CREATE: CLINICAL RECORD ANALYSIS TECHNOLOGY ENSEMBLE

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Skylar Eglowski

June 2017

COMMITTEE MEMBERSHIP

TITLE:	CREATE: Clinical Record Analysis Technology Ensemble
AUTHOR:	Skylar Eglowski
DATE SUBMITTED:	June 2017
COMMITTEE CHAIR:	Alexander Dekhtyar, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	Foaad Khosmood, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	Maria Pantoja, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	V.S. Subrahmanian, Ph.D. Professor of Computer Science

ABSTRACT

CREATE: Clinical Record Analysis Technology Ensemble

Skylar Eglowski

In this thesis, we describe an approach that won a psychiatric symptom severity prediction challenge. The challenge was to correctly predict the severity of psychiatric symptoms on a 4-point scale. Our winning submission uses a novel stacked machine learning architecture in which (i) a base data ingestion/cleaning step was followed by the (ii) derivation of a base set of features defined using text analytics, after which (iii) association rule learning was used in a novel way to generate new features, followed by a (iv) feature selection step to eliminate irrelevant features, followed by a (v) classifier training algorithm in which a total of 22 classifiers including new classifier variants of AdaBoost and RandomForest were trained on seven different data views, and (vi) finally an ensemble learning step, in which ensembles of best learners were used to improve on the accuracy of individual learners. All of this was tested via standard 10-fold cross-validation on training data provided by the N-GRID challenge organizers, of which the three best ensembles were selected for submission to N-GRID's blind testing. The best of our submitted solutions garnered an overall final score of 0.863 according to the organizer's measure. All 3 of our submissions placed within the top 10 out of the 65 total submissions. The challenge constituted Track 2 of the 2016 Centers of Excellence in Genomic Science (CEGS) Neuropsychiatric Genome-Scale and RDOC Individualized Domains (N-GRID) Shared Task in Clinical Natural Language Processing.

ACKNOWLEDGMENTS

Thanks to:

- Our team's work was supported by US Army Medcom/TATRC grant W81XWH-13-C-0030. The CEGS N-GRID 2016 Shared Task in Clinical Natural Language Processing was sponsored in part by two NIH awards: NIH P50 MH106933 (PI: Isaac Kohane) and NIH 4R13LM011411 (PI: Ozlem Uzuner).
- My fellow team members at SentiMetrix: Alex Dekhtyar, Vadim Kagan, Andrew Stevens, V.S. Subrahmanian and Joshua Terrell
- Kees for their understanding and support throughout this process
- Andrew Guenther for preparing this thesis template.

TABLE OF CONTENTS

Page

LIST OF TABLES ix

LIST OF FIGURES xi

CHAPTER

1 Introduction 1

2 Related Work 4

2.1 Classification 4

2.1.1 Supervised Classification 5

2.1.2 Unsupervised Classification 7

2.2 Text Representation and Word Vectors 8

2.2.1 Bag of Words 8

2.2.2 AutoEncoders 9

2.2.3 Word2Vec 10

2.3 Existing Medical Language Methodologies 12

2.3.1 SentEmotion 12

2.3.2 cTAKES 13

2.3.3 Classification using Association Rules 13

2.3.4 Feature Selection using Association Rules 14

2.3.5 Other Work 14

2.4 Ensemble Construction 14

3 The Challenge 18

4 Overview Of CREATE 24

5 Feature Engineering 26

5.1 Cumulative Scores 27

5.2 Extracting Medications 29

5.3 Emotion Features 29

5.4 Simple Representations of Textual Information 29

5.5 Word2Vec for Textual Information 30

5.6 cTAKES Features 31

5.7 LIWC Features . . . 31
5.8 Common Value Features . . . 32
6 Class Association Rules . . . 34
6.1 Data Preparation . . . 34
6.2 Mining . . . 35
6.3 Pruning . . . 36
7 Feature Selection . . . 39
7.1 Association Rule test . . . 39
7.2 Statistical Tests . . . 39
7.2.1 χ^2 test for categorical features . . . 39
7.2.2 ANOVA F-test for continuous features . . . 40
7.2.3 Mutual Information Gain test (MIG) . . . 40
7.3 Linear SVM Recursive Feature Elimination . . . 41
7.4 Surviving Features . . . 41
8 Classifier Training and Adaptations . . . 43
8.1 Classifier Adaptations . . . 43
8.1.1 Random Forest Regression with Classification Inference (RF-reg-clf) . . . 43
8.1.2 SVM Initialized AdaBoost (SVM-Init-ada) . . . 44
8.2 Data Views . . . 46
8.3 Classifier Training and Evaluation . . . 47
9 Ensemble Learning . . . 57
9.1 Voting Schemes . . . 57
9.1.1 Majority voting . . . 57
9.1.2 Plurality voting . . . 58
9.1.3 Majority_favor_MODERATE voting . . . 58
9.1.4 Plurality_favor_MODERATE voting . . . 58
9.1.5 Simple Round voting . . . 58
9.1.6 Tuned Round voting . . . 58
9.2 Submitted Ensembles . . . 59
9.2.1 Hybrid Random Forest Contribution . . . 64
9.3 Test Results . . . 67

10 Conclusion and Future Work 70
10.1 Enhanced features 70
10.2 Use of Class Association Rules as features 70
10.3 Feature Pruning and Data View construction 71
10.4 Adaptations of classifiers 71
10.5 Tuned Round Voting scheme for ensembles of classifiers 72
10.6 Limitations and Challenges 72
10.7 Computational Costs 72
10.8 Future Work 73
10.8.1 Better Application of Deep Learning Classifiers 75
10.8.2 More Efficient Ensemble Construction 75
10.8.3 Usability 75
BIBLIOGRAPHY 76

LIST OF TABLES

Table		Page
3.1	Overview of released data.	19
3.2	The scale of the target Valence variable in the N-GRID challenge training set data.	19
3.3	Breakdown of features in the original N-GRID 2016 challenge Track 2 dataset by category.	20
3.4	List of free-form text fields found in the original data for Track 2 of the N-GRID 2016 challenge.	21
5.1	A list of approaches to enhancing the feature set for the N-GRID Challenge (Track 2) dataset.	28
6.1	Pruning conditions for Association Rule mining process.	35
6.2	Examples of discovered Association Rules.	37
7.1	Description of the final set of features remaining in our operational dataset after the feature selection (pruning) step.	42
8.1	All the classifiers that were tried as part of the N-GRID Shared Task Challenge.	51
8.2	Different data views used for classifier training.	52
8.3	Abbreviated names for classifiers for the next sections.	52
8.4	Individual Confusion Matrices on the 325 document training set for the 10 best Classifiers (Part 1).	53
8.5	Individual Confusion Matrices on the 325 document training set for the 10 best Classifiers (Part 2).	54
8.6	Multi-class Precision, Recall and MAE on the 325 document training set for the 6 classifiers in the competition-winning ensemble. Per-class MAE is normalized with the assumption that all predictions are maximally incorrect for each class.	55
8.7	Summary metrics on the 325 document training set for each of the 10 best Classifiers. All applicable metrics are macro-averaged when necessary. Higher is better.	56

9.1 Top five voting ensembles. The MA-MAE value is computed on the 325-record training set. 61

9.2 Confusion Matrices on the 325 document training set for the 5 analyzed Ensembles (first 3 were submitted). 62

9.3 Multi-class Precision, Recall and MAE on the 325 document training set for the 3 submitted Ensembles. Per-class MAE is normalized with the assumption that all predictions are maximally incorrect for each class. 63

9.4 Summary metrics on the 325 document training set for each of the 3 submitted Ensembles. All applicable metrics are macro-averaged when necessary. Higher is better. 64

9.5 Effect on Confusion Matrices for substituting RF-reg-clf-full with RF-reg-full in Ensemble A. 65

9.6 Confusion Matrices on the 216 document hidden test set for the 3 submitted Ensembles. 65

9.7 Multi-class Precision, Recall and MAE on the 216 document hidden test set for the 3 submitted Ensembles. Per-class MAE is normalized with the assumption that all predictions are maximally incorrect for each class. 66

9.8 Summary metrics on the 216 document hidden test set for each of the three submitted Ensembles. All applicable metrics are macro-averaged when necessary. Higher is better. 67

10.1 Description of approximate computation times for various parts of the CREATE pipeline. 74

LIST OF FIGURES

Figure		Page
1.1	Architecture of the CREATE Framework.	2
2.1	S.t. $(I' - I)^2$ is minimized.	9
2.2	A Context C_t is the Bag of Words representation of the *window* centered at point t, with w_t omitted.	11
2.3	Skip-Gram: construct f and g such a word w at location t predicts its *context* C'_t. The loss is the difference of C'_t and its actual context C_t.	11
2.4	CBOW: construct f and g such a context C at location t predicts *word* w'_t. The loss is the difference of the predicted w'_t and the actual word w_t.	11
2.5	Decision Tree Construction [28].	16
2.6	Random Forest Construction [32].	17
3.1	Formula for computing Macro-Averaged Mean Average Error (MAMAE).	22
3.2	A synthetic record illustrating the type of clinical medical records data contained in the dataset released for Track 2 of the N-GRID challenge.	23
5.1	How to compute our 5 Common Value Features.	33
6.1	Antecedent and Consequent of a Class Association Rule.	34
6.2	A naive CAR coverage check algorithm as described in [27].	38
9.1	Tuned Round Voting.	69

Chapter 1

INTRODUCTION

When a medical professional diagnoses a patient who is seeking help for mental health-related issues, there are two critical factors: correctness, and timeliness of the diagnosis. The correct diagnosis allows the clinical psychiatrist to devise and implement the appropriate treatment. The timeliness of the correct diagnosis means that the appropriate treatment can start as soon as possible. Correctly assessing the severity of a patient's psychological symptoms poses a challenge with substantial negative consequences if estimated incorrectly. If the severity of a patient's condition is underestimated, the patient will not receive proper treatment, and the condition may deteriorate; if the severity of the condition is overestimated, the patient may be unnecessarily prescribed potentially harmful medications.

Therefore, initial psychiatric evaluations of patients play a crucial role in both the timeliness and the correctness of the diagnosis. Such evaluations often contain a plethora of information, including the patient's mental health history, the family's history of mental conditions, and a detailed report of the patient's present symptoms. Some of this data is naturally generated in a well-structured form: e.g. as a patient's answers to a series of self-assessment survey questions. Other parts of the evaluations come as unstructured text: doctors' notes, patients' verbatim comments, and so on.

Track 2 of the CEGS N-GRID 2016 Shared Task in Clinical Natural Language Processing challenged the participants to analyze the initial psychiatric evaluations of a group of patients for the purpose of predicting the severity of their symptoms. The challenge consisted of a two-month development stage with a labeled training set and a three-day window to submit labels for an unlabeled test set. The training set was composed of real world text and survey information with redacted names and

dates. In this paper, we describe the methods used by our team to win this challenge, by leveraging known natural language processing (NLP) and machine learning methods into a single pipeline we called CREATE (Clinical REcords Analysis Technology Ensemble) shown in Figure **??**.

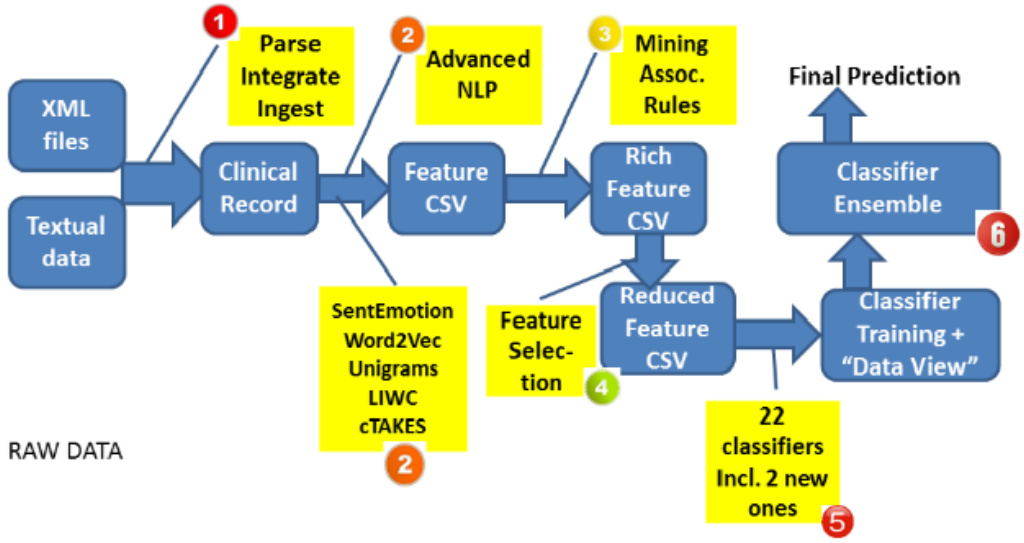

Figure 1.1: Architecture of the CREATE Framework.

The CREATE framework is a machine-learning pipeline that includes several innovations that were developed over the course of project. We start by taking the raw (noisy, cluttered) data provided to N-GRID challenge participants and use a data ingestion phase to ingest and clean it. Second, we apply a host of sophisticated methods to extract a "base" set of features for each clinical record from the ingested data. We expand this set of features via a variety of methods, adding 9263 new features to the original 86. Third, we apply association rule mining to learn approximately 345,000 association rules [20, 37], and then trim them to a set of 628 predictive rules. Though association rules have been widely used in the literature for classification, we use them to generate 628 new features, one for each association rule. Fourth, with the new total of $86 + 9263 + 628 = 9977$ features we engage a set of feature selection operations in order to eliminate non-predictive features. Fifth, we train a total of 22

classifiers, each on seven different subsets of features (data views). Of these, two are novel adaptations of existing Random Forest [21, 10] and AdaBoost [17] classifiers. All of these classifiers utilize the association rule classifiers developed earlier, which is why we call them "stackable" in our framework. On our final step, we train different ensemble classifiers consisting of the subsets of best individual learners in order to improve the final accuracy of detection of patient condition severity. In both the fifth and sixth steps, we do extensive k-fold cross validation and move forward from there to make our final predictions.

