IDSC Internship

Fall 2021

Matthew Rossi

Matthew Rossi

Class of 2023 Foote Fellow
Intern, Systems and Data Engineering
Phone: 305.243.4962 **Office:** Room 600.03, Gables One Tower
Email: matthew.rossi@miami.edu

Matthew is an undergraduate from Northeast Ohio with majors in Mathematics and Computer Science. He intends to pursue an MS in Data Science and hopes to apply the skills he learns to a career as a data scientist in politics, healthcare, or finance.

On campus, Matthew runs an undergraduate political magazine, sits on the executive board of the Data Analytics Student Association, serves as Treasurer of the Racquetball Club, and works as a Student Financial Intern for the Department of Student Activities and Student Organizations.

In his free time, Matthew likes to read, play piano, and go rock climbing.

Preparation

- Stanford CS231n - Convolutional Neural Networks
- Machine Learning Crash Course (MLCC)
- Tensorflow and Pytorch tutorials
- ML experiments
- Scientific/academic papers

Preliminaries

- CITI Program Courses:
  - Conflict of Interest
  - Human Subjects Research
- TRITON Environment
  - Environment configuration
  - Job scheduler
- JupyterHub
- Bitbucket repository
- FileZilla

Python Packages:

- re, pandas, numpy, sklearn, matplotlib, gensim, tensorflow

Pathology Notes Project

Broad Goal: build workflows to extract meaningful, useful data from unstructured cancer pathology reports pulled from the University of Miami's Jackson Memorial Hospital.

Pathology Notes Project

Focused Goals from this semester:

1. data discovery and understanding;
2. data preprocessing and cleaning;
3. building pipelines for classifying documents by diagnosis code as a proof of concept for other classification tasks.

Inspiration

Hierarchical Convolutional Attention Networks for Text Classification

Shang Gao, Arvind Ramanathan, and **Georgia Tourassi**
{gaos, ramanathana, tourassig}@ornl.gov
Computational Science and Engineering Division
Oak Ridge National Laboratory
Oak Ridge, TN, USA

2018

Abstract

machine translation has
self-attention mecha-
in place of recurrent
crease training speed
model accuracy. We
his approach with the
onal filters and a hi-
o create a document
hat is both highly ac-

patterns useful for NLP tasks, especially over long segments of text, they can be slow to train compared to other deep learning architectures – in order to calculate the gradients associated with any given word in a sequence, an RNN must back-propogate through all previous words in that sequence, resulting in backpropogation functions far more complex than those in feedforward or convolutional architectures.

CNNs, traditionally used for computer vision, have also been applied to NLP tasks with notable

Classifying cancer pathology reports with hierarchical self-attention networks

Shang Gao[a,*], John X. Qiu[a], Mohammed Alawad[a], Jacob D. Hinkle[a], Noah Schaefferkoetter[a], Hong-Jun Yoon[a], Blair Christian[a], Paul A. Fearn[b], Lynne Penberthy[b], Xiao-Cheng Wu[c], Linda Coyle[d], Georgia Tourassi[a,*], Arvind Ramanathan[a,*]

[a] Computational Sciences and Engineering Division, Health Data Sciences Institute, Oak Ridge National Laboratory, Oak Ridge, TN, USA
[b] Surveillance Informatics Branch, Division of Cancer Control and Population Sciences, National Cancer Institute, Bethesda, MD, USA
[c] Louisiana Tumor Registry, Louisiana State University Health Sciences Center School of Public Health, New Orleans, LA, USA
[d] Information Management Services Inc, Calverton, MD, USA

ARTICLE INFO

Keywords:
Cancer pathology reports
Clinical reports
Deep learning
Natural language processing
Text classification

