Co-occurrence Network & More on LDA

- get more info from textual data

Outline

- Create Co-occurrence Network
  - graph
- Choose Number of Topics for LDA Model
  - logp
- Compare LDA Solvers
  - fitlda

Create Co-occurrence Network

- This example shows how to create a co-occurrence network using a bag-of-words model.
- Given a corpus of documents, a co-occurrence network is an undirected graph, with nodes corresponding to unique words in a vocabulary and edges corresponding to the frequency of words co-occurring in a document. Use co-occurrence networks to visualize and extract information of the relationships between words in a corpus of documents. For example, you can use a co-occurrence network to discover which words commonly appear with a specified word.

Import Text Data

- Extract the text data in the file weekendUpdates.xlsx using readtable. The file weekendUpdates.xlsx contains status updates containing the hashtags "#weekend" and "#vacation". Read the data using the readtable function and extract the text data from the TextData column.

Import Text Data

```
filename = "weekendUpdates.xlsx";
tbl = readtable(filename,'TextType','string');
textData = tbl.TextData;
textData(1:5)
```

```
ans = 5x1 string
 "Happy anniversary! ❤ Next stop: Paris! ✈ #vacation"
 "Haha, BBQ on the beach, engage smug mode!  ❤  #vacation"
 "getting ready for Saturday night  #yum #weekend "
 "Say it with me - I NEED A #VACATION!!! ☹  "
 " Chilling  at home for the first time in ages…This is the life!  #weekend"
```

Preprocess Text Data

- Tokenize the text, convert it to lowercase, and remove the stop words.

```
documents = tokenizedDocument(textData);
documents = lower(documents);
documents = removeStopWords(documents);
```

- Create a matrix of word counts using a bag-of-words model.

```
bag = bagOfWords(documents);
counts = bag.Counts;
```

Preprocess Text Data

- To compute the word co-occurrences, multiply the word-count matrix by its transpose.

```
cooccurrence = counts.'*counts;
```

- Convert the co-occurrence matrix to a network using the graph function.

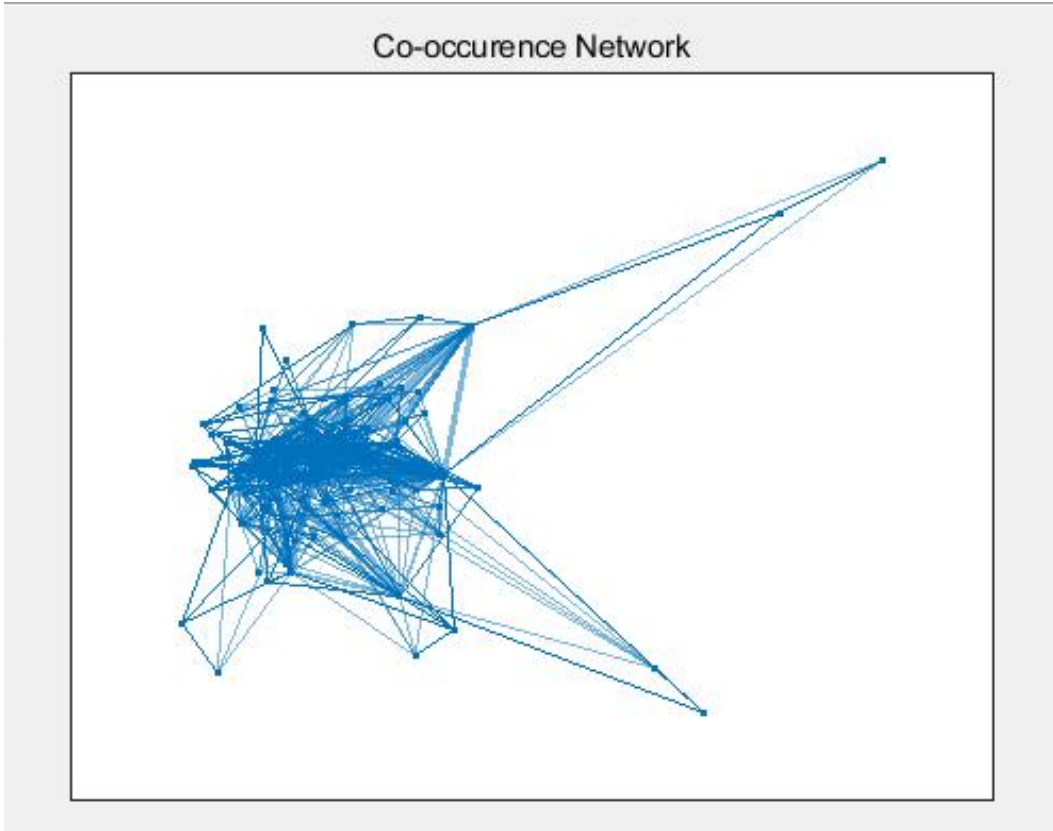
```
G = graph(cooccurrence,bag.Vocabulary,'omitselfloops');
```