This paper is organized as follows. In Section 2 we discuss prior work in this area. Section 3 explains the details of Track 2 of the CEGS N-GRID 2016 Shared Task in Clinical Natural Language Processing (which we, for brevity, refer to as "the N-GRID challenge" throughout the rest of the paper) and provides an overview of the data we have received from the challenge organizers. Section 4 provides an overview of CREATE's six-step approach to the N-GRID challenge. Because Step 1 (ingestion and cleaning) has a straightforward approach, we do not describe it in full detail. Section 5 describes the features we used, while Section 6 describes how we selected just 628 of 345,373 association rules generated by an off-the-shelf association rule mining engine called FP-Growth[20], and by adapting Quinlan's C5 decision tree induction algorithm [36, 37] for association rule mining. Section 7 shows how we determined which features (original and extended) were irrelevant. Section 8 describes how we engaged in an intensive classifier training and hyperparameter optimization procedure on seven different data views of our dataset, which included the adaptations of the well-known Random Forest and AdaBoost classifiers for our purpose. Finally, in Section 9 we describe how we put together a variety of ensembles, made from the best-performing individual learners, to improve the final prediction accuracy. We conclude in Section 10.

Chapter 2

RELATED WORK

Machine Learning another name for the task known as *predictive modeling*: given a set of observations in the past, can you construct a procedure that can identify future observations correctly? There are several different types of classification problems, but the two that we will be focusing on are supervised and unsupervised classification.

2.1 Classification

Classification is the act of associating an *observation* with a pattern [28]. Formally, an observation – typically represented as a collection of text, audiovisual or numerical *features* – is assigned one or more labels. When presented with a new, unlabeled observation, the *classifier* must *infer* or *predict* the most likely label to associate with the observation. For example, if a data scientist were tasked with classifying days as *hot* or *cold*, the number of customers at a local ice cream store could be used as a feature. In general, we'd expect the store to be more busy on hot days, rather than cold days. However, this is not a guarantee – for example, a particularly large birthday party could form an outlier, or perhaps ice cream stores are just more popular on Fridays, even though Fridays are no more likely to be warmer than any other day.

An excellent classifier would be able to identify hot days from cold ones with high *precision* as well as miss very few obviously hot days (known as *recall*). In most tasks, data scientists must balance between both Recall and Precision. In some domains, recall is extremely important. For example, in the medical domain it is much preferred to have false positives while detecting cancer than it is to overlook a

real threat at the risk of a patient's health. On the other hand, some tasks prefer the opposite. A camera that detects if someone has run a red light would prefer to be precise rather than flag innocent drivers.

2.1.1 Supervised Classification

Supervised Classification is focused on identifying membership to one of a set of *predetermined* patterns, typically called a *label* [28]. Supervised learning is called supervised because it is assigned a label by a human that is decided upon before the algorithm is trained. The *training* step will be given each label, and the label must be trustworthy and consistent. Supervised Classification problems, therefore, tend to be somewhat expensive to collect data for. Every data point must be labeled and audited by a human, or even several in order to form a clear consensus.

In the following paragraphs, a brief description will be provided of a variety of the most common supervised machine learning techniques that we used as a part of CREATE.

Naive Bayes – Naive Bayes is one of the simplest machine learning algorithms [28]. During training, the classifier assumes that each feature is independent and marks two pieces of information: the distribution of labels in the training set, and the distribution of values for which a certain value of a certain feature is associated with label. During inference, it computes the probability of a new observation with its previously recorded values and predicts the most likely class.

Decision Trees – Decision Trees are classifiers modeled after a flowchart. The final result of the classifier is a simple of yes/no questions, terminating with a final label at each of its terminal nodes – called *leaves*. Since computing the best possible decision tree is difficult (NP-complete), usually a set of heuristics are employed [28].

To construct the classifier, a recursive algorithm is employed. The base case is

when the input computes only a single class, or if every single attribute has already been analyzed. In that case, the result is a leaf node of the majority class. Otherwise, each remaining attribute [1] is analyzed to see which attribute splits the dataset into two "pure" disjoint sets best. The selected attribute is removed and phrased as a "question" in the final classifier, while each of the disjoint sets are re-evaluated recursively until each sub-problem is terminated with only leaf nodes.

AdaBoost – AdaBoost is a methodology in which an *ensemble* of weak learners are trained in succession; each learner specializing in correcting the errors the previous learner made. AdaBoost traditionally uses fast and weak learners such as Naive Boost or Decision Trees, as the initial weak learners only have to perform slightly better than random in order to eventually converge to a stronger classifier [28].

First, a base classifier is trained. Then, there is a *boosting* step: this step computes all the training instances that were correctly or incorrectly identified. Those which were correctly identified are assigned a weaker level of importance for the successive classifier, while incorrectly classified observations are given a higher level of importance. Finally, the next classifier is trained with the re-weighted dataset [17].

The last two steps are repeated a predetermined number of times, or until a learner cannot improve on a previous learner.

Random Forest – Random Forests is an ensemble of Decision Trees that are trained on random subsamples of the training data [10]. Decision Trees suffer from a few disadvantages. First, for many problems decision trees tend to become very long and complex. This results in the tree overfitting the training data, and building patterns out of random noise; when you classify against out of sample data, it does not generalize very well.

However, you can generally eliminate this overfit without reducing performance by

[1] for very large numbers of attributes, someones only a random sample is analyzed at each level

transforming the Decision Tree into a Random Forest. First, you must determine how many subtrees you would like to train. Then, randomly subsample (with replacement) the data into that many subsets. Finally, construct a Decision Tree with each of those subsamples, with one small change: at each level of the Decision Tree subroutine, only sample a random subset of the features, rather than the entire feature set. This is to provide some entropy for the subtrees so that a handful of dominating features force almost every subtree to look the same. At inference, take the average of all the votes.

Support Vector Machines – Support Vector Machines attempt to identify how to bisect the feature space into two classes. Linear SVMs always bisect with a line in the form of $Ax + b = y$; though there are many *kernels* that can be used to transform the feature space to solve more interesting problems. The observations closest to the bisecting plane are called *support vectors*. In higher dimension problems, the composition of these vectors constructs a bisecting hyperplane. During inference, the label that is returned depends on whether the point is above or below the hyperplane.

2.1.2 Unsupervised Classification

Unsupervised Classification does not have labels, and instead attempts to assign certain patterns to groups, typically called *clusters* [11]. Some algorithms expect hints in the form of the exact number or size of clusters, while others attempt to figure out it out on their own. Unsupervised classification is often used to provide insight for humans, especially for topic modeling [11]. Unsupervised classification can also be used as an automated way to learn compression and feature extraction methodologies, which is what we will focus on in the section 2.2.

2.2 Text Representation and Word Vectors

WordVectors [30] are currently the state-of-the-art method of representing text as features for supervised and unsupervised classification problems. However, the exact benefits of WordVectors are unclear without going over a brief history of simpler means of representation and textual feature extraction.

2.2.1 Bag of Words

The traditional method of text representation was a *Bag of Words*. Bag of Words are unordered, sparse matrices where each column represents a unique term. As the English vocabulary V is large and constantly changing, the first step of many Natural Language Processing tasks would be to scan over the corpus that you want to work with and build a list of each unique term. V' is often smaller than V, but typos, proper names and slang all expand V'. For infinite datasets such as the Internet, one could utilize the *Hashing Trick* [54] which runs each term into a hashing function that computes a random column index. This strategy has a few drawbacks such as colliding terms and having a $|V'|$ that is at least twice as large as your datasets predicted size, but no longer requires an additional pre-processing step.

Some classifiers, such as Naive Bayes [40], work with these very wide and sparse matrices very well. For classifiers that prefer small, dense matrices there are several strategies such as latent Dirichlet Allocation [7] and Singular Value Decomposition [28] that attempt to extract the most frequent and meaningful co-occurring terms and represent them as a single value. Moreover, additional information can be injected by extracting *Parts of Speech* – such as *Noun* or *Adjective* – or constructing the *lemma* of terms – swimming $\rightarrow$ swim.

2.2.2 AutoEncoders

AutoEncoders were originally designed as compression strategies for video and text and were a source of inspiration for WordVectors due to its ability to create graphical features in an unsupervised manner. Graphical data tends to be very high fidelity, and so transferring it over the Internet losslessly is expensive. In addition, small errors and loss of quality are not significant issues for videos that display dozens of frames per second.

Consider an grayscale image matrix I, that is 1000x1000 pixels. To simplify the problem, instead of a traditional RGB channel, each pixel is a float value in the range of $0 \ldots 1$ indicating the grayscale of that pixel. Thus, the image contains 1 million floats if we were to represent it as a naive, dense matrix. Suppose our goal is to achieve 50% compression. We would want to create two functions: f_c and f_d. f_c accepts I and outputs a compressed version C which is sent to the client, which f_d, then decodes and outputs I'. Our algorithm is trained to minimize the *loss* which might be defined as the difference between I and I'. Our black-box AutoEncoder would thus look something like this:

$$f_c(I) => C$$

$$f_d(C) => I'$$

Figure 2.1: S.t. $(I' - I)^2$ is minimized.

Intuitively, there is a trade-off in $|C|$ and the loss. That is, smaller, more-compressed matrices will generally result in a higher loss I'. In a typical system, AutoEncoders are *stacked*. That is, we might have three successive compression functions, each emitting a compressed matrix, followed by three successive decompression functions. The exact implementations of the AutoEncoder is beyond the scope of this

discussion, but they generally rely on a Deep Neural Network [1].

2.2.3 Word2Vec

Mikolov's Word Vector model is one of the greatest recent developments in text classification [30]. Mikolov identified a few weaknesses of the Bag of Words model. First, constructing Bag of Word features requires two passes over the data [2], which can be very costly when working with billions of documents. Second, if one could construct a model that captures all English literature with reasonable precision, then that model can be trained once and then used on nearly every English NLP problem. However, a "Universal English" Bag of Words model would be enormous and contain many unused words for most practical applications. The final and most significant flaw is how Bag of Words treats every word as equally distant from each other. This is intuitively imprecise. Words pairs such as {*lake, swim*} or {*queen, king*} are intuitively related compared to {*evil, throttle*}.

Word vectors solves this by constructing dense, continuous vectors with a limited number of dimensions such that similar words are closer than unrelated words. If vector space representations of *queen* and *king* were close, then we would have at least a partial success at solving the third problem. If our training model is capable of an online training that can handle billions of documents in a reasonable amuont of time, this would go a long way towards addressing the first two flaws.

Mikolov's initial paper described two methods: skip-gram and continuous bag of words (cbow). skip-gram focuses on predicting a word's *context* from the word itself, whereas cbow focuses on predicting a *word* from its context.

Consider w_t which is defined as a word w at location t. w_t's context C_t is defined by all words preceding or succeeding w_t within a window of some parameter n. Now, we

[2] at least, not without using the Hashing Trick, which has its own drawbacks as discussed earlier

define a pair of functions f and g that are analogous to f_c and f_d in an AutoEncoder. f accepts a sparse 1D vector where each column corresponds to a term in the vocabulary, just as it is in the Bag of Words model. f will output a dense vector V analogous to the compressed image C in an AutoEncoder. In skip-gram, f's input is w_t and g's output is C'_t, where the *loss* is the difference between the predicted C'_t and the actual C_t. In *cbow*, f's input is C_t and g's output is w'_t, where the *loss* is the difference between the predicted w'_t and the actual w_t.

$$C_t : [w_{t-N}, \ldots, w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}, \ldots, w_{t+N}]$$

Figure 2.2: A Context C_t is the Bag of Words representation of the *window* centered at point t, with w_t omitted.

$$f(w_t) => V$$

$$g(V) => C'_t$$

Figure 2.3: Skip-Gram: construct f and g such a word w at location t predicts its *context* C'_t. The loss is the difference of C'_t and its actual context C_t.

$$f(C_t) => V$$

$$g(V) => w'_t$$

Figure 2.4: CBOW: construct f and g such a context C at location t predicts *word* w'_t. The loss is the difference of the predicted w'_t and the actual word w_t.

By itself, the *skip-gram* and *cbow* models do not seem to be useful. The key contribution of Word Vectors is understanding that unlike in an AutoEncoder where a human is consuming the final output C', *we can throw away g and its output entirely!* Suppose we want to solve a second classification problem that has labels L_t. We can

replace the original g with a new g' that accepts V and predicts L_t! Thus, if we precompute a comprehensive look-up dictionary of words to V, we are able to greatly simplify the original dataset construction of multiple problems at the same time.

Med2Vec – Building on top of Mikolov's Skip-Gram model, Choi et al. sought to create a deep learning document embedding strategy [13]. They had two datasets of 3 and 5 million documents with a combined total almost 30,000 medical codes that acted as a natural clustering. Med2Vec is constructed in a similar method as Skip-Gram Word2Vec, but treats sequential visits from the same patient as if they were sequence of words in a sentence. Compared to Skip-Gram Word2Vec and GloVe, Med2Vec achieves lower, better normalized Mutual Information Gain scores on Medication and Procedure, implying that it builds embeddings that are better clustered for those tasks [34]. While SVD of document text performed better than Med2Vec, SVD was demonstrated to have far lower interpretability [18].

2.3 Existing Medical Language Methodologies

We limit discussion of existing methologodies to two categories: work on analysis of clinical records in the medical domain, and emerging machine learning and NLP technologies. In addition to these, our work on this project used a wide array of classification [56] and association rule mining [20] techniques, and traditional methods for text parsing and Parts-of-Speech (POS) tagging [46]. We used the Python scikit-learn [32] and nltk [6] toolkits, the Stanford parser and Part of Speech Tagger [46], and the Snowball stemmer for English[8].