2019

ABSTRACT

We introduce a deep learning architecture, hierarchical self-attention networks (HiSANs), designed for classifying pathology reports and show how its unique architecture leads to a new state-of-the-art in accuracy, faster training, and clear interpretability. We evaluate performance on a corpus of 374,899 pathology reports obtained from the National Cancer Institute's (NCI) Surveillance, Epidemiology, and End Results (SEER) program. Each pathology report is associated with five clinical classification tasks – site, laterality, behavior, histology, and grade. We compare the performance of the HiSAN against other machine learning and deep learning approaches commonly used on medical text data – Naive Bayes, logistic regression, convolutional neural networks, and hierarchical attention networks (the previous state-of-the-art). We show that HiSANs are superior to other machine learning and deep learning text classifiers in both accuracy and macro *F*-score across all five classification tasks. Compared to the previous state-of-the-art, hierarchical attention networks, HiSANs not only are an order of magnitude faster to train, but also achieve about 1% better relative accuracy and 5% better relative macro *F*-score.

Their Dataset

- 375k pathology reports.
- Labels for site, laterality, behavior, histology, and grade.
- Manually annotated by human experts.
- Metadata with date, patient ID, tumor ID.

Our Dataset

- 200k pathology report documents.
- Patient ID and ICD-10 diagnosis codes.
- Not necessarily labeled by expert.
- No dates, no tumor ID.
- Each document contains all reports for a patient with the same ICD-10 code.

ICD-10 Codes

International Classification of Diseases - 10th edition

- Used for insurance/billing purposes
- Format: CXX.YYY
  - XX is “top-level classification”
  - YYY is “sub-classification”
- C00-D49: Neoplasms (Tumors)
  - C is malignant, D is benign or unknown behavior
- Ex: **C50.112**

Our Dataset

- 1394 unique diagnoses
  - Average class has only ~140 examples
- 141 unique top-level diagnoses
  - Average class has ~1400 examples
- Shortest document is 971 characters
- Longest document is 8192 characters

Obstacles in Our Dataset

- Class imbalance
- Insufficient data for many classes
- Unreliable ground-truth labels
- Large variation in document length
- Noisy data
- Too many classes / overly broad dataset for its size

Data Management

```
def fetch_subset(data, code, mode='include'):
    """
        data is a pandas table with the following columns: 'c.icd10_after_spilt', 'c.path_notes'
        code is the icd10 code to be included/excluded in the new dataset
        mode indicates whether to include or exclude entries with the specified icd10code

        ex. fetch_subset(data, "C50", mode='include') returns all entries whose icd10 codes contain C50
        fetch_subset(data, "C50.9", mode='exclude') returns all entries that don't contain C50.9
    """
    if mode == 'include':
        return data[data['c.icd10_after_spilt'].str.contains(code)]
    if mode == 'exclude':
        return data[data['c.icd10_after_spilt'].str.contains(code) == False]
```

Allows us to cut datasets that have fewer classes, each with sufficient data

Data Management

```
balance_classes(X, y, max_class_size=None)
```

- Up-samples minority classes to max_class_size.
- By default, max_class_size takes the size of the largest class.
- Takes sparse or dense input.

