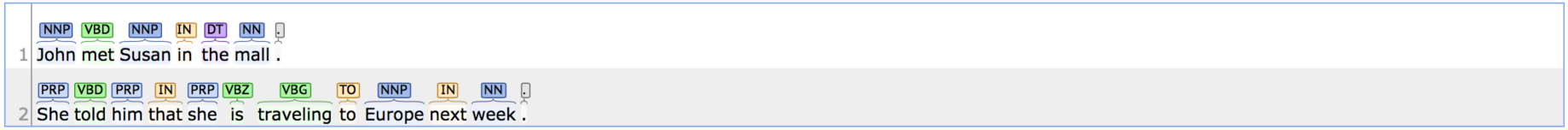
L665 ML for NLPSpring 2018
Carlos Sathler (cssathler@gmail.com)

Assignment 1

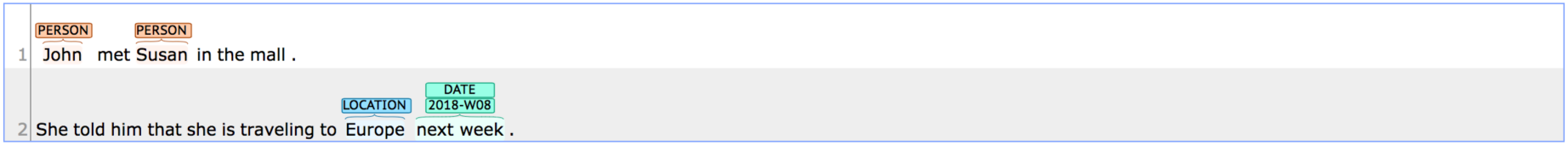
Task 1 – Stanford CoreNLP server and web interface

Analysis of sentence: "John met Susan in the mall. She told him that she is traveling to Europe next week."

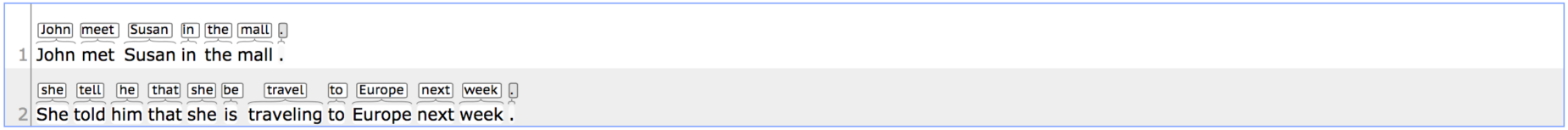
Part-of-Speech:



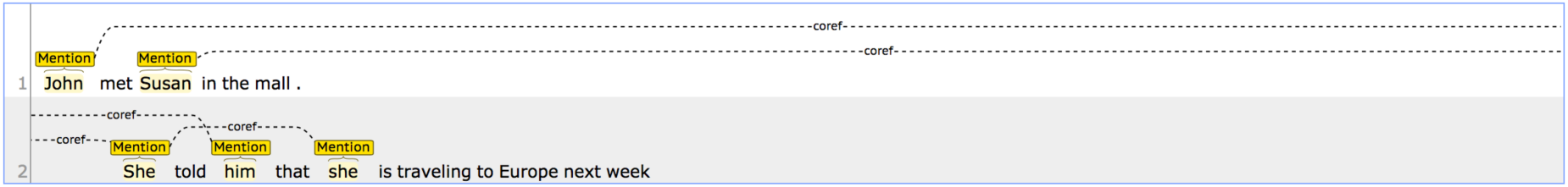
Named Entity Recognition:



Lemmas:



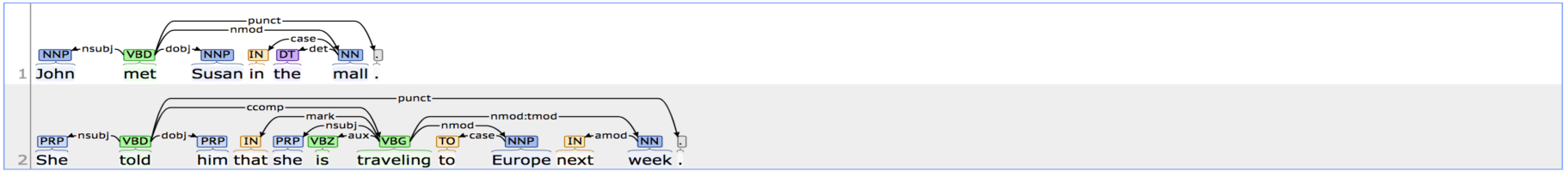
Coreference:



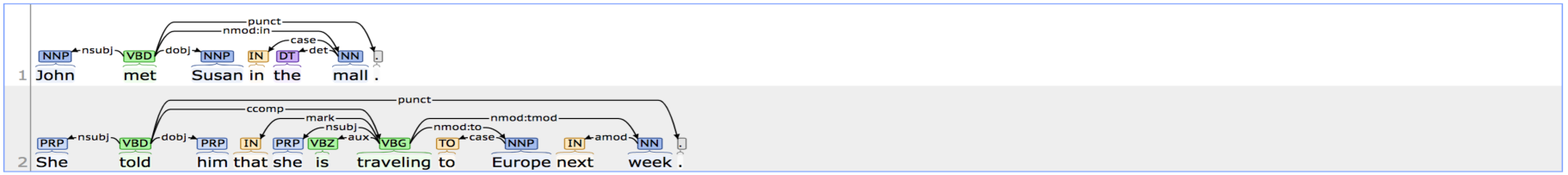
L665 ML for NLPSpring 2018

Carlos Sathler (cssathler@gmail.com)

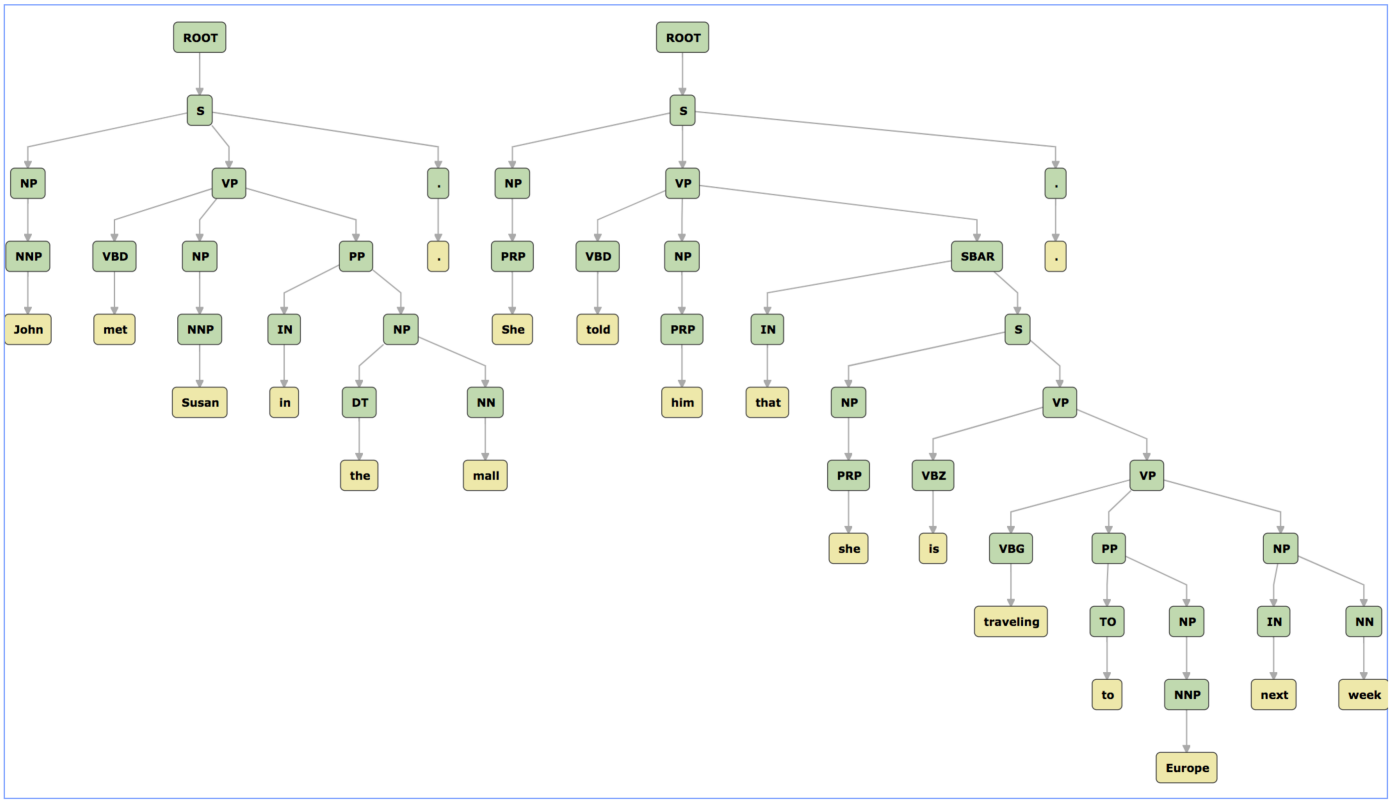
Basic Dependencies:



Enhanced++ Dependencies:



Constituency Parse:



L665 ML for NLPSpring 2018
Carlos Sathler (cssathler@gmail.com)

Examples of tasks that might benefit from NLP features

Here are some examples:

Task	**Level**	**Features**	**Explanation**
Speech recognition	Phonetics	Part-of-speech	If one asks "Alexa, define 'lead'", pronounced to denote the metal "lead", Alexa will reply with the definition of the verb "to lead". But if you ask "Alexa – is 'lead' heavy?" POS tagging of the word "lead" in the sentence "is lead heavy" allows Alexa to answer us correctly, with information about the metal "lead".
Text normalization	Morphological	Lemmatization	Lemmatization reduces the count of word types in text, which would be helpful, for example, when processing a very large vocabulary in text classification.
Semantic role labeling	Semantic	Dependency parse Constituency parse	In the sentence "Jane hurt June" a dependency parser assigns Jane the role of subject and June the object. We can thus infer that Jane is the agent and June the patient. In the sentence "Jane hurt June at the station" the dependency parser qualifies "June" as a temporal modifier to "hurt". The constituency parser however tags "June" as a proper noun, so the constituency parse also lends valuable tags for semantic role labeling.
Q&A systems	Pragmatic	Coreference	(Per class example) If we ask Alexa "Who is the president of the United States?" we get "Donald Trump". If we further ask, "How old is he?" Alexa will not know the answer, because coreference is not implemented across sessions in Alexa. If we ask Google via an Android phone for example the same two questions in succession, Google will answer both questions accurately, and even at third one, such as "How about his sons?"
Timeline extraction	Semantic Pragmatic	Named-entities Dependency parse	(Per class example) In the sentence "Earlier you gave me Aspirin and I felt good. Yesterday the hospital gave me Advil and I felt bad" it is possible to establish a timeline for administered treatment as well as treatment results with the help of named-entity and dependency parse tags.

L665 ML for NLPSpring 2018
Carlos Sathler (cssathler@gmail.com)

Strategies for vectorization of POS tag, lemmas, and named-entities features

We will use POS tag, lemmas and named-entities and lemmas as examples. These features could benefit speech recognition, text normalization, and timeline extraction tasks. There is a one-to-one relationship between each word in the text and each NLP feature value for each of these NLP feature types. For example, in the first sentence, the word "met" is associated with only one POS tag feature: "VBD". The same will be true for all words and their corresponding lemmas and named-entity features. (In the later case a word can be associated with zero named-entities). For such cases in which the relationship between a word and a NLP feature is a one-to-one relationship, one could follow the following vectorization strategy:

1 – identify all distinct tag types found in the text (e.g. all the different POS tags)
2 – Create a one-hot-encoded vector to encode the value of the tag

One of the advantages of this approach is its simplicity. However, it relies on the assumption that the number of different tag values for the vocabulary will be manageable, i.e., the length of the resulting one-hot-encoded feature vector for each word will not exceed memory capacity, considering vocabulary size and number of documents to process. That will likely be the case for POS tags and most likely for named-entities tags. The same may not be true for lemmas, which for large vocabularies could also comprise an excessively large set. If that were the case, I would use a technique such as word2vec to vectorize the lemmas.