2.3.1 SentEmotion

Through its work on past projects [16, 24, 48] SentiMetrix has built an array of technology-based solutions, focusing on the near real-time analysis of large quan-

tities of complex data in multiple languages. SentEmotion [23, 43] is a text-based classification engine developed jointly with psychologists that detects mental health disorders such as *Depression*, *Post-Traumatic Stress Disorder*, and *Traumatic Brain Injury* from patient notes. Leading surveys had identified a set of signals that a victim of depression might have: for example, isolation from others. We built a classification engine on top of the Stanford Parser [46] to extract these signals, and then a second layer to extract both emotions – anger or fear – and symptoms – insomnia or agoraphobia. The final layer generates a confidence value for each of the disorders that COPTADs is configured to recognize.

2.3.2 cTAKES

Clinical Text Analysis and Knowledge Extraction System [44] by Apache is an open-source NLP system that extracts a variety of mental illness signals, physical symptoms, and medication from text.

2.3.3 Classification using Association Rules

B. Liu developed a technique on which to do Classification Based On Association Rules [29]. Liu's main contributions is the notion of a *coverage check* in which rules are sorted in order of their predictive power – *confidence* – and then removes rules that are subsumed by a more powerful rule. At inference time, the first rule in which the antecedent is covered yields its corresponding class. Li et all improves upon this work by raising the minimum coverage of the *coverage check* to 5 subsuming rules, as well as a stronger rule scoring using chi-x^2 tests [27]. In addition, Li suggests storing rules in a trie so that inference can be performed quickly.

2.3.4 Feature Selection using Association Rules

K. Rajeswari continued Liu's work to see if Apriori rule mining can be used select features with high significance, specifically in order to classify the risk of heart disease [38]. First, the authors mined a set of rules with a small k value. Then, they removed all features that did not appear in at least one rule. Next, they re-ran the Apriori but on a much larger k. Finally, when training the final classifier, they included only the features that were an antecedent in at least one rule. The higher k value omits some useful association rules, but should take an order of magnitude less time to complete, due to the large reduction of n candidate rules. The author conclude that this process reduces computation time on their dataset by two full orders of magnitude.

2.3.5 Other Work

Abbe [2] describes different styles of psychiatric NLP and suggests four domains: observational studies, analysis of patient's thoughts and journals, medical records, and published literature. The NGRID challenge falls into the third category, whereas SentEmotion mostly focused on the second. Pestian [35] created an NLP pipeline that could identify whether or not a suicide note was genuine using decision-tree-based classification rules along with AdaBoost and outperformed domain experts by reducing Type II errors by 30%.

2.4 Ensemble Construction

Constructing ensembles was a key step in winning both the N-GRID competition as well as others, such as Kaggle. However, constructing ensemble rules by hand can be time-consuming or miss optimal solutions. Cortes et al. describes an online machine-learning algorithm called ESPBoost that accepts hundreds of potential "ex-

perts" and the correct label [14]. Similar to many other machine learning algorithms, ESPBoost uses coordinate descent to reduce a loss function – in this case, Hamming – to find a local minima without enumerating all possibilities [5, 19]. ESPBoost has been empirically found to work best on large problems with a large number of experts.

D: True Labels

A: Features

T: Tree to recursively build (initially empty)

if *D contains only one class* **then**

  make T a leaf node labeled with the majority class ;

end

else if *A is empty* **then**

  make T a leaf node labeled with the majority class ;

end

else

  (sometimes, only a random sample of attributes are tested)

  $p_0 \leftarrow \mathsf{ImpurityEval1}(D)$;

  for A_i *in* A **do**

    $p_i \leftarrow \mathsf{ImpurityEval2}(A_i, D)$;

  end

  $g \leftarrow \mathsf{argmax}(p_1 \dots p_k)$;

  if $p_0 - p_g < threshold$ **then**

    make T a leaf node labeled with the majority class

  end

  else

    $T_j \leftarrow$ *make T a decision node on* A_g ;

    partition D into disjoint subsets for each value of A_g ;

    for D_j *in partitions* **do**

      $\mathsf{DecisionTree}(D_j, A - A_g, T_j)$;

    end

  end

end

Figure 2.5: Decision Tree Construction [28].

D: True Labels

A: Features

B: Number of subtrees to construct

for i *in* $0 \ldots B$ **do**

$D_i, A_i \leftarrow \mathsf{SubSample}(i,\ D,\ A)$;

$T_i \leftarrow null$;

only sample square-root features in each level of DT

$\mathsf{DecisionTree}(D_i,\ A_i,\ T_i,\ \sqrt{A})$;

$Trees_i \leftarrow T_i$;

end

Figure 2.6: Random Forest Construction [32].

Chapter 3

THE CHALLENGE

The specification of Track 2 of the CEGS N-GRID 2016 Shared Task in Clinical Natural Language Processing (also called the RDoC for Psychiatry Challenge) presented the goal of this particular track of the challenge as:

> *"Determine symptom severity in a domain for a patient, based on information included in their initial psychiatric evaluation. The domain has been rated on an ordinal scale of 0-3. There is one judgment per document, and one document per patient."*[51]

Research Domain Criteria (RDoC) is a framework for facilitating the study of human behavior, both normal and abnormal in various clinical domains. The RDoC provided the data, originally collected by Partners Healthcare Inc. and the Neuropsychiatric Genome-Scale and RDoC Individualized Domains (N-GRID) project at the Harvard Medical School [51]. The data was released to the challenge participants under a strict set of Rules of Conduct and the Data Use Agreement.

As shown in Table 3.1, a total of 649 records were released, broken into a training set of 433 files and a test set of 216 files with no ground truth — the latter released two days before the submissions were due. The initial release of the 433 patient records was broken into two categories: a suggested training set of 325 files, and 108 files called annotated_by_1. Since the contest organizers discouraged us from using the records from the annotated_by_1 set as training set data, we focused most of our efforts on the 325-record training set.

Each record, originally stored in a single XML file, represented the information from the initial psychiatric consultation of a single patient performed by the N-GRID

Table 3.1: Overview of released data.

Total number of records released	649
Number of records in suggested training set	325
Number of records in additional training set	108
Number of records in test set	216

Table 3.2: The scale of the target Valence variable in the N-GRID challenge training set data.

Value	Meaning
0	NONE
1	MILD
2	MODERATE
3	SEVERE

project. For each record in the training set the challenge organizers supplied the ground truth about the severity of the patient's psychiatric condition, called Valence. Table 3.2 shows the scale on which the patients' conditions were evaluated. The judgment contained in the Valence field came from a clinical expert and was based solely on the symptoms and medical, social, mental health, and family history captured in the provided data.

The XML files provided by the challenge organizers contained both structured data, which documented demographic information, mental health history, education, employment, financial status, family history of mental health, medical history, prescription and recreational drug use, and a few other categories of information; along with unstructured, free-form textual data, which documented self-reported symptoms and attending psychiatrists' notes on the patient and their condition. The data was in its raw, originally recorded form; containing numerous typos, conjoined words, missing attributes, inconsistent use of abbreviations, and freeform text.

Table 3.3: Breakdown of features in the original N-GRID 2016 challenge Track 2 dataset by category.

Feature Type	# features	Feature Type	# of features
All features	102		
Demographic information	3	Family History	4
Harming Others Or Self	4	Symptom Denial	8
Mental Health Symptoms	18	Owns Firearms	1
Drug, Caffeine and Alcohol Use	5	Legal History	1
Independence of Daily Activities	9	Appearance	14
Marital Status and Abuse	3	Military History	2
Employment and Finances	4	Mental Health	18
If Underage, Legal Guardian	2	Physical Health	6

Figure **??** shows a *notional (not real) record* created by us to illustrate the nature of the data — it contains no information from the N-GRID dataset. However, this notional record can give the reader an idea of the type of information that the teams had access to while working on the challenge. Actual records contained significantly more data.

Table 3.3 contains a rough breakdown of the types of features found in the original data. Table 3.4 contains the list of features from the original data that were deemed by our team to be free-form text. In the original clinical files, features were informally grouped according to the idiosyncrasies of the RDoC system. The presented high-level feature groups are only given to help the reader understand the typical topics covered in a medical record.

The results of the challenge were evaluated using the a variant of the Mean Absolute Error (MAE) metric. Given a vector $\mathbf{v} = (v_1, \ldots, v_n)$ of ground truth values and a

Table 3.4: List of free-form text fields found in the original data for Track 2 of the N-GRID 2016 challenge.

Free-form Entries	
Childhood History	History of Present Illness and Precipitating Events
Previous Treatments	Prior Medication Side-effects
Current Medication	Chief Complaint (Patient's Own Words)
Interpersonal Concerns	Education
Family Living Situation	Protective Factors
Risk Factors	Actions Taken
Formulation	Level of Care

prediction vector $\mathbf{p} = (p_1, \ldots, p_n)$, the MAE of the prediction is computed as:

$$MAE(\mathbf{p}, \mathbf{v}) = \frac{1}{n} \sum_{i=1}^{n} |p_i - v_i|$$

MAE as described is frequently used in a variety of machine learning tests [55]. However, it has an unbounded maximal value, which can make it unintuitive to reason about. For this reason, the challenge organizes changed the formula such that the score was in the range of [0, 1], where 1 indicated a perfect score. The new MA-MAE (Macro-Averaged Mean Absolute Error) measure was computed by splitting the set of records into four categories (one per ground truth Valence value), computing the MAE for each of the four subsets independently, and combining the computed MAEs into a weighted sum. The normalizing factors for each of the component MAE values are the highest possible errors that can be achieved for a data point with the given Valence value (3 for Valence=0 and Valence=3, and 2 for Valence =1 and Valence = 2). To make the computed value correspond to the *higher is better* intuition, the computed weighted sum was subtracted from 1. The formula for computing the N-GRID Challenge version of MAE is:

$$MA\_MAE(\mathbf{p}, \mathbf{v}) = 1 - \frac{(\frac{MAE(\mathbf{P0},\mathbf{V0})}{3} + \frac{MAE(\mathbf{P1},\mathbf{V1})}{2} + \frac{MAE(\mathbf{P2},\mathbf{V2})}{2} + \frac{MAE(\mathbf{P3},\mathbf{V3})}{3})}{4}$$

Figure 3.1: Formula for computing Macro-Averaged Mean Average Error (MA-MAE).

Every team participating in the challenge was allowed to submit up to three final guesses. Each guess was an XML file containing a single value which was the predicted valence for the specified clinical record from the provided *test set*.

```
RAW DATA
 Name:  John Doe
 Age:  42
 Sex:  Male

Referred by Emergency Services
 Has difficulty remembering if he has taken prescription drugs.
Accidental overdose.
 Referral Notes:  Patient exhibits short-term memory loss
Mixed alcohol with prescription.  Stayed overnight.
Found bruises on shoulders - possibly from falling.

 DEPRESSION: YES
OCD: No
 PANIC: Yes
 Prescriptions:
 Advil (3 times a day)

 Formulation:
 Patient has history of anxiety and bipolar.

 Recommendations:
 Change medication to Alpazolam.
 Require additional visit in 2 weeks.
```

Figure 3.2: A synthetic record illustrating the type of clinical medical records data contained in the dataset released for Track 2 of the N-GRID challenge.

Chapter 4

OVERVIEW OF CREATE

Figure ?? describes the 6 parts of CREATE. We provide brief overviews of the individual components of CREATE below.

1. Data Ingestion - convert the XML data files provided to us into case $\times$ feature matrices that are readily consumed by machine learning pipelines. We limit our discussion of this step to what was presented in Section 3, when we discussed the provided dataset.

2. Feature Extraction - described in Section 5. We started our work on predicting the Valence variable by careful extraction of existing features from the raw XML data provided to us by the organizers. After starting with the features present verbatim (i.e., as unique elements) in the released dataset (see Table 3.3) we defined several other features to generate a single overarching dataset.

3. Development of Association Rule-based Features - described in Section 6. In this stage, we extracted a set of 345,373 Class Association Rules from the above dataset and then eliminated redundant ones to generate a final count of 628. For each retained Class Association Rule we included a binary feature into our augmented dataset.

4. Feature Selection - described in Section 7. We devised a set of tests to identify irrelevant features; a feature that failed all of the tests was eliminated.

5. Classifier Development & Training - described in Section 8. We devised seven different views of our data: each view containing a specific subset of the full set of features. We put together a battery of 22 machine learning algorithms,

including two novel adaptations of Random Forests and AdaBoost. We trained the 22 classifiers on our seven data views and selected the best runs for the ensemble learning step.

6. Ensemble Learning - described in Section 9. On the last step, we evaluated ensembles of best-performing individual classifiers. We used both simple majority/plurality ensemble schemes, as well as more complicated voting techniques to see which, if any, provided the best solutions. At the end, a number of *simple* ensembles over subsets of our classifiers emerged with scores that were clear improvements over the best individual classifiers, and produced MA-MAE scores over 0.86. From those, we selected three predictors that we submitted to the N-GRID challenge organizers. We are proud to report that one of our submissions had the highest overall MA-MAE among the submitted solutions.

Chapter 5

FEATURE ENGINEERING

To analyze the provided data, first we had to transform the original XML data into a tabular, textual format. Each XML file was structured so that it contained all the patient information in a single CDATA block, along with a single tag describing the Valence. Manual examination of several XML files revealed the underlying structure of the patient records (see the synthetic example in Figure **??**). We have previously identified portions of the patient record that we elected to represent as free-form text features (see Table 3.4). These were primarily the restatements of symptoms experienced by the patients recorded from their own words, plus notes and observations of the psychiatrists conducting the evaluations of the patients. Most other content from the XML files are represented as key-value pairs, with both keys and values relatively straightforward to determine and extract.