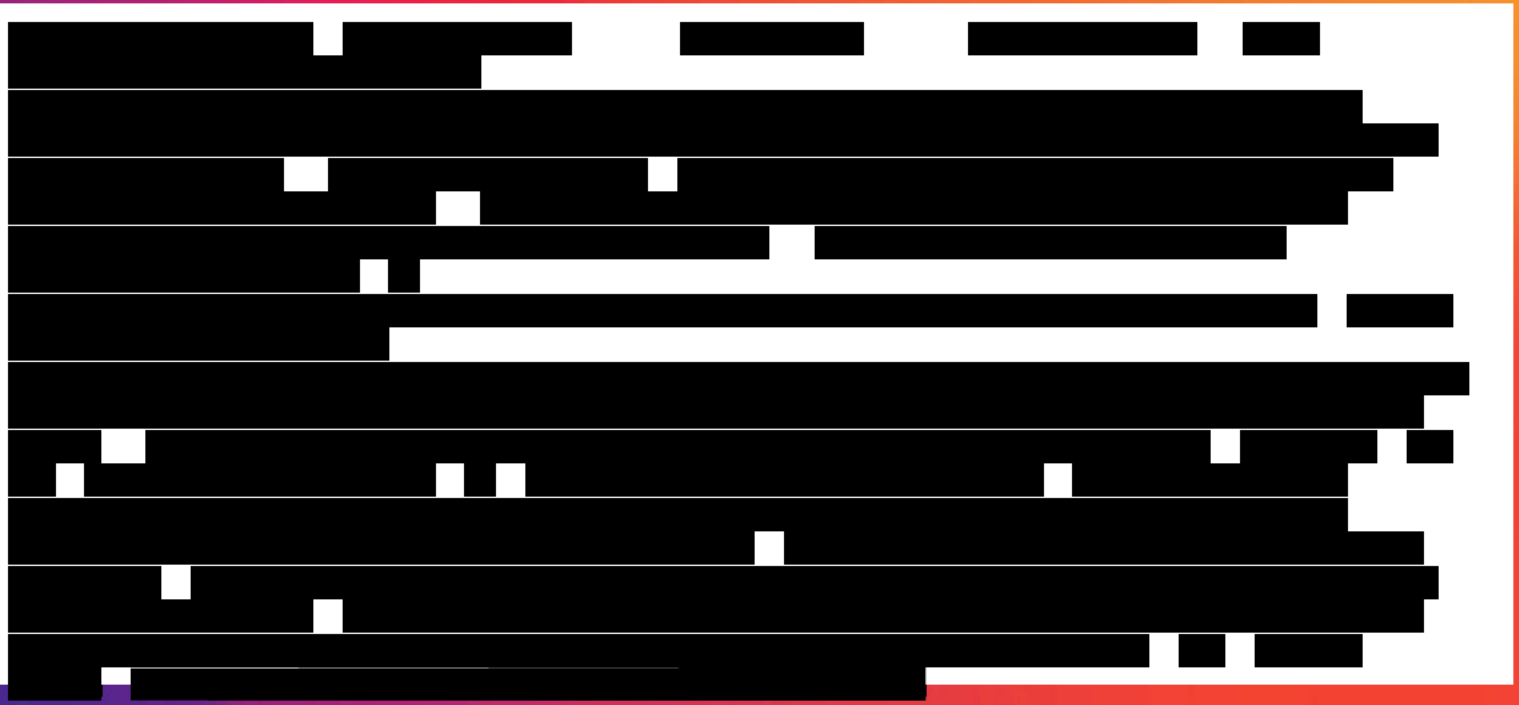
Data Management

```python
from sklearn.utils import resample
import scipy.sparse as sps

def balance_classes(X, y, max_class_size=None):
    sparse = sps.issparse(X)
    if sparse:
        dim = X.shape[1]
        X_res, y_res = sps.csr_matrix([],shape=(1,dim)), np.array([])
    else:
        X_res, y_res = np.array([]), np.array([])
        X = np.array(X)
    y = np.array(np.ravel(y), dtype='int64')

    if max_class_size == None:
        max_class_size = np.max(np.bincount(y))

    cls_list = np.unique(y)
    for cls in cls_list:
        X_cls_res, y_cls_res = resample(X[y==cls], y[y==cls], n_samples=max_class_size, random_state=0)
        if sparse:
            X_res = sps.vstack((X_res, X_cls_res))
        else:
            X_res = np.concatenate((X_res, X_cls_res))
        y_res = np.concatenate((y_res, y_cls_res))

    print("The dataset has %i classes upsampled to %i datapoints." % (len(cls_list), max_class_size))
    if sparse:
        return X_res[1:], np.array(y_res, dtype='int64')
    else:
        return X_res, np.array(y_res, dtype='int64')
```

Natural Language Processing Pipeline

1. **Pre-processing**
   Restructure/reformat data, clean noisy data.
2. **Word Embedding**
   Create numerical representation of categorical language data.
3. **Machine Learning**
   Apply algorithms.