An alternative approach would be to simply generate integer numbers for each different value of POS tag, lemma, etc. and encode each value as a separate dimension in a 1D vector. We will show that approach in Python notebooks for tasks 2 and 3 of this assignment. For task 4 we will use the vectorized features to compare the performance of a BOW approach vs. a BOW + NLP features approach on a text classification problem.

Exploring POS tag vectorization in a classification problem

Python notebook here: https://github.com/csathler/IU_MSDS/blob/master/NLP/assignment1/assignment1_task1.ipynb

We will explore in more detail if using NLP features, specifically POS tags, will help on a classification task called "Toxic Comment Classification Challenge" proposed in the website Kaggle.com (https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge#description).

The dataset is very simply, consisting of a key, a comment field, and 6 separate flags (taking values 0 or 1) that qualify a comment as: toxic, severe_toxic, obscene, threat, insult, and identity_hate.

L665 ML for NLPSpring 2018
Carlos Sathler (cssathler@gmail.com)

Below are three examples of comments from the data set that are “toxic” and “threat” (but not “severe_toxic”, “obscene”, “insult”, or “identity_hate”). I do not include other examples because the language of the comments is extremely offensive.

Hi! I am back again! Last warning! Stop undoing my edits or die!
I'm also a sock puppet of this account...SUPRISE!! -sincerely, The man that will track you down from the Internet and kill you
Whoever put a notices on my page. I will kill u

For this exploration we will pre-preprocess comments to create 3 datasets, as follows:

- BOW
- BOW + POS tag (vectorized)
- POS tag (as BOW)
- BOW + POS tag (as BOW)

POS tag vectorization is done by assigning an integer number representing each POS tag value to a unique integer.

In the assignment1_task1.ipynb notebook posted to GitHub we compare the performance of Logistic Regression on a BOW dataset against an AdaBoost decision tree classifier on a dataset that concatenates BOW with vectorized POS tags.

Dataset	Classifier	Accuracy
BOW	Logistic Regression	0.9548379010695187
BOW	AdaBoostClassifier	0.9490307486631016
BOW + POS tags (vectorized)	AdaBoostClassifier	0.9504094251336899

Conclusion: POS tag vectors do not yield a significant gain in accuracy performance for this classification task.

L665 ML for NLPSpring 2018
Carlos Sathler (cssathler@gmail.com)

Task 2 – spaCy

Python notebook here: https://github.com/csathler/IU_MSDS/blob/master/NLP/assignment1/assignment1_task2_spaCy.ipynb

Comparison with CoreNLP features

The output from spaCy is very similar to the output from CoreNLP. Some differences are noted below:

- POS tagging
  - Token abbreviations are slightly different for some POS
  - spaCy isolates and tags spaces
- Named entity
  - CoreNLP gives more details on dates, such as year and number of the week
- Lemmas
  - CoreNLP removes spaces from the result
  - In spaCy pronouns are replaced with tag, whereas in CoreNLP pronouns are repeated
- Dependency parse
  - For our example sentences results were the same, with POS tagging showing slight different tags in some cases
  - spaCy only has one dependency parse results, whereas CoreNLP has an enhanced parse

Vectorization strategy

Notebook assignment1_task2_spaCy.ipynb posted to GitHub shows our exploration of spaCy and demonstrates one way to vectorize POS tags, by converting input sequences of works into sequences of POS tab embeddings. This is the strategy we used for the classification task documented in the homework task 1.

L665 ML for NLPSpring 2018
Carlos Sathler (cssathler@gmail.com)

Task 3 – CoreNLP

Python notebook here: https://github.com/csathler/IU_MSDS/blob/master/NLP/assignment1/assignment1_task3_CoreNLP.ipynb

Notebook "assignment1_task3_CoreNLP.ipynb" posted to github shows our exploration of CoreNLP calls and data structures received from the server. We used the package "py-corenlp" per recommendation/suggestion found in the Stanford CoreNLP website. The package works very well. Calls to CoreNLP seamlessly populates a python dictionary variable, as seen in the notebook.