To transform this XML file into a pipeline-ingestable format, a serious of regular expressions were applied searching for text starting with a special *key*. Pure textual data was assigned its own feature column, but would eventually be concatenated into one feature called text_ALL. Ordinal numbers were simply used as is. Categorical values were handled on a case-by-case basis. Often, there was a limited number of potential values – either text or numerical – and we would map each categorical value to its own boolean feature, doing our best to map inconsistent abbreviation usage to the correct values. Missing data was represented with the value Not a Number. The final result of the extraction process, reduces the 102 features (identifiable in the XML files as individual prompts) to 86 features, which we term the "original" N-GRID dataset features.

The initial breakdown of features is described in Table 3.3. As mentioned above,

not every XML document had values for all of the extracted features; in fact, some features were present only in a handful of records, and other features were often omitted from records. Another data quality issue worth noting is the relative frequency of typos (which could have originated either from the process of digitization of the records, or from the initial medical records themselves). Regular expressions were used to reduce the amount of error in boolean and categorical entries. Some examples include catching different ways to say No: N, Missing or Not. Other expressions simplified synonymous medical codes or shorthand in categorical features, such as ld for a learning disability. For free-form text, no typo detection or conjoined word detection was used.

We have then proceeded to enhance the original N-GRID dataset with a wide range of additional features. Below we discuss the nine different ways in which we augmented our feature set. Table 5.1 contains the summary of our feature enhancement efforts.

5.1 Cumulative Scores

Our initial investigation of the original features extracted from the raw data unveiled groups of related features, typically with "yes"/"no" values, where each individual feature was rarely set to "yes" and no relationship with Valence appeared to exist. Moreover, the overall number of such features set to "yes" in a single patient case history seemed to be in some relationship with Valence. In such cases, we added a new feature: a cumulative score of "yes" values in a group of features, to the dataset.

For example, the original features contained a relatively rich arsenal of substances that a patient could abuse or consume, from readily-available substances such as tobacco, caffeine, and alcohol to a wide range of recreational drugs. We identified all such features, and added a new feature Cumulative_Substance_Use which stored a

Table 5.1: A list of approaches to enhancing the feature set for the N-GRID Challenge (Track 2) dataset.

Approach	Explanation	#
Original (Munged) Features	Original clinical record entries	102
Cumulative Scores	Aggregations of like features	62
Medications	Individual medications taken by patients	47
Association Rules	ARs from features to Valence	628
Unigrams	Representations of textual data	8033
Word2Vec vectors	Representations of textual data	300
SentEmotion	Sentiment and emotion extraction from text	49
cTAKES	Medical symptom tagging	658
LIWC	Topic detection and POS counts	93
Commonality of Patient	A measure of how typical a patient is	5
TOTAL		9977

count of substances which the patient admitted to using. Similar cumulative count features were created for a few more groups of variables: number of psychiatric review conditions deemed positive for the patient, number of "abnormal" items from the mental status exam, number of activities the patient does not perform independently, and more.

The reasoning behind adding such features to the dataset was straightforward: we saw features which appeared to carry important information, but which, due to relative lack of positive/abnormal/out-of-ordinary values, could not individually contribute to the learning of Valence. By creating cumulative count features, we represented the quantitative effects: case histories with more positive/abnormal responses in those feature columns received higher counts. This removed some of the sparsity of the dataset.

5.2 Extracting Medications

To capitalize on the possibility of using medications in predicting Valence, we: (i) manually created a list of 47 medications deemed relevant for patient conditions, complete with alternate spellings, brand names and abbreviations where applicable; (ii) developed a Medication Extractor which analyzed the input data and produced a list of all the medications listed within it; and (iii) created a dataset of medication mentions with 47 columns corresponding to each of the medications our Medication Extractor tool was tracking.

5.3 Emotion Features

SentiMetrix's SentEmotion is a web service, developed as part of the COPTADS project [23, 43] (see also Section 2) that extracts the intensity of emotions such as anger, fear, depression, anxiety, stress, etc. from freeform text. In addition to labeling the overall sentiment of a text fragment [49] and individual emotions expressed in the text (anger, fear, depression, etc), the system outputs a confidence value which expresses the level of confidence the system has in the presence of the emotion. We ran all textual information for each of the records through SentEmotion and added 49 new mental health-related features.

5.4 Simple Representations of Textual Information

At our initial examination of the provided data, we identified a number of features whose contents constituted free-form text. We considered using the free-form text from each of the features as a separate input into any text analysis procedures we were employing. However, in the end, we decided to concatenate the contents of all free-form features into a single free-form text feature, and conduct all text analysis

on it. This resulted in the richest possible text being processed for each of our various patient records.

We investigated a number of different ways to represent textual data in our dataset. The first and most straightforward approach we took was a part of the SentiMetrix Common Pipeline framework for data processing and data ingestion. The steps are as follows:

- Replace dates with a special tag of SMXDATE
- Replace integers with a special tag of SMXNUM
- Stopword removal using the suggested english stopword lists in NLTK [6] and Scikit-Learn [32]
- Stemming using the Snowball Stemmer [8]
- Term-Frequency Inverse-Document Frequency [4, 47] of unigram features for each surviving word stem/term

5.5 Word2Vec for Textual Information

Our second approach used Word2Vec methodology [30] introduced recently by Google. to represent each word found in each freeform text as a vector of 300 features. We used Google's own collection of Word2Vec vectors trained on the Google News corpus and provided by Google [1] . Despite N-GRID data containing many specialized technical terms from the psychiatric domain, and proper names such as names of medications, 96.8% of tokenized text contained in the N-GRID training set was also found in the Google's Word2Vec dataset with a coverage of 78.4% of unique words. Examples of words not covered are typos such as "weopons", "ibuprofin" or "bipolaar";

[1] https://code.google.com/p/word2vec/

conjoined words such as "employment.He"; dates such as "8/17/86"; and medical jargon such as an exact dosage for a patient.

To represent the text from individual patient records, we took the vector representations of each term found in the free-form text in the patient's record, and computed the mean vector. This is the Word2Vec equivalent of the traditional Bag of Words model, and acknowledged as a naive baseline to construct a ParagraphVector by Mikolov and Le [26]. This procedure added 300 features to our dataset. We used gensim to load the binary Word2Vec word-to-vector file [39].

5.6 cTAKES Features

As mentioned in Section 2, Apache cTAKES is a framework for extracting a variety of information from medical records. cTAKES looks for terminology related to medical symptoms, mentions of medications, body parts, procedures, diseases, disorders, and a few other categories of information. For each patient record, we ran the concatenated free-form text extracted from the record through cTAKES to collect these signals.

5.7 LIWC Features

LIWC, Linguistic Inquiry and Word Count [33], is a linguistic computerized text analysis tool similar to SentEmotion. LIWC produces 93 signals, which include various low-level Parts-of-Speech analysis such as the number/frequency of pronouns; semantic features such as if the document has a positive or negative tone; and basic topic-analysis such as detecting if the document focuses on home, money, leisure, the past, or friends. We have run the free-form text extracted from each record, collected all LIWC features, and added them to our dataset.

5.8 Common Value Features

Common Value Features are another form of a cumulative feature, but rather than summarizing logically related features, they summarizes features that individually have little explanatory power. For example most individual observations of a variety of patient behaviors were labeled with the code "WNL" which is interpreted as "within normal limits". In fact, *most* patients had *all* their observations set to "WNL", so a group of "WNL"-valued features formed a very well-defined, *but not very interesting* frequent itemset. These very frequent, but essentially benign itemsets give rise to a large number of useless association rules during the rule generation process. Since an exhaustive mining process on our dataset is extremely slow for any k greater than 4, these very frequent itemsets tended both to consume significant CPU resources while not producing any interesting results.

To reduce the size of our market baskets, we created the concept of a typical value. We set up five separate "commmonality" thresholds: 51%, 62.5%, 75%, 87.5%, and 90%. Given a number t from the list above, and given a feature from our feature set, a specific value of the feature was called *t-common* if more than t percent of all records in the training set contained this value.

We aggregated the notion of *t-commonality* by introducing five *common value features* into the dataset: one per commonality threshold. The common values feature for threshold t was set to the total number of other features in the given record which contained *t-typical* values. See the algorithm described in Algorithm **??** for more detail. These new features allowed us to quickly see whether a specific patient evaluation record yielded rare, atypical, or unusual values for its features.

```
f        : matrix of feature values
output: 5 new common value features

num_common_values ← Array(5) ;
common_values ← Array(RowRank(F), 5) ;
for for each feature column f in F do
    freq ← compute a histogram for f ;
    most_freq_val ← argmax(freq) ;
    for ndx, t in enumerate([0.51, 0.625, 0.75, 0.875, 0.90]) do
        f is common for threshold t ;
        if max(freq) ≥ t then
            num_common_values[ndx] += 1 ;
            for each record ;
            for r in f do
                if r == most_freq_val then
                    common_values[r, ndx] += 1
                end
            end
        end
    end
end
normalize output so it is in 0...1
common_values /= num_common_values
```

Figure 5.1: How to compute our 5 Common Value Features.

Chapter 6

CLASS ASSOCIATION RULES

We decided to see if we could discover some clear dependencies between the features present in (potentially small) subsets of patients, and the value of their Valence. To test this, we engaged in the mining of our feature data for Class Association Rules.

6.1 Data Preparation

We constructed a subset of binary and categorical features found in the data. These primarily included the original features, medication and cumulative features along with boolean features from LIWC, SentEmotion, cTakes. With these, we concentrated on discovery of class association rules of the form:

$$F_1, F_2, \ldots, F_k \longrightarrow \mathsf{Valence},$$

Figure 6.1: Antecedent and Consequent of a Class Association Rule.

where $F_1, \ldots, F_k$ are conditions on the binary/categorical features. Table 6.1 shows the parameters for our Class Association Rule search; we pruned away all rules that did not satisfy them.

Table 6.1: Pruning conditions for Association Rule mining process.

Parameter	Value
Minimal Support	20 records
Minimal Confidence	0.6
Maximal Inverse Confidence	0.4
Maximal Negative Confidence	0.4

6.2 Mining

We used an existing Python implementation [31] of the FP-Growth [1] [20] algorithm to perform an exhaustive search for Class Association Rules with $k \in \{1, 2, 3, 4, 5\}$. For larger values of k ($k = 6 \ldots 9$) we used $C5$ [37, 36], which is non-exhaustive.

The discovered rules went through a rigorous pruning procedure. In addition to pruning away all discovered Class Association Rules (CARs) that did not pass the minimum standards shown in Table 6.1, we also conducted a χ^2 test of significance for each discovered CAR (see Section 7 for a more detailed explanation of the χ^2 tests conducted). All CARs that did not pass the χ^2 test at the significance level of $p = 0.05$ were also eliminated from consideration. Failing the χ^2 test implies that the CAR was a by-product of individual frequencies of the features it contained, rather than an actual meaningful relationship between these features and the Valence variable.

[1] Since the original Python implementation is not actively maintained, SentiMetrix has a private fork of the repository. SentiMetrix's API tweaks allow the emission of only Class Association Rules, rather than all Association Rules; support for aggressive filtering of redundant rules while mining; and utilizes Numpy arrays rather than Python lists for more compact memory allocation and faster cache coherence [52]. The overall improvements result in a modest reduction of memory, and a 33% reduction in run-time. In addition, considering only Class Association Rules reduces the problem size by multiple orders of magnitude. This is significant, because even with these improvements mining higher $k \in \{4, 5\}$ took days to complete.

6.3 Pruning

Finally, we performed a *Coverage Test* as proposed by Li et al [27]. The purpose of the Coverage Test is to reduce the set of CARs to the ones that most accurately describe our data while avoiding excessive duplication. First, we sorted all of our generated CARs by confidence, support and χ^2 score from best to worst. Starting with the first rule, all documents with features in the antecedent of the rule were marked. Then, we advanced onto the second rule and marked all documents with features in the antecedent of that rule. The process was repeated until each input record was covered. After a single document has been marked five times, we removed it from future consideration. If a rule did not *cover* any considered documents, we discarded the rule. Once all documents have been marked five times, we discarded all remaining rules. For more information, see Algorithm **??**.

Altogether, the pruning process reduced the total number of CARs extracted from the data from 345,373 to 628. *For each extracted Class Association Rule, we added one binary feature to the dataset, which was set to 1 on records where the antecedent of the Association Rule applied* [2]. Some examples of the Association Rules we mined during this process are presented in Table 6.2. The first two rules were found by the FP-Growth process, and the third by C5.

[2]The conclusion of the CAR was not considered, as that would result in leaking the label information during training, nor could these features be constructed on a hidden test set

Table 6.2: Examples of discovered Association Rules.

Antecedent	Valence	Support	Conf.	Neg. Conf.
`patient is an inpatient` and `currently undergoing addiction treatment`	SEVERE	22	21/22 (95.5%)	18.25%
`patient suffers from OCD` and `has no history of drug abuse` and `NOT taking Aplenzin`	MOD.	25	20/25 (80%)	22.06%
`patient is NOT inpatient` and `does not drink alcohol` and `is NOT taking Allernaze` and `is NOT taking Levothroid` and `is NOT taking Cultivate` and `is NOT taking Abilify` and `does not suffer from OCD` and `has no history of violence` and `suffers from depression`	MILD	73	67/73 (92%)	38.37%

rules : Sorted set of candidate association rules (best to worst)

observations: Set of observations to cover

k : number of observations to cover an observation

output : A set of covering association rules

```
observation_counts ← Array(Rank(observations)) ;
for rule in rules do
    keep ← false ;
    for observation, count in Zip(observations, observation_counts) do
        if count < k and covers(rule, observation) then
            keep ← true;
            increment the corresponding observation_count ;
        end
    end
    if keep then
        add our current rule to output ;
    end
end
```

Figure 6.2: A naive CAR coverage check algorithm as described in [27].

Chapter 7

FEATURE SELECTION

Because we now had thousands of features to consider, we developed a feature selection procedure that subjected each feature in our dataset to a battery of tests. *Features that failed every single test were eliminated from consideration.* The battery of tests is described in the following sections, followed by a brief analysis of the surviving features.