- Visualize the network using the plot function. Set the line thickness to a multiple of the edge weight.

```
LWidths = 5*G.Edges.Weight/max(G.Edges.Weight);
```

Preprocess Text Data

```
plot(G,'LineWidth',LWidths)
title("Co-occurence Network")
```



Preprocess Text Data

- Find neighbors of the word "great" using the neighbors function.

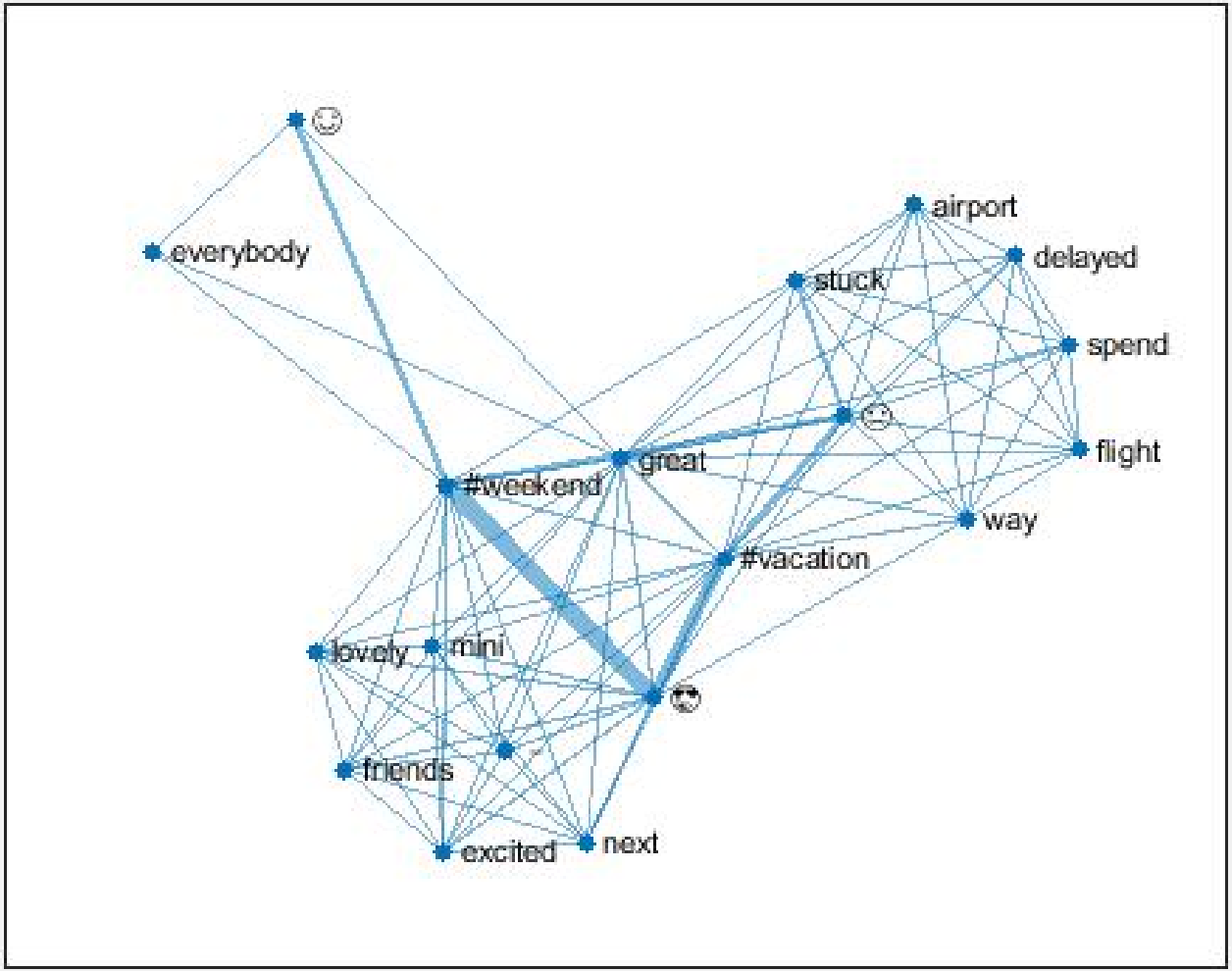
```
word = "great"
idx = find(bag.Vocabulary == word);
nbrs = neighbors(G,idx);
bag.Vocabulary(nbrs)'
```

```
ans =
  1×18 string 数组
"next"  "#vacation"  "😎"  "#weekend"  "☹ "  "excited"  "flight"
"delayed"  "stuck"  "airport"  "way"  "spend"  "☺"  "lovely"  "friends"
"–"  "mini"  "everybody"
```

