7 Appendix

A. Forward

The functions utilized to prepare the data shown in the report are included below. Note that this is by no means a polished finished product. These are utility function and meant to be used and experimented with. Perhaps with additional time, these could have been prepared into a single application with a clean user interface, but that remains for the future work.

Wherever possible, detailed function/script descriptions have been provided. Inline comments are provided for almost all of the code which explain the fine deatils while the description above provides a more high-level overview. With the exception of the spider_plot() function (sometimes referred to as a radar plot), all functions not outlined here are included with MATLAB. spider_plot() was obtained from the Mathworks File Exchange. All credit to belongs to its original author.

B. ldacluster.m

This script reads in a pre-saved data from a .mat file. For each paper, it will extract out the abstract, filtering out papers with missing or invalid abstracts. In doing so, it feeds each through the preprocess function. (See E..) It then makes a bag of words model for each and removes empty words. Finally, it selects a subset of the data and repeatedly runs LDA model fittings for the collection, aggregating the perplexity for each. At the end, it displays a topic distribution and a wordclouds.

```
clear;
close all;

% Read in abstracts
load('data/Articles/IEEEWkshpMachLearn.mat');
abstracts(1:size(articles)) = tokenizedDocument;
cnt = 1;
for i=1:numel(articles)
    % Filter out articles that don't have an abstract
    if any(string(fieldnames(articles{i})) == 'abstract')
        % Filter out any articles that have an html-based abstract
        if ~contains(articles{i}.abstract,'</div>')
            % Filter out any articles that have an empty abstract
            if numel(strsplit(articles{i}.abstract)) > 2
                abstracts(cnt) = preprocess(articles{i}.abstract);
                cnt = cnt + 1;
            end
        end
    end
end
abstracts = abstracts(:,1:cnt);
clear articles

% Make a bag of words
bag = bagOfWords(abstracts);
bag = removeInfrequentWords(bag,2);
[bag,idx] = removeEmptyDocuments(bag);

% % Select a subset for training to determine perplexity
% cvp = cvpartition(numel(abstracts),'HoldOut',0.1);
% abstractsTrain = abstracts(cvp.training);
% abstractsTest = abstracts(cvp.test);
% bag = bagOfWords(abstractsTrain);
% bag = removeInfrequentWords(bag,2);
% bag = removeEmptyDocuments(bag);

% % Find a target number of topics
% numTopicsRange = [10 20 40 80 160];
% parfor i = 1:numel(numTopicsRange)
%     mdl(i) = fitlda(bag,numTopicsRange(i),'Verbose',1);
%     [~,ppl(i)] = logp(mdl(i),abstractsTest');
% end

% Run an LSA model
numcomponents = 15;
mdl = fitlda(bag,numcomponents);

% Show wordclouds for each category
dim = ceil(sqrt(numcomponents));
figure
```

```
for i = 1:numcomponents
    subplot(dim,dim,i)
    wordcloud(mdl,i);
    title("Topic " + i)
end

% Show topic mixtures
figure
topicMixtures = transform(mdl,abstracts(1:numcomponents)');
barh(topicMixtures(1:numcomponents,:),'stacked')
xlim([0 1])
title("Topic Mixtures")
xlabel("Topic Probability")
ylabel("Document")
```