7.1 Association Rule test

This decision procedure is a simple existence check: *keep a feature if it appears in the antecedent of at least one of the 628 Class Association Rules in our dataset.*

7.2 Statistical Tests

We used 3 statistical tests to filter features. Each of our three methods works best with different feature types and captures different associations with a class labels.

7.2.1 χ^2 test for categorical features

We ran a χ^2 test [59] for each categorical feature against the Valence variable. This test checks whether there are sufficient grounds to believe that a specific categorical feature is associated with another categorical feature purely by coincidence.

Suppose a feature is completely uncorrelated with a valence. This means that the distribution of its categories will be very similar to that of the distribution of the ground truth labels. Now, suppose a **YES** value for a particular categorical

feature always corresponds to a **MILD** valence. If we have sufficient number of **YES** observations, we can compute the likelihood that this happened purely by chance – which would quickly become very low. We rejected any categorical feature whose χ^2 test yielded a p-value higher than 0.05. The χ^2 test was implemented by using scipy's chisquare function to compute the p-value of each categorical feature [22].

7.2.2 ANOVA F-test for continuous features

ANOVA F-tests are used to test the significance of a regression model [9]. While we used the χ^2 test to test for potential significance of our categorical features, we used the multi-way ANOVA F-test for all numeric features. For each feature tested, we separated the data into four subsets, based on the value of the target Valence attribute. We then randomly sampled from these four groups. We then tested the means and standard deviations in each of the four subsets to see if they represented similar or different distributions, and compared them across our multi-way samples to see if there is a statistical bias. Similarly to the χ^2 test, we set rejected any numeric attribute whose ANOVA F-test produced a p-value of more than 0.05. We used scikit-learn's f_classif function to compute the multi-way ANOVA tests [32].

7.2.3 Mutual Information Gain test (MIG)

Mutual Information Gain is typically used in measuring the robustness of clustering methods. In unsupervised problems, MIG is measured by calculating $\mathbf{P(X, Y)}$ - the probability that two variables $\mathbf{X}$ and $\mathbf{Y}$ occur in the same cluster - compared to the probability $\mathbf{P(X) * P(Y)}$ of their occurring in the same cluster by random chance. If there is a clear dependence between the two variables, then the probability of $\mathbf{P(X, Y)}$ will be higher than $\mathbf{P(X) * P(Y)}$. Recent research shows that MIG provides an additional level of feature selection in the context of textual classifica-

tion and clustering [58]. In the case of supervised feature selection, we compare the entropies and distributions of Valence vs. each feature using K Nearest Neighbors. At the time of the N-GRID Challenge, scikit-learn [32] did not have a completed implementation of mutual_info_classif, but it was in the process of being developed. We ported scikit-learn's partial implementation into our system.

7.3 Linear SVM Recursive Feature Elimination

Our final test involved running scikit-learn's version of the Support Vector Machine (SVM) classifier with a linear kernel [15] and observe whether the feature survived the *Recursive Feature Elimination* process implemented within it. An advantage of using a linear SVM to find support vectors is that it provided our system with multivariate feature selection. In addition, χ^2 test and our Class Association Rules only worked on categorical features, while Mutual Information Gain requires a heuristic [58] to operate on continuous features. The Linear SVM recursive feature elimination allowed us an additional test on the continuous features in our dataset.

7.4 Surviving Features

Table 7.1 contains the overview of the features that survived this process: i.e., that passed successfully at least one of the tests from the list above. We make a few observations here about the final shape of the dataset. Only LIWC did not contribute any features. All other means of enhancing non-textual features provided meaningful contributions, with cTakes, original dataset, and, interestingly enough, our cumulative scores accounting for the majority of non-textual features. All five Common Value features also made it. Our manual work on documenting medications resulted in 10 out of 47 medication features kept.

Table 7.1: Description of the final set of features remaining in our operational dataset after the feature selection (pruning) step.

Feature Category	# Features	Feature Category	# Features
TOTAL	788		
Original	30	cTakes	40
Cumulative scores	34	Common Value	5
Medications	10	Word2Vec	34
SentEmotion	6	CAR	628
Unigrams	1		

An unexpected outcome of this process was an essential depletion of directly word related features from the dataset. Only 34 out of 300 Word2Vec dimensions were kept. For non-Word2Vec features, only a single unigram survived – "*other*". This implies that the categorical and yes/no responses have far more predictive power than long form text.

Chapter 8

CLASSIFIER TRAINING AND ADAPTATIONS

For our next step, we have constructed a battery of 22 different classifiers to train on the dataset we built on previous steps. Table 8.1 lists the classifiers we used on this project. 12 of the 22 classifiers came from scikit-learn. Another five classifiers came from the internal SentiMetrix implementations primarily developed prior to the N-GRID challenge, but modified where needed to work with the data from this challenge. Additionally, we used two neural network learners from Google's TensorFlow: their deep neural network implementation; and their so called *deep and wide* classifier, which combines neural nets (deep learning) with Support Vector Machines (wide learning). Finally, two extra classifiers — XGBoost, the boosted gradient classifier [12], and Quinlan's implementation of C5.0 decision tree classifier [1] [37] — were used as well.

8.1 Classifier Adaptations

Of the 22 classifiers we used two, the Random Forest Regression with Classification Inference (RF-reg-clf), and the SVM-initialized Naïve Bayes AdaBoost were novel adaptations of the well-known Random Forest [10, 21] and AdaBoost [17] machine learning techniques. They are described below, and can be found on Table 8.1.

8.1.1 Random Forest Regression with Classification Inference (RF-reg-clf)

Random Forest is a powerful yet forgiving algorithm that can perform a modest amount of feature selection due to its subsampling [10, 21]. In scikit-learn, there are

[1] Due to licensing restrictions, we did not integrate C5.0 into the Common Pipeline Framework

both regression and classification modes of Random Forest [32]. As Valence can be treated as both a class or an ordinal value, we tried both methods. Since regression provides additional insight for the classifier, it often had slightly higher MA-MAE scores. However, in practice regression at inference time biases the kernels right in between MILD (eg, 1.4) and MODERATE (eg, 1.6). The result of this is MILD Valences might be moved to be slightly more MODERATE and vice-versa. When it comes to building the ensemble, this small amount of drift can result in large classification errors if it causes the ensemble's vote to cross a rounding threshold. Our adaptation was to train the Random Forest on the regression version of the problem. Then, during inference, round the inferred value to the nearest Valence.

Performance – As an individual classifier, RF-reg-clf is no different than RF-reg. However, we can compare the performance of the Random Forest Regression versus Random Forest Classification. The regression learner strongly biases the central values, only predicting NONE and SEVERE for the most obvious of cases. On the other hand, the classification learner has better recall scores for every class except MODERATE which it does horribly on. In addition, the classification learner makes more three-degree mistakes than any other of our 10 classifiers. We continue this discussion and how classification inference affects ensemble performance in Section 9.

8.1.2 SVM Initialized AdaBoost (SVM-Init-ada)

Another novel technique we used on this project is the initialization of AdaBoost learning process with SVM (SVM-Init-AdaBoost classifier in our parlance). AdaBoost trains a sequence of estimators one after another [17]. After each iteration, the training set is be reweighed; documents that were just misclassified will have their weight increased, while documents that were just classified correctly will have their weight decreased. This forces the next classifier to correct the mistakes its predecessor

made. While AdaBoost is traditionally done a fast and weaker classifier such as Naive Bayes, any kernel can be used.

In other project, SentiMetrix has had success by introducing a single round of a slow and strong classifier as a seed for AdaBoost [48]. Recall that AdaBoost generally relies on an ensemble of weak classifiers that only need to be slightly better than random to gradually converge. As Support Vector Machines was one of our better classifiers and provides a meaningful decision function that cleanly divides the problem space, it works well as an initial *bootstrapping* classifier to provide an anchor for the successive weak classifier to converge around. We modified the original AdaBoost process (see Section 2.1.1) as follows:

1. Step 1: Train a Linear SVM classifier on input data.
2. Step 2: Analyze kernel decision function to reweigh document weights
3. Step 3...52: Run Naïve Bayes classification 49 times, reweighing document weights after every iteration, and checking for convergence. Reaching convergence before 49 iterations will terminate the process early

On the input N-GRID challenge data, linear kernel SVM produced better accuracy results than a single Naïve Bayes run. This allowed our modification of AdaBoost to start with a sufficiently accurate bootstrap. This additional accuracy gained on the first step has proven to be a core factor in the overall accuracy of this classifier, as one of its runs wound up being the best individual classifier in our battery.

One might point out that if 1 iteration of SVM is good, then wouldn't 50 iterations of SVM to be even better? First, there is a significant cost to training and hyper-parameterizing a Support Vector Machine, compared to Naive Bayes – hours versus seconds. Second, the Support Vector Machine is inserted for "free" by a natural consequence of our process. Since we had already trained and hyper-parameterized

a Support Vector Machine for each view, we only had to train the 49 successive iterations of Naive Bayes in order to build this model. Finally, such a process would be similar to that of the Gradient Boosted Decision Trees implemented in XGBoost [12], which did not perform very well in our data sets. This was partially expected as XGBoost typically requires millions of observations to generalize well.

Performance – Without the initialization of SVM, AdaBoost regressed into a rotating single-class classifier. That is, the very first iteration would see good results, then the second iteration would only predict MILD, the third would only predict MODERATE, the fourth would only predict SEVERE, and the fifth would only predict ABSENT. As none of these classifiers provided any lift as a whole, its performance was identical to that of only the initially trained classifier.

Compared to MNB-CARs, SVM-Init was 0.103 worse at NONE, 0.012 worse at MILD and 0.010 worse at SEVERE in exchange for being 0.171 better at MODERATE. As NONE was a comparatively uncommon class, whereas MODERATE was comparatively difficult, this tradeoff was worth it and resulted in a macro-averaged MAE increase of 0.016 as an individual classifier. The increased performance of MODERATE was particularly crucial during ensemble creation; see Section 9. Please refer to MNB-CARs in Table 8.4 for the exact comparison between MNB-CARs and SVM-Init.

8.2 Data Views

Each of the 22 classifiers was separately trained on nine different *data views* described in Table 8.2. A data view is a collection of features onto which the data is projected prior to being supplied to the classifier. Different subsets of features were selected due to their distinct origins, the hypothesis was if certain minimalist sets of features contain enough information for training the classifiers, and if two classifiers trained on different data views would agree on the Valence of a particular document.

Two of the nine data views listed, ANOVA-wordvector-34 (the 34 Word2Vec features that passed our ANOVA significance tests) and WordVector (all 300 Word2Vec features) yielded abysmal accuracy for all classifiers, and were eliminated from further consideration.

Of the remaining seven data views one, Full, represents the entire collection of features selected during the process described in Section 7, five are its subsets, and one, TF-IDF is the complete set of tf-idf vectors representing the textual portion of each record. The subsets of the Full data view were selected to represent different categories of features (CARs, Numeric[2]) as well as the best features that passed a specific test: χ^2, ANOVA or Multiple Information Gain. We experimented briefly with top 100, 82 [3] and top 75 best features for each of the tests, but settled on top 50, as this provided better accuracy.

At the end of this process we had a total of $22 \times 7 = 154$ trained (classifier, data view) pairs. As a final preprocessing step, we normalized the data view as appropriate for each classifier. For most classifiers, the normalization centered each feature at 0 and scaled it to have unit standard deviation using the interquartile range. For classifiers that cannot use negative numbers, such as Multinomial NB, we did the above normalization and then rescaled the data in the range of $[0, 1]$. This was accomplished with scikit-learn's RobustScaler and MinMaxScaler, respectively [32].

8.3 Classifier Training and Evaluation

For each classifier – data view pair we used 10-fold Cross Validation across the entire training set of 325 data points, using a seeded stratified method provided

[2]The name of this view is a bit of a misnomer, and is kept for historical reasons. This view includes both numeric and categorical features that were present in the original dataset, as well as constructed using cTAKES, SentEmotion, and LIWC toolkits.

[3]82 was the fewest number of significant features found by all three statistical measures – which happened to be chi^2

by scikit-learn. These predictions were eventually fed into the Ensemble Creation procedures.

Scikit-Learn provides some utilities for hyper-parameter selection RandomizedSearchCV and GridSearchCV which allow an engineer to specify a parameter grid which will be either randomly sampled or exhaustively searched, respectively. The sheer number of parametric combinations for some pipelines, such as Bernoulli RBM followed by a SVM, were forced to use RandomizedSearch but GridSearch is preferred otherwise [32].

Towards the end of the competition, we occasionally switched to using a pipeline trained on the 325 records from the training set, and predict the Valence of the 108 annotated_by_1 dataset to make sure that there was no significant over-fitting (in addition to our usage of 10-fold cross validation). For every classifier except XGBoost, scores on the 108 annotated_by_1 files were lower than the the 325 record test set.

Of the 154 total runs, 16 learners who scored above 0.60 [4]. Of those 16, 4 were strictly inferior to other options, leaving us with 12 learners to use in the next step: ensemble training. Of these 12 learners, 10 participated in the five *best* ensembles (see Section 9.) These 10 are shown in Table 8.3 which associates an abbreviated naming convention for each classfier+dataview.

For the sake of brevity, we limit the demonstration and discussion of the results of the individual classifiers to the six classifiers from Table 8.3 which constituted our top performing classification ensemble. These classifiers are:

1. RF-ref-full: the Random Forest regression run on the Full data view.

2. Lin-SVM-chi2-best: Linear kernel SVM classifier run on the 50 best features

[4] the exact threshold is coincidental – there was a large gap between 0.60 and a next-best cluster classifier+dataview combinations of scores around 0.54. Altogether, more than half of classifier+dataview combinations did only slightly better or equal to random

selected by the χ^2 test.

3. RBF-SVR-mig-best: Radial basis function kernel SVM regressor run on the top 50 best features selected by the mutual information gain test.