Before



Preprocessing - using re

```
def process(text):
    text = text.lower()
    text = re.sub("dr\.",'dr', text)
    text = re.sub('m\.d\.', 'md', text)
    text = re.sub('a\.m\.','am', text)
    text = re.sub('p\.m\.','pm', text)
    text = re.sub("\d+\.\d+", 'floattoken', text)
    text = re.sub("\.{2,}", '.', text)
    text = re.sub('[^\w_|\.|\?|!]+', ' ', text)
    text = re.sub('\.', ' .', text)
    text = re.sub('\?', ' ? ', text)
    text = re.sub('!', ' ! ', text)
    text = re.sub('\d{3,}', '', text)
    return text
```

Preprocessing - using re

```
def process(text):
    text = text.lower()
    text = re.sub("dr\.",'dr', text)
    text = re.sub('m\.d\.', 'md', text)
    text = re.sub('a\.m\.','am', text)
    text = re.sub('p\.m\.','pm', text)
    text = re.sub("\d+\.\d+", 'floattoken', text)
    text = re.sub("\.{2,}", '.', text)
    text = re.sub('[^\w_|\.|\?|!]+', ' ', text)
    text = re.sub('\.', ' .', text)
    text = re.sub('\?', ' ? ', text)
    text = re.sub('!', ' ! ', text)
    text = re.sub('\d{3,}', '', text)
    return text
```

Lowercase the text.

Remove periods from “dr.”, “m.d.”, “a.m.” and “p.m.”

Replace decimals.

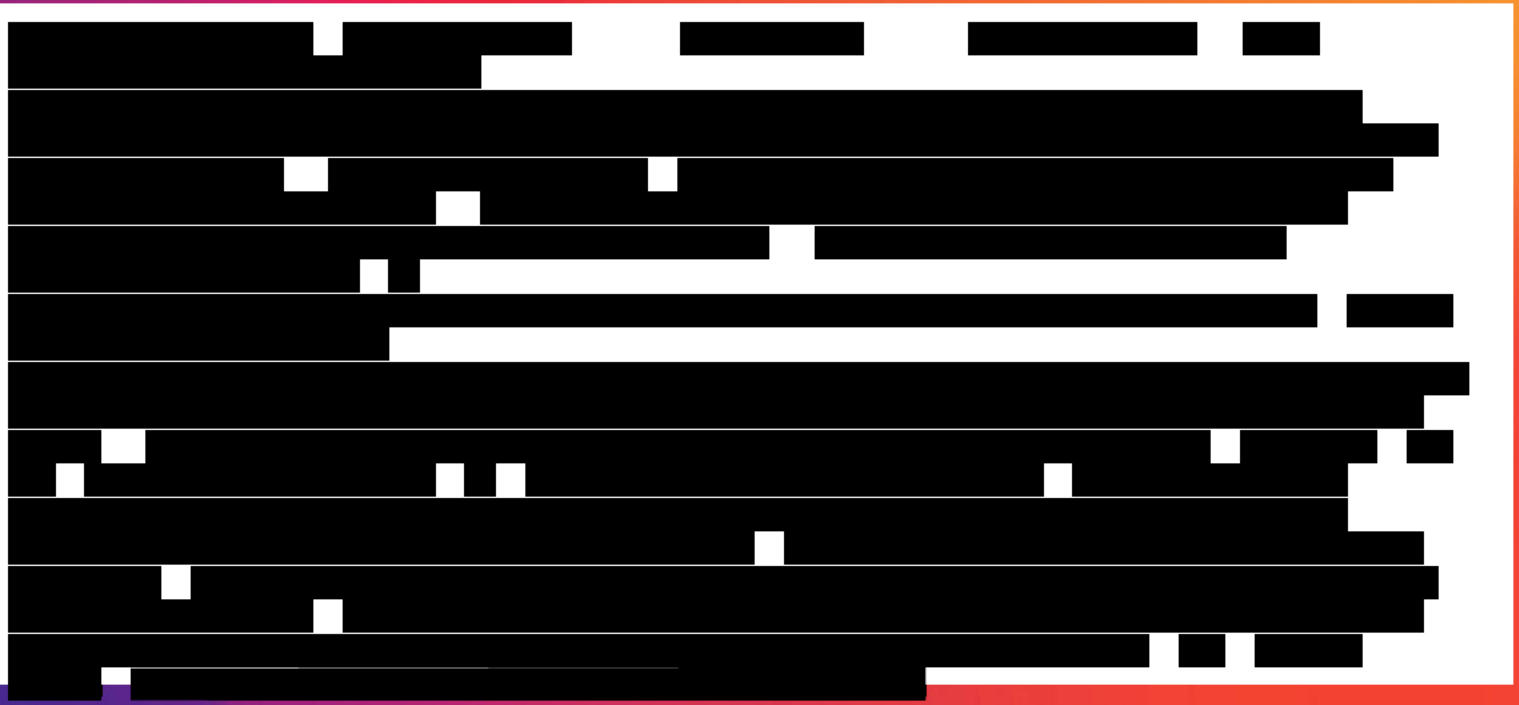
Replace multiple periods with a single period.