Preprocess Text Data

- Visualize the co-occurrences of the word "great" by extracting a subgraph of this word and its neighbors.

```
H = subgraph(G,[idx; nbrs]);
LWidths = 5*H.Edges.Weight/max(H.Edges.Weight);
plot(H,'LineWidth',LWidths)
title("Co-occurence Network - Word: """ + word + """");
```


Co-occurence Network - Word: "great"
everybody
airport
delayed
stuck
spend
flight
great
#weekend
way
#vacation
lovely
mini
friends
excited
next

Choose Number of Topics for LDA Model

- This example shows how to decide on a suitable number of topics for a latent Dirichlet allocation (LDA) model.
- To decide on a suitable number of topics, you can compare the goodness-of-fit of LDA models fit with varying numbers of topics. You can evaluate the goodness-of-fit of an LDA model by calculating the perplexity of a held-out set of documents. The perplexity indicates how well the model describes a set of documents. A lower perplexity suggests a better fit.

Extract and Preprocess Text Data

- Load the example data. The file factoryReports.csv contains factory reports, including a text description and categorical labels for each event. Extract the text data from the field Description.

```
filename = "factoryReports.csv";
data = readtable(filename,'TextType','string');
textData = data.Description;
```

Extract and Preprocess Text Data

- Tokenize and preprocess the text data using the function preprocessText which is listed at the end of this example.

```
documents = preprocessText(textData);

documents(1:5)
```

```
ans =
  5×1 tokenizedDocument:
   6 tokens: item occasionally get stuck scanner spool
   7 tokens: loud rattle bang sound come assembler piston
   4 tokens: cut power start plant
   3 tokens: fry capacitor assembler
   3 tokens: mixer trip fuse
```

Extract and Preprocess Text Data

- Set aside 10% of the documents at random for validation.

```
numDocuments = numel(documents);
cvp = cvpartition(numDocuments,'HoldOut',0.1);
documentsTrain = documents(cvp.training);
documentsValidation = documents(cvp.test);
```

- Create a bag-of-words model from the training documents. Remove the words that do not appear more than two times in total. Remove any documents containing no words.

```
bag = bagOfWords(documentsTrain);
bag = removeInfrequentWords(bag,2);
bag = removeEmptyDocuments(bag);
```

Choose Number of Topics

- The goal is to choose a number of topics that minimize the perplexity compared to other numbers of topics. This is not the only consideration: models fit with larger numbers of topics may take longer to converge. To see the effects of the tradeoff, calculate both goodness-of-fit and the fitting time. If the optimal number of topics is high, then you might want to choose a lower value to speed up the fitting process.

Choose Number of Topics

- Fit some LDA models for a range of values for the number of topics. Compare the fitting time and the perplexity of each model on the held-out set of test documents. The perplexity is the second output to the **logp** function. The fitting time is the TimeSinceStart value for the last iteration. This value is in the History struct of the FitInfo property of the LDA model.