4. D&W-num: the TensorFlow's Deep and Wide classifier run on the numeric and categorical features.

5. SVM-Init-Ada-CARs: SVM-initialized Adaboost running on our CAR features.

6. MNB-CARs: Multinomial Naïve Bayes running on our CAR features.

Table 8.4 contains the confusion matrices for these six runs, Table 8.6 shows precision, recall and MAE for each class, while Table 8.7 document the overall accuracy metrics: MA-MAE, RoC-AUC, precision, recall, f-score and accuracy. We discuss the work of individual classifiers below.

Our RandomForest regression run on the full data view (RF-reg-full) tended to over-predict MILD and MODERATE valences at the expense of NONE and SEVERE, however, it contained excellent separation between the NONE/MILD, and MODERATE/SEVERE pairs of valences, with only MILD⇒MODERATE false positives being of concern. While this run had the second lowest MA-MAE value out of our six runs, it should be noted (see Section 9) that this is *the only* run that participated in all final ensembles.

The linear kernel SVM classifier running on our top 50 χ^2 features (Lin-SVM-chi2-best) has the third highest MA-MAE and has produced an excellent confusion matrix, with majority of NONE and SEVERE conditions being classified correctly, and with very few "costly" misses.

The SVM regressor with RBF kernel running on our top 50 Mutual Information Gain features (RBF-SVR-mig-best) had the lowest performance of these six runs

(although was still among the better classifiers overall). It over-predicted the MODERATE class, and had some trouble distinguishing MODERATE and MILD valences. It also was very strict at predicting NONE and SEVERE valences.

TensorFlow's Deep and Wide classifier, run on all our numeric and categorical attributes, excluding CAR and Word2Vec attributes (D&W-num) had no significant distinctive features as compared to other runs. It did the worst on properly capturing MILD valences (MILD recall), and tended to admit more "big" mistakes (misclassifications two or more classes apart) than some other methods. But it kept the overall number of misclassified cases reasonable, and earned a MA-MAE in excess of 0.8.

Our overall best single run came from our own AdaBoost classifier trained on a single round of SVM followed by 49 rounds of Naïve Bayes applied to the dataset consisting solely of CAR attributes (SVM-Init-Ada-CARs, see Section 8.1). This classifier excelled almost everywhere, giving by far the most accurate predictions of SEVERE valence and minimizing false positives. The only "weak spot" for this method came from improperly classifying 13 cases with valence of NONE as MODERATE. However, as this was a clear outlier prediction among our six runs (the other runs predicted anywhere from 0 to 4 cases this way), this miss was effectively eliminated in the followup ensembles.

The final classifier run, Multinomial Naïve Bayes run on the same data view of CAR attributes (MNB-CARs), edged the Lin-SVM-chi2-best run by a hair to give us our second best single run MA-MAE of 0.835. It got the largest number of both NONE and SEVERE true positives, as well as tying the Lin-SVM-chi2-best for the largest number of MILD true positives, only stumbling a bit on the MODERATE valence, where it had a very high precision, but low recall.

Table 8.1: All the classifiers that were tried as part of the N-GRID Shared Task Challenge.

No.	Abbreviation	Classifier	Source
1	SVM-Ini	AdaBoost [17] [32]	SentiMetrix
	AdaBoost	1 round Linear-SVM [15], 49 rounds of NB	
2	RF-reg-clf	Train: Regression RF [21];	SentiMetrix
		inference: Classification RF	
3	MI SVR	Mutual-Info [41] Feature Boosted SVM [15]	SentiMetrix
4	CMAR	Classifier on Multiple Assoc. Rules [27]	SentiMetrix
5	CMAR SVM	CMAR-Boosted [27] SVM [15]	SentiMetrix
6	CBA	Classification Based on Associations [29]	SentiMetrix
7	MB NB	Multinomial/Bernoulli Naïve Bayes [32]	scikit-learn
8	Lin-SVM	Linear Kernel SVM [15]	scikit-learn
9	RBF SVM	Radial Basis Function Kernel SVM [32]	scikit-learn
10	LogReg	Logistic Regression [32]	scikit-learn
11	RF	Random Forests [21]	scikit-learn
12	Adaboost NB	AdaBoosted NaiveBayes [17]	scikit-learn
13	KNN	K-Nearest Neighbors [32]	scikit-learn
14	SGD	Stochastic Gradient Descent [32]	scikit-learn
15	BRBM	Bernoulli Restricted Boltzmann Machine [32]	scikit-learn
16	RF-reg	Regression version of Random Forest [21]	scikit-learn
17	Lin-SVM-reg	Regression version of Lin-SVM [15]	scikit-learn
18	RBF SVM-reg	Regression version of RBF SVM [15]	scikit-learn
19	DNN	Deep Neural Network [1]	TensorFlow
20	Deep & Wide	Deep-and-Wide classifier [1]	TensorFlow
21	XGBoost	XGBoost (scalable gradient boosting) [12]	XGBoost
22	C5	C5.0 Decision Tree Classifier [37]	RuleQuest

Table 8.2: Different data views used for classifier training.

No.	Label	Explanation	Size
1	Full	All features that passed filtering	788
2	Numeric	All numeric (and categorical) filtered	125
3	CARs	All CAR features	628
4	TF-IDF	All tf-idf unigram features	8033
5	WordVector	All Word2Vec features	300
6	Chi-square-best50	50 features with highest χ^2 value	50
7	ANOVA-best50	50 features with highest ANOVA F-scores	50
8	MIG-best50	50 features with best MIG values	50
9	ANOVA-wordvector34	Word2Vec features that passed ANOVA	34

Table 8.3: Abbreviated names for classifiers for the next sections.

No.	Name	Algorithm	View
1	MNB-CARs	MNB	CARs
2	RF-full	RF	Full
3	RF-reg-full	RF-reg	Full
4	RF-reg-clf-full	RF-reg-clf	Full
5	Lin-SVM-chi2-best	Lin-SVM	chi2-square-best50
6	Lin-SVM-anova-best	Lin-SVM	ANOVA-best50
7	RBF-SVR-mig-best	RBF-SVR	MIG-best50
8	SVM-Init-Ada-CARS	SVM-Init AdaBoost	Rules
9	D&W-num	Deep & Wide	Numeric
10	DNN-full	DNN	Full

Table 8.4: Individual Confusion Matrices on the 325 document training set for the 10 best Classifiers (Part 1).

MNB-CARs	Pred NONE	Pred MILD	Pred MOD.	Pred SEVERE
True NONE	31	14	0	0
True MILD	16	103	3	8
True MODERATE	5	33	34	10
True SEVERE	1	6	5	56
RF-full	Pred NONE	Pred MILD	Pred MOD.	Pred SEVERE
True NONE	28	15	2	0
True MILD	17	103	9	1
True MODERATE	4	38	25	15
True SEVERE	3	9	18	38
RF-reg-full	Pred NONE	Pred MILD	Pred MOD.	Pred SEVERE
True NONE	13	28	4	0
True MILD	3	94	33	0
True MODERATE	0	16	60	6
True SEVERE	0	4	49	15
RF-reg-clf-full	Pred NONE	Pred MILD	Pred MOD.	Pred SEVERE
True NONE	13	28	4	0
True MILD	3	94	33	0
True MODERATE	0	16	60	6
True SEVERE	0	4	49	15
Lin-SVM-chi2-best	Pred NONE	Pred MILD	Pred MOD.	Pred SEVERE
True NONE	28	13	4	0
True MILD	11	103	13	3
True MODERATE	1	18	41	22
True SEVERE	1	2	21	44

Table 8.5: Individual Confusion Matrices on the 325 document training set for the 10 best Classifiers (Part 2).

Lin-SVM-anova-best	Pred NONE	Pred MILD	Pred MOD.	Pred SEVERE
True NONE	29	14	2	0
True MILD	16	89	17	8
True MODERATE	5	20	36	21
True SEVERE	0	5	21	42
SVM-Init-Ada-CARS	Pred NONE	Pred MILD	Pred MOD.	Pred SEVERE
True NONE	30	2	13	0
True MILD	9	96	21	4
True MODERATE	1	16	58	7
True SEVERE	0	2	14	52
RBF-SVR-mig-best	Pred NONE	Pred MILD	Pred MOD.	Pred SEVERE
True NONE	17	25	3	0
True MILD	10	90	30	0
True MODERATE	1	26	43	12
True SEVERE	1	1	47	19
D&W-num	Pred NONE	Pred MILD	Pred MOD.	Pred SEVERE
True NONE	27	17	1	0
True MILD	13	84	25	8
True MODERATE	3	17	41	21
True SEVERE	1	4	20	43
DNN-full	Pred NONE	Pred MILD	Pred MOD.	Pred SEVERE
True NONE	23	19	2	1
True MILD	13	100	15	2
True MODERATE	7	23	37	15
True SEVERE	1	3	17	47

Table 8.6: Multi-class Precision, Recall and MAE on the 325 document training set for the 6 classifiers in the competition-winning ensemble. Per-class MAE is normalized with the assumption that all predictions are maximally incorrect for each class.

RF-reg-full	Prec.	Recall	MAE	SVM-Init	Prec.	Recall	MAE
NONE	0.812	0.289	0.733	NONE	0.750	0.667	0.793
MILD	0.662	0.723	0.824	MILD	0.828	0.738	0.854
MODERATE	0.411	0.732	0.809	MODERATE	0.547	0.707	0.848
SEVERE	0.714	0.221	0.738	SEVERE	0.825	0.765	0.912
Lin-SVM-chi2	Prec.	Recall	MAE	D&W-num	Prec.	Recall	MAE
NONE	0.683	0.622	0.844	NONE	0.614	0.600	0.859
MILD	0.757	0.792	0.885	MILD	0.689	0.646	0.793
MODERATE	0.519	0.500	0.744	MODERATE	0.471	0.500	0.732
SEVERE	0.638	0.647	0.863	SEVERE	0.597	0.632	0.848
RBF-SVR-mig	Prec.	Recall	MAE	MNB-CARs	Prec.	Recall	MAE
NONE	0.586	0.378	0.757	NONE	0.585	0.689	0.896
MILD	0.634	0.692	0.787	MILD	0.660	0.792	0.866
MODERATE	0.350	0.524	0.745	MODERATE	0.810	0.415	0.677
SEVERE	0.613	0.279	0.739	SEVERE	0.757	0.824	0.902

Table 8.7: Summary metrics on the 325 document training set for each of the 10 best Classifiers. All applicable metrics are macro-averaged when necessary. Higher is better.

Classifier	MA-MAE	ROC-AUC	R^2	
MNB-CARs	0.835	0.773	0.459	
RF-full	0.789	0.707	0.339	
RF-reg-full	0.776	0.683	0.554	
RF-reg-clf-full	0.776	0.683	0.554	
Lin-SVM-chi2-best	0.834	0.765	0.521	
Lin-SVM-anova-best	0.803	0.724	0.384	
RBF-SVR-mig-best	0.757	0.657	0.476	
SVM-Init-Ada-CARs	0.851	0.811	0.515	
D&W-num	0.808	0.721	0.394	
DNN-full	0.809	0.742	0.427	
Classifier	Precision	Recall	F1-Score	Accuracy
MNB-CARs	0.703	0.680	0.673	0.689
RF-full	0.582	0.570	0.567	0.597
RF-reg-full	0.650	0.491	0.495	0.560
RF-reg-clf-full	0.650	0.491	0.495	0.560
Lin-SVM-chi2-best	0.649	0.640	0.644	0.665
Lin-SVM-anova-best	0.585	0.596	0.590	0.603
RBF-SVR-mig-best	0.546	0.468	0.481	0.520
SVM-Init-Ada-CARs	0.738	0.719	0.724	0.726
D&W-num	0.593	0.595	0.593	0.600
DNN-full	0.614	0.606	0.609	0.637

Chapter 9

ENSEMBLE LEARNING

From our prior research [3, 25], we know that ensembles frequently beat vanilla classifiers. As a consequence, we decided to try out ensembles on our data. From our set of 154 classifier data view runs, we selected the 12 best runs (six of which were presented in detail in Section 8). We constructed a variety of ensembles of size 2 to 9 classifiers in each from these runs, and via attrition zeroed in on the best performing ones. Our measure of performance of an ensemble was straightforward:

> *the* MA-MAE *of the ensemble must be higher than* 0.851*, the* MA-MAE *of our best standalone method* (SVM-Init-Ada-CARs).

9.1 Voting Schemes

Our classifier ensembles were constructed in a straightforward way. Each ensemble consisted of a subset of classifiers from our list of 12 best runs. Each classifier in the ensemble received an equal vote share (i.e., we did not attempt to weigh classifiers differently). We devised six different voting schemes to determine the ensemble prediction of the Valence based on the predictions of the constituent classifiers. These voting schemes are defined below.

9.1.1 Majority voting

A value of the Valence is selected if it was predicted by the majority (at least half) of classifiers in the ensemble. If such value does not exist, this method picks the *most common* Valence in the training set[1].

[1] In our training set, this was Valence=MILD.

9.1.2 Plurality voting

Given a parameter *min_votes*, the most common Valence with at least *min_votes* is returned. If such value does not exist, this method picks the *most common* Valence in the training set.

9.1.3 Majority_favor_MODERATE voting

This scheme selects the majority value of predicted Valence if one exists, the same way the *majority* scheme works. However, if a *majority* value does not exist, this voting scheme favors Valence = MODERATE: it selects this value if *at least one* classifier predicts it. If no classifier predicts Valence = MODERATE, this scheme defaults to the most common Valence in the training set.

9.1.4 Plurality_favor_MODERATE voting

This scheme selects the plurality value of Valence if it is predicted by more than *min_votes* votes. If such value does not exist, but at least one classifier predicts Valence = MODERATE, this scheme selects this value. If no Valence = MODERATE prediction exists in the ensemble, the voting scheme defaults to the most common Valence in the training set.