Remove symbols (except sentence endings).

Add space around sentence-ending punctuation.

Remove occurrences of three or more consecutive digits.

Before



After

Preprocessing - filtering stop words

- Common words like “the” don’t tell you about the content of a sentence.
- Yet they increase dimensionality and slow down algorithms.

Two sets in our data:

1. Common words in english
2. “Meta”/administrative words particular to this dataset that have nothing to do with a patient’s health, like “telephone”

Preprocessing - filtering stop words

Stop words list:

```
"a", "an", "the", "and", "for", "that", "of", "outside", "out", "with", "in",
"to", "these", "is", "are", "by", "x", "atient", "name", "mrn", "case",
"report", "umhc", "comment", "s", "signed", "electronically", "note", "p",
"f", "m" "fellow", "clinical", "laboratory", "room", "uhealth", "pathology",
"path", "client", "fax", "phone", "telephone", "md", "dr", "phd", "mba",
"pathologist", "attending", "i", "ii", "iii", "department", "received", "sw",
"nw", "name", "dob", "gender", "medical", "director", "demographics", "ave",
"surgical", "physician", "location", "final", "see", "description",
"provided", "status", "ordered", "addendum", "patient", "resident",
"examination", "issuance", "involved", "history", "diagnosis", "release"
```

Preprocessing

	No Pre-processing	Just re	Re & stop word filtering
Average # of characters / report	5886.5	4131.1	2959.0
Average # of words / report	906.27	696.30	472.08

Label Processing

From sklearn:

- LabelEncoder()
  - Converts categorical label to integer
- LabelBinarizer()
  - Converts to one-hot encoding

Word Embedding

TFIDF Vectorization:

- **Term frequency** = (Number of times term t appears in a document) / (Total number of terms in the document)
- **Inverse document frequency** = ln(Total number of documents / Number of documents with term t in it)
- Score for t is the product of these quantities

Outputs a sparse matrix of shape (n_datapoints, vocab_size)

- row = document
- element = score for term t for that document

Word Embedding

Word2Vec:

- Neural network approach.
- Learns associations between a word and the words that appear around it.
- Generates a vector embedding of a fixed length for each word in the vocabulary.

Outputs a dense matrix of shape (vocab_size, embedding_dimension)