Choose Number of Topics

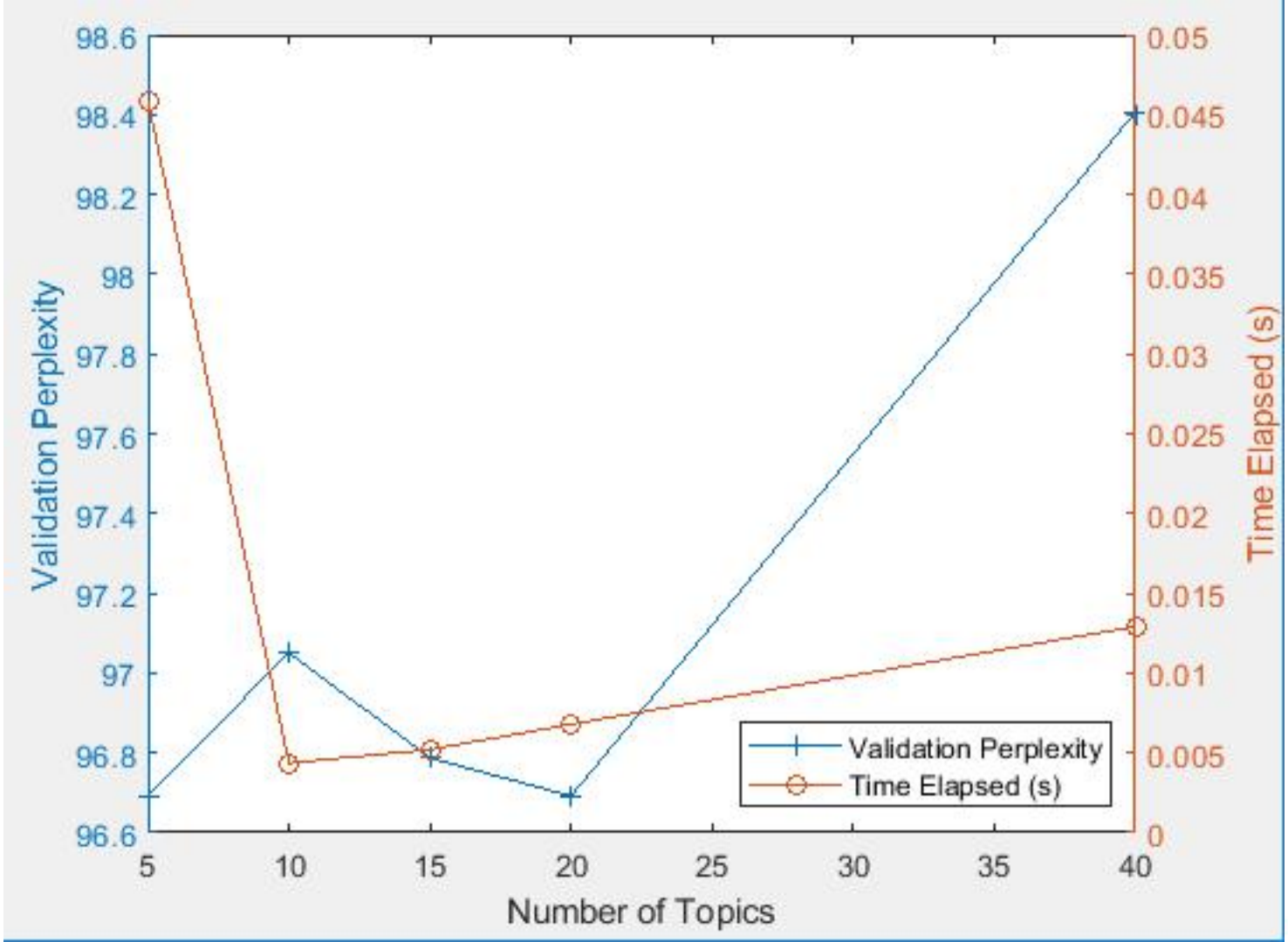
- For a quicker fit, specify 'Solver' to be 'savb'. To suppress verbose output, set 'Verbose' to 0.

```
numTopicsRange = [5 10 15 20 40];
for i = 1:numel(numTopicsRange)
  numTopics = numTopicsRange(i);
  mdl = fitlda(bag,numTopics, 'Solver','savb', 'Verbose',0);
  [~,validationPerplexity(i)] = logp(mdl,documentsValidation);
  timeElapsed(i) = mdl.FitInfo.History.TimeSinceStart(end);
end
```

Choose Number of Topics

- Show the perplexity and elapsed time for each number of topics in a plot. Plot the perplexity left and the time elapsed right.

```
figure
yyaxis left
plot(numTopicsRange,validationPerplexity,'+-')
ylabel("Validation Perplexity")
yyaxis right
plot(numTopicsRange,timeElapsed,'o-')
ylabel("Time Elapsed (s)")
legend(["Validation Perplexity" "Time Elapsed (s)"],'Location','southeast')
xlabel("Number of Topics")
```