9.1.5 Simple Round voting

This voting scheme simply finds the average prediction value among the ensemble classifiers, and rounds it to the nearest Valence value.

9.1.6 Tuned Round voting

Our most complicated voting scheme offers another layer of hyper-parametrization.

The *simple round voting* scheme assumes that the average valence of 2.6 (out of 3) points to the class Valence = SEVERE, as 2.6 is greater than the midpoint between the numeric Valence scores for MODERATE and SEVERE classes. However, Valence (despite how we choose to treat it on occasion) is **not** a continuous variable, but an ordinal one. Therefore, 0.5, 1.5 and 2.5 *do not have to be* the threshold values separating the neighboring valence classes. What should these values be? Well, we can treat this as yet another hyper-parameter tuning problem, and find such values of the three threshold parameters that optimize the MA-MAE score.

For our experiments we used the following threshold sets, which give rise to a search space of 343 possibilities.

- NONE/MILD threshold: 0.25, 0.3, 0.4, 0.5, 0.6, 0.7, 0.75
- MILD/MODERATE threshold: 1.25, 1.3, 1.4, 1.5, 1.6, 1.7, 1.75
- MODERATE/SEVERE threshold: 2.25, 2.3, 2.4, 2.5, 2.6, 2.7, 2.75

To run all our ensembles through this voting mechanism we would have to generate in excess of 13.3 million combinations. To achieve this, the *tuned round voting* ensemble algorithm was parallelized onto a c4.4xlarge Cloud Instance on Amazon AWS. In addition, the mean votes of each of the 38,760 candidate ensembles were pre-computed using OpenBLAS [57]. Nonetheless, the entire computation took 8 hours with classifier ensembles of no larger than 6, whereas *all* Majority and Plurality schemes up to 9 completed on a single core in less time.

9.2 Submitted Ensembles

Among the multitude of voting ensembles, we selected the five top performers (all providing us with 1.5 – 3% of lift over the best individual classifier) shown in Table

9.1. Notably, all these ensembles used the *tuned voting* ensemble voting, confirming to us that the tuning of the thresholds separating the neighboring Valence classes was a useful procedure. The MA-MAE scores reported in Table 9.1 were computed over the 325-record training set.

As we could only submit three guesses, we had to make our final choices from these five ensembles. *We selected ensembles **A**, **B** and **C** for official submission.*

Ensemble B consisted only of the Random Forest regressor run on the full data view, and our most accurate standalone classifier, SVM-initialized Naïve Bayes AdaBoost on the Class Association Rules data view. It also was the best performer on the training set.

Ensembles A and C were selected to diversify our pool of guesses. Ensemble A was selected as the most accurate ensemble *that did not feature classifiers trained on Class Association Rules alone*. We chose Ensemble C over Ensemble E despite its marginally lower MA-MAE score, because in a secondary run on the 108 annotated_by_1 records Ensemble E had a drop in accuracy that worried us. Additionally, Ensemble C was far better than the other ensembles in properly recognizing the Valence = SEVERE class. Thus, if the withheld test set actually had a large amount of SEVERE Valence scores, we would expect this classifier to perform much better than the other four. In a real-world environment, identifying these SEVERE Valence cases is life-critical.

Table 9.2 shows the confusion matrices of the three submitted ensembles on the 325-record training set. Table 9.3 shows the precision, recall and MAE for each Valence class for each ensemble. Table 9.4 shows the MA-MAE as well as the ROC-AUC, precision, recall, f-measure and accuracy of the ensembles.

Table 9.1: Top five voting ensembles. The MA-MAE value is computed on the 325-record training set.

Name	Classifiers	Voting	MA-MAE
A	RF-full RF-reg-clf-full RF-Reg-full Lin-SVM-chi2-best Lin-SVM-anova-best DNN-full	Tuned round (0.7,1.6,2.25)	0.865
B	RF-reg-full SVM-Init-Ada-CARs	Tuned round (0.7,1.7,2.3)	0.882
C	RF-Reg-full Lin-SVM-chi2-best RBF-SVR-mig-best D&W-num SVM-Init-Ada-CARs MNB-CARs	Tuned round (0.75,1.5,2.25)	0.865
D	RF-reg-full Lin-SVM-chi2-best Lin-SVM-ANOVA DNN-full	Tuned round (0.5,1.3,2.3)	0.850
E	RF-Reg-full SVM-Init-Ada-CARs DNN-full Lin-SVM-chi2-best	Tuned round (0.7,1.4,2.4)	0.866

Table 9.2: Confusion Matrices on the 325 document training set for the 5 analyzed Ensembles (first 3 were submitted).

Ensemble A	Pred NONE	Pred MILD	Pred MOD.	Pred SEVERE
NONE	34	10	1	0
MILD	9	102	17	2
MODERATE	1	15	48	18
SEVERE	0	3	19	46
ENSEMBLE B	Pred NONE	Pred MILD	Pred MOD.	Pred SEVERE
NONE	29	15	1	0
MILD	8	116	4	2
MODERATE	1	20	54	7
SEVERE	0	1	20	47
ENSEMBLE C	Pred NONE	Pred MILD	Pred MOD.	Pred SEVERE
NONE	31	14	0	0
MILD	10	107	10	3
MODERATE	2	21	42	17
SEVERE	0	3	10	55
ENSEMBLE D	Pred NONE	Pred MILD	Pred MOD.	Pred SEVERE
NONE	28	13	4	0
MILD	7	99	24	0
MODERATE	2	14	47	19
SEVERE	0	2	17	49
ENSEMBLE E	Pred NONE	Pred MILD	Pred MOD.	Pred SEVERE
NONE	30	12	33	0
MILD	8	101	21	0
MODERATE	0	12	60	10
SEVERE	0	2	16	50

Table 9.3: Multi-class Precision, Recall and MAE on the 325 document training set for the 3 submitted Ensembles. Per-class MAE is normalized with the assumption that all predictions are maximally incorrect for each class.

ENSEMBLE A	Precision	Recall	MAE
NONE	0.773	0.756	0.911
MILD	0.785	0.785	0.885
MODERATE	0.565	0.585	0.787
SEVERE	0.697	0.676	0.877
ENSEMBLE B	Precision	Recall	MAE
NONE	0.763	0.644	0.874
MILD	0.763	0.892	0.939
MODERATE	0.684	0.659	0.823
SEVERE	0.839	0.691	0.892
ENSEMBLE C	Precision	Recall	MAE
NONE	0.721	0.689	0.896
MILD	0.738	0.823	0.900
MODERATE	0.677	0.512	0.744
SEVERE	0.733	0.809	0.922

Table 9.4: Summary metrics on the 325 document training set for each of the 3 submitted Ensembles. All applicable metrics are macro-averaged when necessary. Higher is better.

Ensemble	MA-MAE	ROC-AUC	R^2	
A	0.865	0.795	0.622	
B	0.882	0.823	0.694	
C	0.865	0.801	0.629	
Ensemble	**Precision**	**Recall**	**F1-Score**	**Accuracy**
A	0.705	0.701	0.703	0.708
B	0.762	0.722	0.738	0.757
C	0.717	0.708	0.709	0.723

9.2.1 Hybrid Random Forest Contribution

As discussed in Section 8.1, our Random Forest Regression with Classification Inference provided a modest amount of lift to our hybrid rounding scheme. We analyze this lift here by substituting RF-reg-clf-full with RF-reg-full in Ensemble A. Table 9.5 compares the confusion matrix that this substitution makes. The substitution actually slightly increases MILD by removing two errors. However, for NONE and MODERATE the error increase is 9 points each! This comes out to be a MA-MAE difference of 0.0318 and worse than just using SVM-Init-Ada-CARs alone. It is quite possible that these errors could have been mitigated by re-running our Tuned Voting process to find more precise breakpoints, so this should be used as an estimation of maximum loss rather than actual loss. The actual loss is guaranteed to be at least 0.015 as Ensemble D was our fourth-best ensemble with a MA-MAE of 0.850.

Table 9.5: Effect on Confusion Matrices for substituting RF-reg-clf-full with RF-reg-full in Ensemble A.

	Pred NONE	Pred MILD	Pred MOD.	Pred SEVERE
NONE	-9	+9	0	0
MILD	0	+2	-2	0
MODERATE	-1	+4	-10	+7
SEVERE	0	+1	-1	0

Table 9.6: Confusion Matrices on the 216 document hidden test set for the 3 submitted Ensembles.

ENSEMBLE A	Pred NONE	Pred MILD	Pred MOD.	Pred SEVERE
NONE	20	10	1	0
MILD	12	66	5	3
MODERATE	2	16	23	5
SEVERE	1	3	10	39
ENSEMBLE B	Pred NONE	Pred MILD	Pred MOD.	Pred SEVERE
NONE	20	11	0	0
MILD	4	68	13	1
MODERATE	2	19	21	4
SEVERE	0	8	7	38
ENSEMBLE C	Pred NONE	Pred MILD	Pred MOD.	Pred SEVERE
NONE	21	10	0	0
MILD	7	64	13	2
MODERATE	2	12	29	3
SEVERE	0	3	9	41

Table 9.7: Multi-class Precision, Recall and MAE on the 216 document hidden test set for the 3 submitted Ensembles. Per-class MAE is normalized with the assumption that all predictions are maximally incorrect for each class.

ENSEMBLE A	Precision	Recall	MAE
NONE	0.571	0.645	0.871
MILD	0.695	0.767	0.867
MODERATE	0.590	0.500	0.739
SEVERE	0.830	0.736	0.881
ENSEMBLE B	Precision	Recall	MAE
NONE	0.769	0.645	0.882
MILD	0.642	0.791	0.890
MODERATE	0.512	0.457	0.707
SEVERE	0.884	0.717	0.855
ENSEMBLE C	Precision	Recall	MAE
NONE	0.700	0.677	0.892
MILD	0.719	0.744	0.861
MODERATE	0.569	0.630	0.794
SEVERE	0.891	0.774	0.906

Table 9.8: Summary metrics on the 216 document hidden test set for each of the three submitted Ensembles. All applicable metrics are macro-averaged when necessary. Higher is better.

Ensemble	MA-MAE	ROC-AUC	R^2	
A	0.837	0.776	0.534	
B	0.833	0.763	0.539	
C	0.863	0.799	0.629	
Ensemble	**Precision**	**Recall**	**F1-Score**	**Accuracy**
A	0.671	0.662	0.664	0.685
B	0.702	0.652	0.671	0.681
C	0.720	0.706	0.712	0.718

9.3 Test Results

The results of running our three submitted ensembles on the 216-record test set are shown in Table 9.6. Table 9.6 shows the confusion matrices of the three ensembles on the test set.

It is worth immediately noting, by comparing confusion matrices in Table 9.6 to those in Table 9.2 (for the training set), that all three ensembles overall performed as hoped and did not overfit the training set by much. Much of the ensemble's MA-MAE improvement was by avoiding misclassifying documents with an error of 2 or more; 10, 11, and 7 data points (respectively) with classification error of 2 or more. This compares with 7, 5, and 8 such data points misclassified on (somewhat larger) training set. This is an expected behavior – in order to misclassify a document with a larger error, many of an ensemble's constituent classifiers must make the same mistake.

All three ensembles exhibited very similar performance on the 31 records with Valence = NONE. Ensemble A tended to underrate MILD cases, preferring to predict Valence = NONE when it made a mistake. Ensembles B and C went in the other

direction, overrating the majority of mistakes on cases with Valence=MILD.

It is on cases with Valence=MODERATE and Valence=SEVERE Ensemble C showed a clearly better performance, both in terms of recall (correctly classifying 29 out of 46 MODERATE cases and 41 out of 53 SEVERE cases), and in terms of precision (keeping it above 50% for Valence=MODERATE, and allowing for only 5 false positives for Valence=SEVERE). These numbers, especially the precision for the Valence=SEVERE class wound up actually being better than the training set results (where Ensemble C had 20 false positives and 55 true positives in this)!

Table 9.7 shows precision, recall and MAE for each valence class for each ensemble. Table 9.8 shows the overall MA-MAE, ROC-AUC, precision, recall, f1-score and accuracy and R^2 metrics for each ensemble.

Ensemble C wound up being the top scorer among our submissions. Ensemble C *was the best overall predictor of patient condition severity* for CEGS N-GRID 2016 Shared Task in Clinical Natural Language Processing (Track 2).

truth : correct Valence scores for each document

predictions: predictions of all trained classifiers to attempt to construct an ensemble for

k : max size of ensemble

thresholds : boundary thresholds to use as triplets

output : The best ensemble from training data

```
best_ensemble ← null ;
best_score ← 0 ;
for i in 2..k do
    for candidate_ensemble in Combinations(predictions, i) do
        predictions ← RowSum(candidate_ensemble) ;
        for none, mild, moderate in thresholds do
            rounded ← [
            SEVERE if v > moderate else
            MODERATE if v > mild else
            MILD if v > none else NONE
            for v in predictions
            ] ;
            score ← MA_MAE(rounded, truth) ;
            if score > best_score then
                best_score ← score ;
                best_ensemble ← ensemble, none, mild, moderate ;
            end
        end
    end
end
```

Figure 9.1: Tuned Round Voting.

Chapter 10

CONCLUSION AND FUTURE WORK

The CREATE framework we built for Track 2 of the CEGS N-GRID 2016 Shared Task in Clinical Natural Language Processing introduces a number of novel features in the field of automated analysis of medical records. The core novel features of CREATE that proved to be crucial to our success included:

10.1 Enhanced features

An aggressive approach to enhancing the initial patient evaluation records provided to us with a multitude of features from diverse sources. Almost all of our feature enhancement efforts contributed non-trivial amounts of features to the final feature set. In addition to traditional features used for medical data analysis, such as diagnosis signals and sentiment, we have added novel categories of features: cumulative scores, commonality features, and medication-use features, which were demonstrated to be statistically significant. In particular, commonality features were significant both in terms of surviving through our rigorous feature selection processes, as well as greatly compacting the search space for association rule mining.