C. lsacluster.m

This script reads in a pre-saved data from a .mat file. For each paper, it will extract out the abstract, filtering out papers with missing or invalid abstracts. In doing so, it feeds each through the preprocess function. (See E..) It then makes a bag of words model for each and removes empty words. Finally, it selects a subset of the data and runs an LSA model for the specified number of components. At the end, it displays a radar plot for the topic vectors and a wordclouds.

```
clear;
close all;

% Read in abstracts
load('data/IEEECommJournArticles.mat');
abstracts(1:size(articles)) = tokenizedDocument;
for i=1:numel(articles)
    % Filter out articles that don't have an abstract
    if any(string(fieldnames(articles{i})) == 'abstract')
        % Filter out any articles that have an html-based abstract
        if ~contains(articles{i}.abstract,'</div>')
            % Filter out any articles that have an empty abstract
            if numel(strsplit(articles{i}.abstract)) > 2
                abstracts(i) = preprocess(articles{i}.abstract);
            end
        end
    end
end
clear articles;

% Make a bag of words
abstracts = removeEmptyDocuments(abstracts);
bag = bagOfWords(abstracts);
bag = removeInfrequentWords(bag,2);
[bag,idx] = removeEmptyDocuments(bag);

% Run an LSA model
numcomponents = 15;
mdl = fitlsa(bag,numcomponents);

% Show wordclouds for each category
figure;
dim = ceil(sqrt(numcomponents));
for i=1:numcomponents
    subplot(dim,dim,i);
    [temp,idx] = sort(mdl.DocumentScores(:,i),'descend');
    bagtemp = bagOfWords(abstracts(idx(1:12)));
    wordcloud(bagtemp);
    title("Topic " + i)
end

% Show a radar plot for
figure;
data = mdl.DocumentScores(1:10,:);
labels = strcat('Topic',{' '},string(1:numcomponents));
spider_plot(data,labels,10,1,'LineWidth',2);
```

D. plotNumTopics.m

This script plots the perplexity of all data sets. For each .mat saved data file in the specified directory, it reads in and displays the saved perplexity.

```
clear;
close all;

% Input
range = 21:30;
name = 'PPLIEEETrans';

% Get files
path = './data/MDLs/';
dir_info = dir(path);
files = {dir_info.name};
files = files(3:end);
files = string(files);
files = files(~contains(files,'.png'));
files = files';

fig = figure;
% set(fig,'Visible','off');
xlim([0,175]);
ylim([400,1250]);
xlabel('Number of Topics')
ylabel('Perplexity')
title('Perplexity per Data Set')
hold all;

for i=range
    load(strcat(path,files(i)));
    plot(numTopics,ppl)
    clear mld ppl numTopics
end

legend(files(range),'location','eastoutside')

fig.PaperPositionMode = 'auto';
print(strcat('data/Final/',name),'-dpng','-r0')
```

E. preprocess.m

This function cleans and sanitizes input data. Before any processing can take place, all raw data must be fed through this function. It performs several steps: tokenizing, removing stopwords, removing words that are too big or too small, and finally runs each word through MATLAB's build-in Porter stemmer function.

```
function [documents] = preprocess(textData)

% Erase URLs, punctuation and numbers
cleanTextData = eraseURLs(textData);
cleanTextData = regexprep(cleanTextData,'\d','');
cleanTextData = erasePunctuation(cleanTextData);

% Convert the text data to lowercase.
cleanTextData = lower(cleanTextData);

% Tokenize the text.
documents = tokenizedDocument(cleanTextData);

% Remove a list of stop words.
documents = removeWords(documents,stopWords);

% Remove words with 2 or fewer characters, and words with 15 or greater
% characters.
documents = removeShortWords(documents,2);
documents = removeLongWords(documents,15);

% Normalize the words using the Porter stemmer.
documents = normalizeWords(documents);

end
```

F. processAbstracts.m

This is an abbreviated version of B. and C.. It simply extracts abstracts and does not attempt any model fittings. It reads in all files in the specified directory. For each paper in each file, it will extract out the abstract, filtering out papers with missing or invalid abstracts. In doing so, it feeds each through the preprocess function. (See E..) It then makes a bag of words model for each and removes empty words.

```
clear;
close all;

% Get files
path = './data/Articles/';
dir_info = dir(path);
files = {dir_info.name};
files = files(3:end);
files = string(files);

% Loop through each file of articles
for i=16:numel(files)
    load(strcat(path,files(i)));

    % Extract abstract from each and turn into a tokenized document
    abstracts(1:size(articles)) = tokenizedDocument;
    cnt = 1;
    for j=1:numel(articles)
        % Filter out articles that don't have an abstract
        % Filter out any articles that have an html-based abstract ...
        % Filter out any articles that have an empty abstract
        if any(string(fieldnames(articles{j})) == 'abstract') && ...
            ~contains(articles{j}.abstract,'</div>') && ...
            numel(strsplit(articles{j}.abstract)) > 2
             try
                 abstracts(cnt) = preprocess(articles{j}.abstract);
                 cnt = cnt + 1;
             % If it still fails, move on and throw the error as a warning
             catch ME
                 %disp(getReport(ME,'extended','hyperlinks','on'));
             end
        end
    end
    abstracts = abstracts(:,1:cnt);

    % Make a bag of words
    bag = bagOfWords(abstracts);
    bag = removeInfrequentWords(bag,2);
    [bag,~] = removeEmptyDocuments(bag);

    % Save data
    save(strcat('data/Abstracts/',files(i)),'abstracts','bag');
    clear articles abstracts
end
```