Validation Perplexity
Time Elapsed (s)
Number of Topics
98.6
98.4
98.2
98
97.8
97.6
97.4
97.2
97
96.8
96.6
0.05
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
5
10
15
20
25
30
35
40
Validation Perplexity
Time Elapsed (s)

Choose Number of Topics

- The plot suggests that fitting a model with 10–20 topics may be a good choice. The perplexity is low compared with the models with different numbers of topics. With this solver, the elapsed time for this many topics is also reasonable. With different solvers, you may find that increasing the number of topics can lead to a better fit, but fitting the model takes longer to converge.

Compare LDA Solvers

- This example shows how to compare latent Dirichlet allocation (LDA) solvers by comparing the goodness of fit and the time taken to fit the model.

Import Text Data

- Import a set of abstracts and category labels from math papers using the arXiV API. Specify the number of records to import using the importSize variable. Note that the arXiV API is rate limited to querying 1000 articles at a time and requires waiting between requests.

```
importSize = 50000;
url = "https://export.arxiv.org/oai2?verb=ListRecords" + ...
 "&set=math&metadataPrefix=arXiv";
options = weboptions('Timeout',160);
code = webread(url,options);
```

Import Text Data

- Parse the returned XML content and create an array of htmlTree objects containing the record information.

```
tree = htmlTree(code);
subtrees = findElement(tree,"record");
numel(subtrees)
```

Import Text Data

- Iteratively import more chunks of records until the required amount is reached, or there are no more records. To continue importing records from where you left of, use the resumptionToken attribute from the previous result. To adhere to the rate limits imposed by the arXiV API, add a delay of 20 seconds before each query using the pause function.

```
while numel(subtrees) < importSize
  subtreeResumption = findElement(tree,"resumptionToken");
  if isempty(subtreeResumption)
    break
  end
  resumptionToken = extractHTMLText(subtreeResumption);
  url = "https://export.arxiv.org/oai2?verb=ListRecords" + ...
  "&resumptionToken=" + resumptionToken;
  pause(20)
  code = webread(url,options);
  tree = htmlTree(code);
  subtrees = [subtrees; findElement(tree,"record")];
end
```

Extract and Preprocess Text Data

- Extract the abstracts and labels from the parsed HTML trees. Find the "<abstract>" elements using the findElement function.

```
subtreesAbstract = htmlTree("");
for i = 1:numel(subtrees)
  subtreesAbstract(i) = findElement(subtrees(i),"abstract");
end
```

Extract and Preprocess Text Data

- Extract the text data from the subtrees containing the abstracts using the extractHTMLText function.

```
textData = extractHTMLText(subtreesAbstract);
```

- Set aside 30% of the documents at random for validation.

```
numDocuments = numel(textData);
cvp = cvpartition(numDocuments,'HoldOut',0.1);
textDataTrain = textData(training(cvp));
textDataValidation = textData(test(cvp));
```

Extract and Preprocess Text Data

- Tokenize and preprocess the text data using the function preprocessText which is listed at the end of this example.

```
documentsTrain = preprocessText(textDataTrain);
documentsValidation = preprocessText(textDataValidation);
```

- Create a bag-of-words model from the training documents. Remove the words that do not appear more than two times in total. Remove any documents containing no words.

```
bag = bagOfWords(documentsTrain);
bag = removeInfrequentWords(bag,2);
bag = removeEmptyDocuments(bag);
```

Extract and Preprocess Text Data

- For the validation data, create a bag-of-words model from the validation documents. You do not need to remove any words from the validaiton data because any words that do not appear in the fitted LDA models are automatically ignored.

```
validationData = bagOfWords(documentsValidation);
```

Solvers for optimization

- Stochastic Solver
  - 'savb' – Use stochastic approximate variational Bayes. This solver is best suited for large datasets and can fit a model in fewer passes of the data.
- Batch Solvers
  - 'cgs' – Use collapsed Gibbs sampling. This solver can be more accurate at the cost of taking longer to run.
  - 'avb' – Use approximate variational Bayes. This solver typically runs more quickly than collapsed Gibbs sampling and collapsed variational Bayes, but can be less accurate.
  - 'cvb0' – Use collapsed variational Bayes, zeroth order. This solver can be more accurate than approximate variational Bayes but take longer time.