- row = word

Machine Learning

A. Unsupervised Learning (Clustering)

Machine Learning

B. Traditional/Classical ML

Traditional ML

Multinomial Naive Bayes

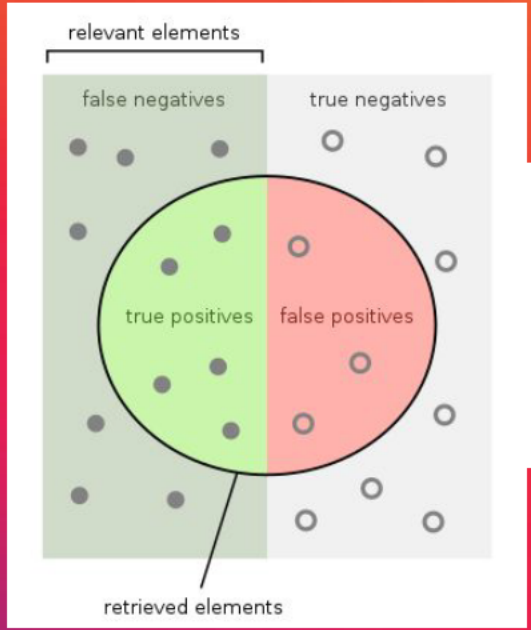
Logistic Regression

Random Forest Classifier

Support Vector Classifier

Evaluating Models

- Issues with plain accuracy
- Precision and Recall
- Macro F1 score



How many retrieved items are relevant?

Precision =

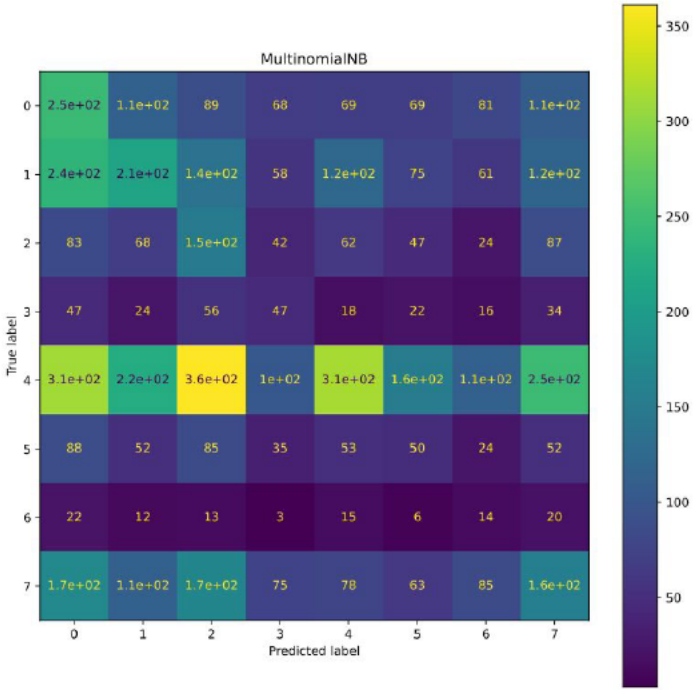
How many relevant items are retrieved?

Recall =

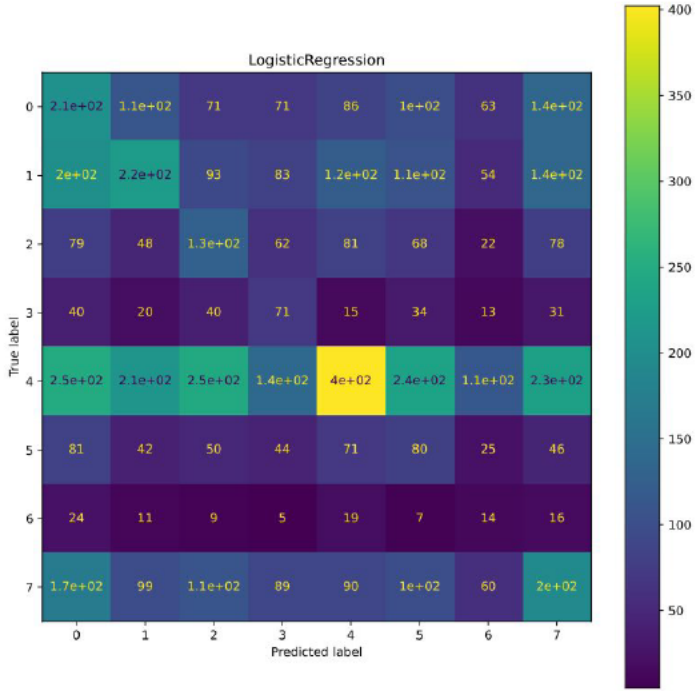
Results

Classifier	Macro F1 Score
Multinomial Naïve Bayes	17.2166
Logistic Regression	19.3076
Random Forest Classifier	14.3541
Support Vector Classifier	20.6868

