have suggested that modafinil may not be equally effective for all patients with cocaine use disorder. Specifically, via a cluster analysis of the aggregated subject sample, we identified a cluster of non-severe cocaine and alcohol users who may respond more significantly to modafinil than more severe users[8]. We believe that more clinical trials will be needed in order to replicate our observations in an independent sample.

As a secondary result, we believe that our data preprocessing pipeline may be useful when designing and conducting retrospective studies of other clinical trials in the future.

[8] Our trajectory analysis helped to characterize this cluster of patients over time.

References

1. National Institute on Drug Abuse, National Institutes of Health & U.S. Department of Health and Human Services.What is cocaine?. Retrieved from https://www.drugabuse.gov/publications/drugfacts/cocaine
2. National Institute on Drug Abuse, National Institutes of Health & U.S. Department of Health and Human Services.What is the scope of cocaine use in the united states?. Retrieved from https://www.drugabuse.gov/publications/research-reports/cocaine/what-scope-cocaine-use-in-united-states
3. Schote, A., Jäger, K., Kroll, S., Vonmoos, M., Hulka, L., Preller, K., . . . Quednow, B. (2018). Glucocorticoid receptor gene variants and lower expression of NR3C1 are associated with cocaine use. Addiction Biology, 24(4), 730.
4. Hasin, D., O'Brien, C., Auriacombe, M., Borges, G., Bucholz, K., Budney, A., . . . Grant, B. (2013). DSM-5 criteria for substance use disorders: Recommendations and rationale. The American Journal of Psychiatry, 170(8), 834.
5. Warner, M., Trinidad, J., Bastian, B., Miniño, A., & Hedegaard, H. (2016). Drugs most frequently involved in drug overdose deaths: United states, 2010–2014. National Vital Statistics Reports Volume 65, Number 10: Centers for Disease Control and Prevention.
6. Dackis, Kampman, Lynch, Pettinati, & O'Brien. (2005). A double-blind, placebo-controlled trial of modafinil for cocaine dependence. Neuropsychopharmacology, 30(1), 205.
7. Dackis, C., Kampman, K., Lynch, K., Plebani, J., Pettinati, H., Sparkman, T., & O'Brien, C. (2012). A double-blind, placebo-controlled trial of modafinil for cocaine dependence. Journal of Substance Abuse Treatment, 43(3), 303.
8. Kampman, K., Lynch, K., Pettinati, H., Spratt, K., Wierzbicki, M., Dackis, C., & O'Brien, C. (2015). A double blind, placebo controlled trial of modafinil for the treatment of cocaine dependence without co-morbid alcohol dependence. Drug and Alcohol Dependence, 155, 105.
9. Ruskin, D.Penn Data Preprocessing Report. Unpublished manuscript.
10. International Business Machines Corporation.Python reference guide for IBM SPSS statistics. Retrieved from ftp://public.dhe.ibm.com/software/analytics/spss/documentation/statistics/22.0/en/server/Manuals/Python_Reference_Guide_for_IBM_SPSS_Statistics.pdf.

11. Ruskin, D.Export data stage source code [computer software]. UConn Github Project: HealthInfoLab/upenn-project, File Location: load_pipeline/export_data.py.
12. Ruskin, D.Parse variables stage source code [computer software]. UConn Github Project: HealthInfoLab/upenn-project, File Location: load_pipeline/parse_variables.py.
13. Machin, J.Xlrd [computer software]. Github Project: python-excel/xlrd.
14. The Python Software Foundation.CSV file reading and writing. Retrieved from https://docs.python.org/3/library/csv.html.
15. The Python Software Foundation.Python object serialization. Retrieved from https://docs.python.org/3/library/pickle.html.
16. Ruskin, D.Merge data stage source code [computer software]. UConn Github Project: HealthInfoLab/upenn-project, File Location: load_pipeline/merge_data.py.
17. Ruskin, D.Configuration generation source code [computer software]. UConn Github Project: HealthInfoLab/upenn-project, File Location: load_pipeline/make_merge_repeated_config.py.
18. Ruskin, D.Merge repeated variables source code [computer software]. UConn Github Project: HealthInfoLab/upenn-project, File Location: load_pipeline/merge_repeated_variables.py.
19. Zhu, T., Ruskin, D., Kampman, K., & Bi, J.Machine learning analysis of aggregated cocaine treatment studies to understand the efficacy of modafinil. Unpublished manuscript.
20. Cephalon, Inc. (2007). Modafinil FDA approved labeling. Retrieved from https://www.accessdata.fda.gov/drugsatfda_docs/label/2007/020717s020s013s018lbl.pdf.
21. Ruskin, D.Statistical Analysis source code [computer software]. UConn Github Project: HealthInfoLab/upenn-project, Folder Location: generate_clustering_tables.