Fit and Compare Models

- For each of the LDA solvers, fit a model with 40 topics. To distinguish the solvers when plotting the results on the same axes, specify different line properties for each solver.

```
numTopics = 40;
solvers = ["cgs" "avb" "cvb0" "savb"];
lineSpecs = ["+-" "*-" "x-" "o-"];
```

- Fit an LDA model using each solver. For each solver, specify the initial topic concentration 1, to validate the model once per data pass, and to not fit the topic concentration parameter. Using the data in the FitInfo property of the fitted LDA models, plot the validation perplexity and the time elapsed.

Fit and Compare Models

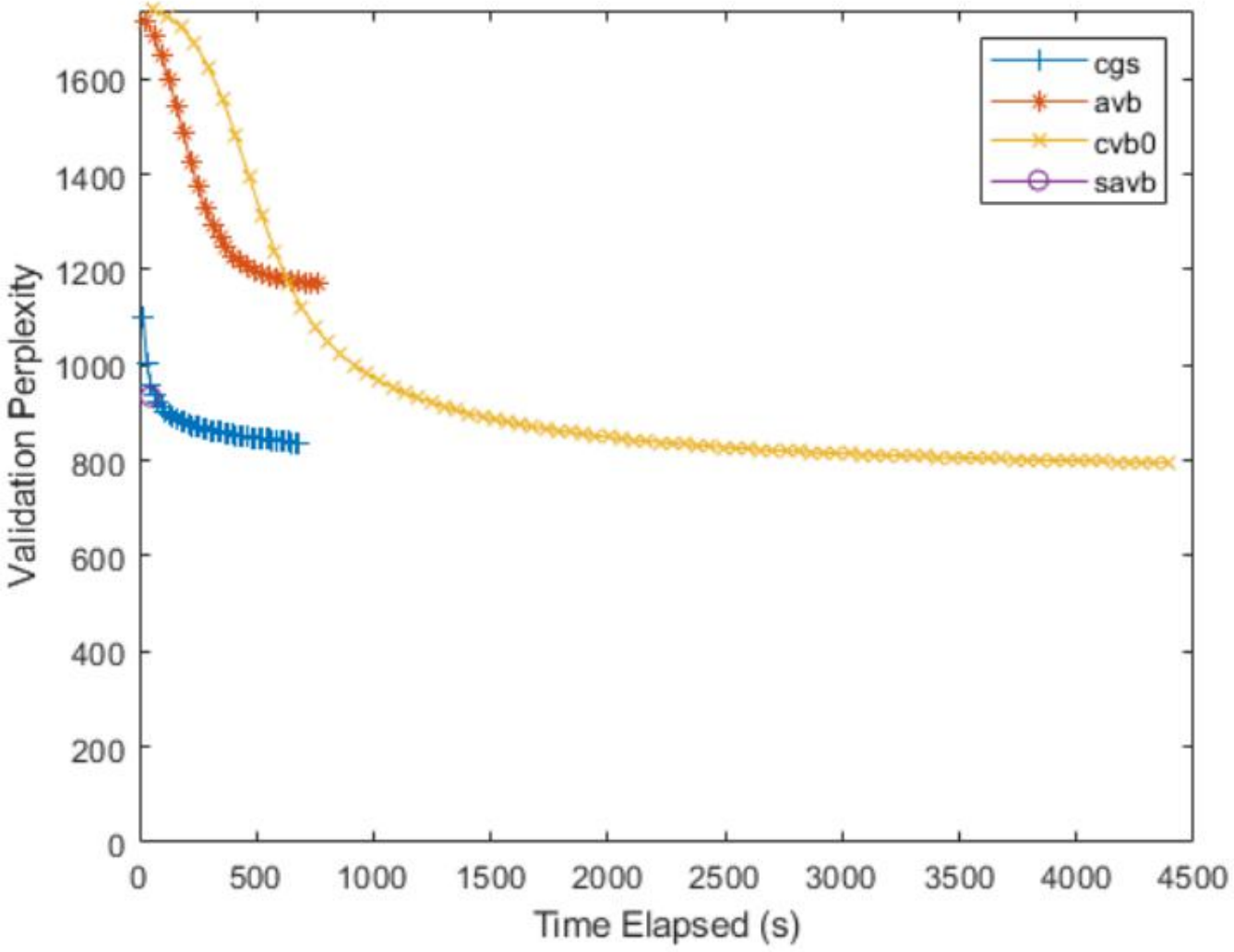
- The stochastic solver, by default, uses a mini-batch size of 1000 and validates the model every 10 iterations. For this solver, to validate the model once per data pass, set the validation frequency to ceil(numObservations/1000), where numObservations is the number of documents in the training data. For the other solvers, set the validation frequency to 1.
- For the iterations that the stochastic solver does not evaluate the validation perplexity, the stochastic solver reports NaN in the FitInfo property. To plot the validation perplexity, remove the NaNs from the reported values.

```
numObservations = bag.NumDocuments;
figure
for i = 1:numel(solvers)
  solver = solvers(i);
  lineSpec = lineSpecs(i);
  if solver == "savb"
    numIterationsPerDataPass = ceil(numObservations/1000);
  else
    numIterationsPerDataPass = 1;
  end
  mdl = fitlda(bag,numTopics, 'Solver',solver, 'InitialTopicConcentration',1, ...
  'FitTopicConcentration',false, 'ValidationData',validationData, ...
  'ValidationFrequency',numIterationsPerDataPass, 'Verbose',0);
```

```
    history = mdl.FitInfo.History;
    timeElapsed = history.TimeSinceStart;
    validationPerplexity = history.ValidationPerplexity;
    % Remove NaNs.
    idx = isnan(validationPerplexity);
    timeElapsed(idx) = [];
    validationPerplexity(idx) = [];
    plot(timeElapsed,validationPerplexity,lineSpec)
    hold on
end
```