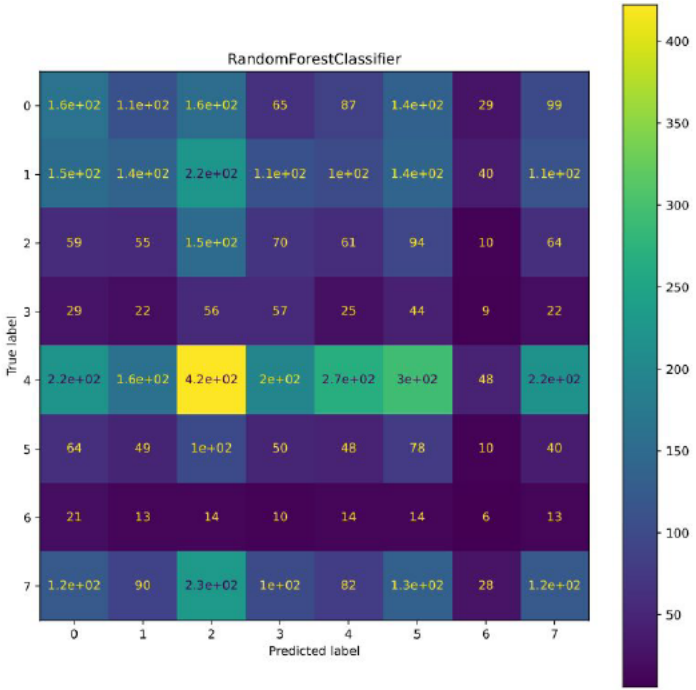
Multinomial Naive Bayes



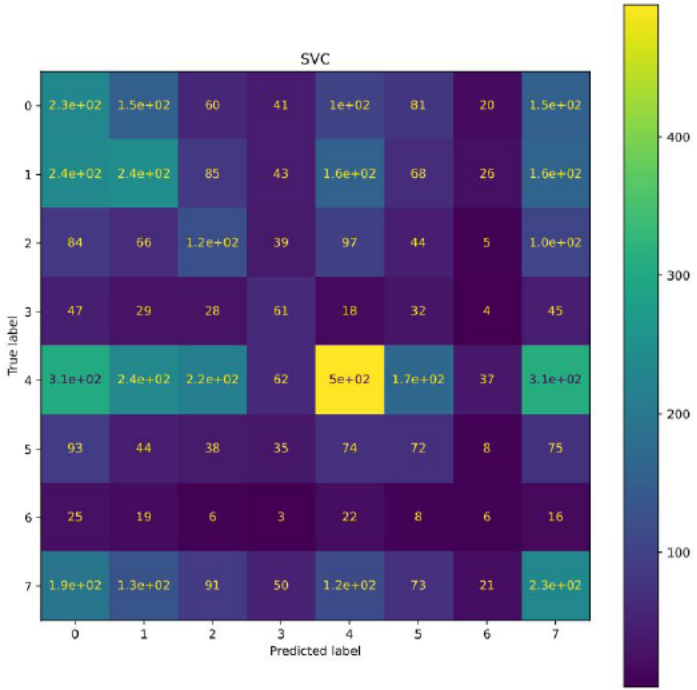
Logistic Regression



Random Forest Classifier



Support Vector Classifier



Conclusions

- Classifiers outperformed random chance.
- Associations between text and label are weak.
- Data is still too noisy.