G. processFinalMDL.m

Once a number of topics has been chosen based on the perplexity distribution in B., this function is called to generate a single LDA and LSA model based on the number of topics chosen. Function runs for all saved data files in the specified directory. See `https://developer.ieee.org/docs/read/Metadata_API_details` for all available parameters.

```
clear;
close all;

% Get files
path1 = './data/MDLs_Small/';
path2 = './data/Abstracts/';
dir_info = dir(path2);
files = {dir_info.name};
files = files(3:end);
files = string(files);
files = files';

for i=2:numel(files)
    load(strcat(path2,files(i)));
    numTopics = 20;
    lda = fitlda(bag,numTopics,'Verbose',0);
```

```
    lsa = fitlsa(bag,numTopics);
    save(strcat(path1,files(i)),'lda','lsa','numTopics');
end
```

H. queryApi.m

This functions creates a web request in the format the the IEEE XPlore API expects. HTTP get parameters can be passed as arguments to this MATLAB function and they will be appended to the query. See

```
function [json] = queryApi(apiKey, varargin)

% Build URL
url = 'http://ieeexploreapi.ieee.org/api/v1/search/articles?parameter';
url = [url,'&apikey=',apiKey];
for i=1:numel(varargin)
    url = [url,'&',varargin{i}];
end

% Make HTTP request
options = matlab.net.http.HTTPOptions('ConnectTimeout',30);
request = matlab.net.http.RequestMessage;
response = request.send(url,options);
json = response.Body.Data;

end
```

I. graphPPLAll.m

This script creates the scatter plot of paper perplexity vs. number of papers. It aggregates the data from earlier processing functions and creates a scatter plot and text labels.

```
clear
close all

path = './data/Abstracts/';
dir_info = dir(path);
files = {dir_info.name};
files = files(3:end);
files = string(files);
files = files';

load('data/MDLS_Small/ppl.mat')

scatter(cnt,ppl)
xlabel('Number_of_Documents')
ylabel('Perplexity')
set(gca, 'Ydir', 'reverse')

dx = 0.012;
dy = -0.012;

dx = dx*max(cnt);
dy = dy*max(ppl);

for i=1:numel(cnt)
   text(cnt(i)+dx,ppl(i)+dy,files(i),'fontsize',6)
end
```

J. tfidf.m

This script creates a TF-IDF ranking of all collections in a specified directory (specified on line 10) according to the term input on line 6. A bar chart is created and each data set in the directory is added to the bar chart.

```
clear;
close all;

% Input
range = 1;
term = 'cluster';
```

```
k = 20;

% Get files
path = './data/Abstracts/';
dir_info = dir(path);
files = {dir_info.name};
files = files(3:end);
files = string(files);
files = files';

for i=range
    load(strcat(path,files(i)));
    [~,idx] = find(bag.Vocabulary == term);
    tfidfMatrix = tfidf(bag,'TFWeight','log','IDFWeight','smooth');
    [sorted,sortedidx] = sort(tfidfMatrix(:,idx),'descend');
    figure
    bar(sorted(1:k))
    xtickangle(45)
    set(gca,'Xtick',1:k)
    set(gca,'XTickLabel',string(sortedidx(1:k)))
    xlabel('Document')
    ylabel('TFIDF')
    title(strcat('TFIDF of','{ }','"',term,'"'))
end
```