Fit and Compare Models

```
hold off
xlabel("Time Elapsed (s)")
ylabel("Validation Perplexity")
ylim([0 inf])
legend(solvers)
```


cgs
avb
cvb0
savb
Validation Perplexity
Time Elapsed (s)
0
200
400
600
800
1000
1200
1400
1600
0
500
1000
1500
2000
2500
3000
3500
4000
4500

Fit and Compare Models

- For the stochastic solver, there is only one data point. This is because this solver passes through input data once. To specify more data passes, use the 'DataPassLimit' option. For the batch solvers ("cgs", "avb", and "cvb0"), to specify the number of iterations used to fit the models, use the 'IterationLimit' option.
- A lower validation perplexity suggests a better fit. Usually, the solvers "savb" and "cgs" converge quickly to a good fit. The solver "cvb0" might converge to a better fit, but it can take much longer to converge.

Fit and Compare Models

- For the FitInfo property, the fitlda function estimates the validation perplexity from the document probabilities at the maximum likelihood estimates of the per-document topic probabilities. This is usually quicker to compute, but can be less accurate than other methods. Alternatively, calculate the validation perplexity using the logp function. This function calculates more accurate values but can take longer to run.

Calculate Document Log-Probabilities from Word Count Matrix

- Load the example data. sonnetsCounts.mat contains a matrix of word counts and a corresponding vocabulary of preprocessed versions of Shakespeare's sonnets.

```
load sonnetsCounts.mat
size(counts)
ans = 1×2
 154 3092
```

Calculate Document Log-Probabilities from Word Count Matrix

numTopics = 20;

mdl = fitlda(counts,numTopics)

```
Initial topic assignments sampled in 0.0799309 seconds.
```

Iteration	Time per iteration (seconds)	Relative change in log(L)	Training perplexity	Topic concentration	Topic concentration iterations
0	0.23		1.159e+03	5.000	0
1	0.08	5.4884e-02	8.028e+02	5.000	0
2	0.08	4.7400e-03	7.778e+02	5.000	0
3	0.14	3.4597e-03	7.602e+02	5.000	0
4	0.09	3.4662e-03	7.430e+02	5.000	0
5	0.09	2.9259e-03	7.288e+02	5.000	0
6	0.10	6.4180e-05	7.291e+02	5.000	0

Calculate Document Log-Probabilities from Word Count Matrix

- Compute the document log-probabilities of the training documents. Specify to draw 500 samples for each document.

```
numSamples = 500;
logProbabilities = logp(mdl,counts, ...
 'NumSamples',numSamples);
```

Calculate Document Log-Probabilities from Word Count Matrix

- Show the document log-probabilities in a histogram.

```
figure
histogram(logProbabilities)
xlabel("Log Probability")
ylabel("Frequency")
title("Document Log-Probabilities")
```

Calculate Document Log-Probabilities from Word Count Matrix