In the end of the notebook we demonstrate one way to vectorize a constituent parse tree using a bracketed representation of the tree. Due to performance and memory requirements we did not experiment adding constituent parse tree vectorized features to the dataset used in the toxic challenge classification task.

Vectorization strategy

We identified the dictionary fields containing the constituent parse tree, removed line feeds, and manipulated the tree as a string in bracketed notation. Each different token in the string was mapped to an integer to generate an embedded representation of the tree.

Task 4 – Bonus task

spaCy Vectors

Python notebook here: https://github.com/csathler/IU_MSDS/blob/master/NLP/assignment1/assignment1_task4.1_spaCy_vectors.ipynb

Notebook assignment1_task4.1_spaCy_vectors.ipynb shows how to use a pretrained vectorized vocabulary of 1M words delivered by spaCy to generate word embeddings for general NLP tasks. Each vectorized word in the vocabulary has dimension 300. In the notebook we truncate comments in the "toxic" dataset to a maximum of 200 words (shorter comments are left padded to 200) and generate a matrix of dimension 200x300 for each comment (row) in the dataset. The data is then fed to a neural network implemented with Keras (TensorFlow backend), with two layers: one LSTM layer and one MLP layer. With practically no tuning, the model over fit on the training dataset (please see plot in notebook), but produced an accuracy score of 0.9555 in the test partition. This result was obtained after training on less than 20% of the data used in task 1 of the homework. On task 1 we obtained an accuracy of 0.9548 training on 70% of the dataset with BOW + Logistic Regression approach. Clearly word embedding and recurrent neural nets generated better results, however, slow processing time is a limiting factor for this approach.

L665 ML for NLPSpring 2018
Carlos Sathler (cssathler@gmail.com)

FastText

Python notebook here: https://github.com/csathler/IU_MSDS/blob/master/NLP/assignment1/assignment1_task4.2_fastText.ipynb

Notebook assignment1_task4.2_fastText.ipynb shows a fastText implementation using Keras. My code is modeled after code written by the Keras team here https://github.com/keras-team/keras/blob/master/examples/imdb_fasttext.py and by tips on embedding in Keras from Jason Brownlee, Ph.D., from https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/. The implementation adheres to the fastTask text classifier proposed in the pager "Bag of Tricks for Efficient Text Classification" by Joulin, Armand et al [1]. The approach can be summarized as follows:

1. Create sequence of n-grams for each document that needs to be classified (for our problem, each comment in the "toxic" dataset)
2. Create NN input node to learn vectorized representation of the n-grams as part of the classification task
3. Create NN hidden node to average n-grams vectorized representation across the dimension of their vector representation
4. Create NN output layer with softmax activation to generate class prediction*

As stated in the paper, fastText is very fast. Our notebook shows the network over fit after few epochs and accuracy result on the test data was inferior to the BOW + Logistic Regression benchmark (task 1 of homework). However we did not try regularization and only used 1-grams as input, whereas the BOW had up to 3-grams. So clearly there is room for easily improving fastText on the "toxic" comment classification problem.

* Joulin, Armand et al. proposed softmax on multi-class classification in their paper. We use sigmoid because we're dealing with binary problem.

[1] Joulin, Armand, et al. "Bag of tricks for efficient text classification." *arXiv preprint arXiv:1607.01759* (2016).

Gensim doc2vec
Python notebook here: https://github.com/csathler/IU_MSDS/blob/master/NLP/assignment1/assignment1_task4.3_gensim_doc2vec.ipynb

Notebook assignment1_task4.3_gensim_doc2vec.ipynb shows how train vectorized representations of sentences for use in the "toxic" comment classification problem. We compared our benchmark BOW with Logistic Regression result with BOW + vectorized sentences and vectorized sentences alone with AdaBoostClassifier and obtained poor results by comparison.

Benchmark accuracy result (BOW + Logistic Regression): 0.9548
AdaBoostClassifier with BOW: 0.9490
AdaBoostClassifier with BOW + vectorized sentence: 0.9433
AdaBoostClassifier with vectorized sentence: 0.8834

Conclusion: A simple approach to use vectorized sentences (adding sentence vectors to BOW) with AdaBoostClassifier did not improve our results.