K. tfidfCluster.m

This performs much the same function as J., but does it for two terms (lines 6 and 7). Instead of a bar graph, a scatter plot is created, plotting the two term's tf-idf ranking comparatively.

```
clear;
close all;

% Input
range = 1;
term1 = 'cluster';
term2 = 'languag';
k = 20;

% Get files
path = './data/Abstracts/';
dir_info = dir(path);
files = {dir_info.name};
files = files(3:end);
files = string(files);gi
files = files';

for i=range
    load(strcat(path,files(i)))
    [~,termidx1] = find(bag.Vocabulary == term1);
    [~,termidx2] = find(bag.Vocabulary == term2);
    tfidfMatrix = tfidf(bag,'TFWeight','log','IDFWeight','smooth');
    termlist1 = tfidfMatrix(:,termidx1);
    termlist2 = tfidfMatrix(:,termidx2);
    figure
    scatter(termlist1,termlist2)
    xlabel(strcat('TFIDF of"',term1,'"'))
    ylabel(strcat('TFIDF of"',term2,'"'))
    cnt = 1;
end
```

L. getAbstractFromArnum.m

This utility function provides a quick way to retrieve only the abstract from a given IEEE document reference number. It creates and formats a web response formatted to pull data from plain HTML, rather than the IEEE API (as the API is limited to 200 queries per day). A valid session cookie from the university's library proxy will need to be obtained so that the web request .

```
function[abstractText] = getAbstractFromArnum(arnumber, libproxycookie)

% Build http request
request = matlab.net.http.RequestMessage;
request = request.addFields(matlab.net.http.field.CookieField(libproxycookie));
```

```
% Send request and read response
url = ['http://ieeexplore.ieee.org.libproxy.mst.edu/', ...
       'xpl/articleDetails.jsp?arnumber=', num2str(arnumber)];
response = request.send(url);
html = char(response.Body.Data);

% Check for invalid libproxy session
if(contains(html, 'Please enter your Missouri S&T Userid and Password.'))
    error('Invalid libproxy login. Please check the cookie paramters.');
end

% Extract out abstract text
startIdx = strfind(html,'abstract');
endIdx = strfind(html, 'isJournal');
abstractText = html(startIdx+11:endIdx-4);

end
```

M. getArticles.m

This is a specialized version of H. that pulls the maximum allowed number of records for a specific publication. This function was heavily used for example 2 in the paper. For as long as the API continues to return requests, the script will increment the records counter and request new data. If an error code is returned instead of valid data, the script will abort. Upon completion, the data is saved to a .mat file that can be read by the various other scripts and functions listed here.

```
clear;

% Load API key
load('data/private.mat');

% Initializations
records = 0;
articles = [];
total_records = records + 1;
dry_run = false;

% Query API and collect results
while records < total_records
    json = queryApi(apikey,...
                    'max_records=200',...
                    ['start_record=',num2str(records)],...
                    ['publication_title=',urlencode('Language Processing')]);
    if ~any(contains(fieldnames(json),'error_code'))
        total_records = json.total_records;
        articles = [articles;json.articles];
    end
    records = records + 200;
    if dry_run
        queries = ceil(total_records / 200);
        break;
    end
end

if ~dry_run
    save('data/Raw_API_Data/IEEELangProc.mat','articles');
end
```

N. graphPerplexity.m

This function graphs the perplexity of all articles in a specified directory (input on line 5). Assuming they have already been processed by a fitting model, the perplexities are aggregated and placed on a chart.

```
clear
close all

% Get files
path = './data/Abstracts/';
dir_info = dir(path);
files = {dir_info.name};
files = files(3:end);
files = string(files);
```

```
files = files';
cnt = [];

for i=1:numel(files)
    load(strcat(path,files(i)));
    cnt(i) = numel(abstracts);
end

% Draw chart
figure
bar(ppl);
set(gca,'XTickLabel',string(files));
set(gca,'XTick',1:numel(files));
xtickangle(45);
```