10.2 Use of Class Association Rules as features

Class Association Rules are often used by themselves to classify underlying data. In CREATE we "stacked" the learning processes by using a set of CARs with complete five-fold[1] coverage of our training set as *additional features* in our dataset, and

[1]Meaning that each record in the training set was covered by at least five discovered Class Association Rules.

using both the CARs-only and combined feature sets in subsequent classification and regression tasks.

10.3 Feature Pruning and Data View construction

Our battery of feature pruning tests eliminated a large amount of useless features. In addition, rather than using the full set of features for each classification tasks, we attempted to zero in on useful subsets of the features, either by feature type (all CAR features, all non-CAR features) or by the scores assigned to them by some of our pruning tests (features with highest χ^2, mutual information gain, ANOVA $F-value$). Separation of our data into these data views allowed us to better train our classifiers: the winning ensemble of six classifiers used four out of seven data views. The fact that some of the classifiers in the ensemble were trained on disjoint sets of features helped prevent overfit in the ensemble, because while a single classifier may be biased with a certain data view, the collection of different data views would be less biased.

10.4 Adaptations of classifiers

We adapted two classifiers to better work with the data. The Random Forest regressor with classification inference adaptation was made specifically to account for the nature of the target Valence class attribute and resulted in improved performance of the Random Forest classifier. This regressor was featured in one of the three of our final submissions. The SVM-Initialized AdaBoost outperformed every other individual classifiers-data view in almost every metric and featured prominently the competition-winning ensemble.

10.5 Tuned Round Voting scheme for ensembles of classifiers

While our classifier ensembles were formed in a simple way by giving each classifier an equal vote in each outcome, the *tuned round* voting scheme for deciding the results of the vote, *which was featured in all five best classifier ensembles* was the third "stacked" learner in CREATE: it performed the hyper-parameter tuning to determine the best way to separate averaged (and therefore no longer integer) values between neighboring Valence classes. As seen from Table 9.1, the class thresholds learned by this method were different than the default values in almost all cases, which, by virtue of the method, improved the final accuracy of the ensembles.

10.6 Limitations and Challenges

One key limitation of CREATE is its tightly coupled functionality as a part of the structure of the challenge itself. While no *individual* component was tightly coupled to the domain, the entire pipeline itself was trained on a very specific data format. This makes it somewhat less extensible compared to other NLP systems such as Stanford Parser [46]. Second, both the training and test sets were small; 325 and 216 documents, respectively. Therefore, it is possible that our ensemble took first place only by chance. Nonetheless, we believe that the contributions outlined in the previous section provide lift when applied in the correct domain. In particular, Class Association Rules provide strong story-telling capabilities when results need to be interpreted by a professional.

10.7 Computational Costs

Unfortunately, CREATE has significant implementation and training costs that may limit its applicability compared to simpler models in budget constrained en-

vironments. While some labor costs are unavoidable – such as text ingestion into feature matrices – the training costs of all of different data views combined with all our different classifiers is high. Cross-validation and hyper-parameterization is also a time-consuming process, even if it can be computed in parallel. Complete Class Association Rule mining in particular was one of our slowest stages, and would get even worst with additional data. Recently, some tweaks to FP-Growth have been suggested that allow association rules to be mined on distributed GPUs, despite the sequential process of traditional FP-Growth [53].

In addition, the knowledge that we have gleaned from participation in this contest would inform us on which features to focus on in future clinical record analysis tasks. Table 10.1 describes some of effect that was spent at each stage of the pipeline. The computational complexity of training most stages of the pipeline is $O(n*k)$ where n is the number of documents and k is the maximum of $|features|$ and $|documents|$. The critical path of our pipeline is mining as currently implemented is Class Association Rules.

10.8 Future Work

Improve Embedding Word2Vec and other emerging text embedding NLP strategies have gained a large amount of notoriety since the release of TensorFlow. Although Google's GoogleNews vectors worked surprisingly well despite its apparent non-domain applicability [2], utilizing PubMed's massive database of medical text would be a more domain-aware embedding strategy and training our own PubMedWordVectors would likely increase the amount of topic coherency.

[2] for more information, see Section 5.5

Table 10.1: Description of approximate computation times for various parts of the CREATE pipeline.

Step	Computation Time	Critical Path
Text to Munged CSV	Seconds	-
Feature Expansion	5 Minutes	SentEmotion
(already existing tools)		
WordVectors	20 minutes	Loading Pre-trained Vectors
Class Association Rules	Minutes	(up to k=3)
Class Association Rules	Hours	(up to k=4)
Class Association Rules	3 Days	(up to k=5)
C5.0 CARs	Minutes	(up to k=9)
Feature Selection	Minutes	Mutual Information Gain
Classifier Training	6 Hours	Linear SVM
Ensemble Creation	8 Hours	-

10.8.1 Better Application of Deep Learning Classifiers

A second area of improvement is using of deep learning algorithms such as LSTMs from TensorFlow [1] in an attempt to find convoluted, non-linear feature interactions. Since our work on the N-GRID competition, we have since implemented custom embedding strategies modeled after Word2Vec. If we were to repeat this competition, we would likely attempt to pursue a strategy merging PubMed with Word2Vec to create medical vectors similar to Med2Vec [3] [13]. As a part of the COPTADs project [43], we have since spent some time looking at Word2Vec skip-gram training strategies [30] to model clinical records in lower dimensional space over time.

10.8.2 More Efficient Ensemble Construction

We are currently expending effort comparing online learning strategies using ESP-Boost [14] compared to Tuned Round Voting. Preliminary analysis leads us to believe that ESPBoost works best for datasets in the millions of documents where comprehensive brute force analysis is intractable.

10.8.3 Usability

Finally, the CREATE framework currently exists purely offline and is driven by a command line interface. Developing a more user-friendly and automated pipeline would allow SentiMetrix to more easily extend the applicability of the framework to a larger domain of medical records analysis, as well as to any data analysis tasks that involve large combined structured data and textual data feature sets.

[3] for more information, see Section 2.2.3

BIBLIOGRAPHY

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). Savannah, Georgia, USA*, 2016.

[2] A. Abbe, C. Grouin, P. Zweigenbaum, and B. Falissard. Text mining applications in psychiatry: a systematic literature review. *International journal of methods in psychiatric research*, 2015.

[3] B. An, H. Chen, N. Park, and V. Subrahmanian. Map: Frequency-based maximization of airline profits based on an ensemble forecasting approach. In *Proceedings of the 22nd ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD16)*, pages 421–430, 2016.

[4] R. Baeza-Yates, B. Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[5] J. Bezdek, R. Hathaway, R. Howard, C. Wilson, and M. Windham. Local convergence analysis of a grouped variable version of coordinate descent. *Journal of Optimization Theory and Applications*, 54(3):471–477, 1987.

[6] S. Bird, E. Klein, and E. Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[7] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[8] R. Boulton and M. Porter. Snowball. `http://snowballstem.org`, 2001. Accessed: 2017-04-01.

[9] G. E. Box. Non-normality and tests on variances. *Biometrika*, 40(3/4):318–335, 1953.

[10] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[11] M. E. Celebi et al. *Partitional clustering algorithms*. Springer, 2015.

[12] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2016.

[13] E. Choi, M. T. Bahadori, E. Searles, C. Coffey, M. Thompson, J. Bost, J. Tejedor-Sojo, and J. Sun. Multi-layer representation learning for medical concepts. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1495–1504. ACM, 2016.

[14] C. Cortes, V. Kuznetsov, and M. Mohri. Learning ensembles of structured prediction rules. In *ACL (1)*, pages 1–12, 2014.

[15] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[16] J. P. Dickerson, V. Kagan, and V. Subrahmanian. Using sentiment to detect bots on twitter: Are humans more opinionated than bots? In *Advances in Social Networks Analysis and Mining (ASONAM), 2014 IEEE/ACM International Conference on*, pages 620–627. IEEE, 2014.

[17] Y. Freund, R. Schapire, and N. Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[18] G. H. Golub and C. Reinsch. Singular value decomposition and least squares solutions. *Numerische mathematik*, 14(5):403–420, 1970.

[19] R. W. Hamming. Error detecting and error correcting codes. *Bell Labs Technical Journal*, 29(2):147–160, 1950.

[20] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *ACM Sigmod Record*, volume 29 (2), pages 1–12. ACM, 2000.

[21] T. K. Ho. Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 278–282. IEEE, 1995.

[22] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2017-04-01].

[23] V. Kagan, E. Rossini, and D. Sapounas. *Sentiment Analysis for PTSD Signals*. Springer, 2013.

[24] V. Kagan, A. Stevens, and V. Subrahmanian. Using twitter sentiment to forecast the 2013 pakistani election and the 2014 indian election. *IEEE Intelligent Systems*, 30(1):2–5, 2015.

[25] C. Kang, N. Park, B. A. Prakash, E. Serra, and V. Subrahmanian. Ensemble models for data-driven prediction of malware infections. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 583–592. ACM, 2016.

[26] Q. V. Le and T. Mikolov. Distributed representations of sentences and documents. In *ICML*, volume 14, pages 1188–1196, 2014.

[27] W. Li, J. Han, and J. Pei. Cmar: Accurate and efficient classification based on

multiple class-association rules. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 369–376. IEEE, 2001.

[28] B. Liu. *Web data mining: exploring hyperlinks, contents, and usage data*. Springer Science & Business Media, 2007.

[29] B. L. W. H. Y. Ma and B. Liu. Integrating classification and association rule mining. In *Proceedings of the 4th*, 1998.

[30] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[31] E. Naeseth. Python fp-growth. `https://github.com/enaeseth/python-fp-growth`, 2009. MIT License. Accessed: 2017-04-01.

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[33] J. W. Pennebaker, M. E. Francis, and R. J. Booth. Linguistic inquiry and word count: Liwc 2001. *Mahway: Lawrence Erlbaum Associates*, 71(2001):2001, 2001.

[34] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[35] J. Pestian, H. Nasrallah, P. Matykiewicz, A. Bennett, and A. Leenaars. Suicide note classification using natural language processing: A content analysis. *Biomedical informatics insights*, 2010(3):19, 2010.

[36] J. R. Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[37] R. Quinlan. Rulequest research data mining tools, 2006.

[38] K. Rajeswari. Feature selection by mining optimized association rules based on apriori algorithm. *International Journal of Computer Applications*, 119(20), 2015.

[39] R. Rehurek and P. Sojka. Software framework for topic modelling with large corpora. In *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Citeseer, 2010.

[40] I. Rish. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46. IBM New York, 2001.

[41] B. C. Ross. Mutual information between discrete and continuous data sets. *PloS one*, 9(2):e87357, 2014.

[42] D. L. Roter, R. M. Frankel, J. A. Hall, and D. Sluyter. The expression of emotion through nonverbal behavior in medical visits. *Journal of general internal medicine*, 21(S1):S28–S34, 2006.

[43] A. S. S. Banaszak, V. Kagan and V. Subrahmanian. Coptads:clinical online ptsd and tbi analysis and detection system. In *Proc. Workshop on Visual Analytics in Healthcare'2013*, pages 86–89, 2013.

[44] G. K. Savova, J. J. Masanz, P. V. Ogren, J. Zheng, S. Sohn, K. C. Kipper-Schuler, and C. G. Chute. Mayo clinical text analysis and knowledge extraction system (ctakes): architecture, component evaluation and applications. *Journal of the American Medical Informatics Association*, 17(5):507–513, 2010.

[45] B. Shiner, L. W. D'avolio, T. M. Nguyen, M. H. Zayed, B. V. Watts, and L. Fiore. Automated classification of psychotherapy note text: implications for quality assessment in ptsd care. *Journal of evaluation in clinical practice*, 18(3):698–701, 2012.

[46] R. Socher, J. Bauer, C. D. Manning, and A. Y. Ng. Parsing with compositional vector grammars. In *ACL (1)*, pages 455–465, 2013.

[47] K. Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[48] V. Subrahmanian, A. Azaria, S. Durst, V. Kagan, A. Galstyan, K. Lerman, L. Zhu, E. Ferrara, A. Flammini, and F. Menczer. The darpa twitter bot challenge. *Computer*, 49(6):38–46, 2016.

[49] V. S. Subrahmanian and D. Reforgiato. Ava: Adjective-verb-adverb combinations for sentiment analysis. *IEEE Intelligent Systems*, 23(4):43–50, 2008.

[50] S. M. Tan, P.N. and V. Kumar. *Introduction to Data Mining*. Addison-Wesley Longman Publishing, Boston, MA, USA, 1 edition, 2005.

[51] S. A. F. M. C. T. C. S. K. I. M. T. P. R. S. P. V. U. Uzuner, O. and P. Wang. Announcement of data release and call for participation 2016 cegs n-grid shared-tasks and workshop on challenges in natural language processing for clinical dat. `https://www.i2b2.org/NLP/RDoCforPsychiatry`, 2016. Accessed: 2017-04-01.

[52] S. v. d. Walt, S. C. Colbert, and G. Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[53] F. Wang and B. Yuan. Parallel frequent pattern mining without candidate generation on gpus. In *2014 IEEE International Conference on Data Mining Workshop*, pages 1046–1052, Dec 2014.

[54] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1113–1120. ACM, 2009.

[55] C. J. Willmott and K. Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30(1):79–82, 2005.

[56] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip, et al. Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1):1–37, 2008.

[57] Z. Xianyi. Openblas: an optimized blas library. `http://www.openblas.net`, 2016. Accessed 2017-04-01.

[58] Y. Xu, G. J. Jones, J. Li, B. Wang, and C. Sun. A study on mutual information-based feature selection for text categorization. *Journal of Computational Information Systems*, 3(3):1007–1012, 2007.

[59] F. Yates. Contingency tables involving small numbers and the χ 2 test. *Supplement to the Journal of the Royal Statistical Society*, 1(2):217–235, 1934.