Machine Learning

C. Deep Neural Networks

Feature Extraction

- Uses Word2Vec:
  - Vocab embeddings (stores in numpy array)
  - Word-indexed documents (stores as pickle)
- These serve as input to the architectures

Deep Neural Networks

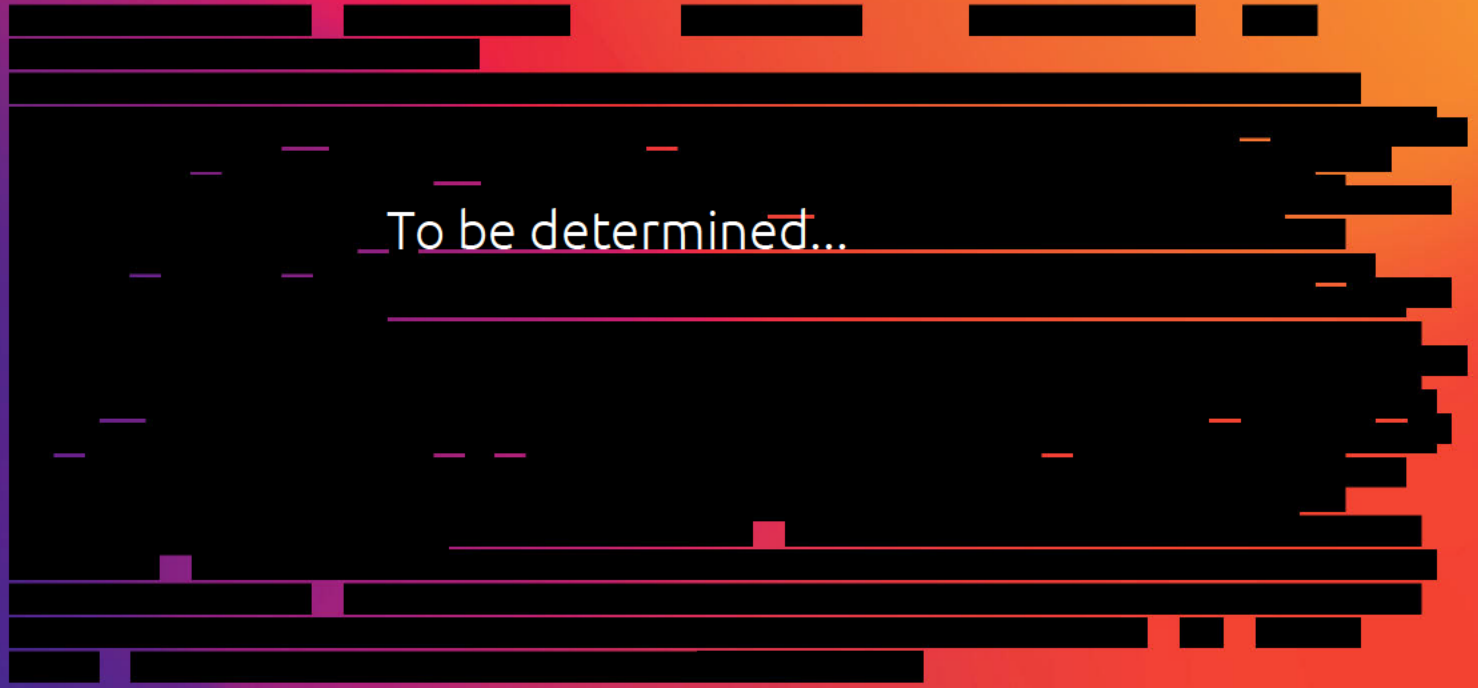
CNN

HAN

HCAN

HiSAN

Results



Some things I learned...

- Be patient
- Stay organized and focused
- I learned a lot
- git
- vim

Deliverables

- Technical Report
- Clustering Notebooks
- Graphics
- Helper Functions

Next steps...

Deep Learning:

- CNN
- HAN
- HCAN
- HiSAN

Get an expert to label some data, and then apply semi-supervised learning methods

Look for additional methods to remove noise from data to improve performance

Extract other data from reports: size, stage/grade, etc.