CS410: Notes on Topics to be Covered in Exam 2

General advice

The format of Exam 2 is exactly the same as that of Exam 1, thus you can prepare for Exam 2 in the same way as you did for Exam 1. In general, the best way to prepare for the exam is to (1) **make sure that you have watched and digested all the lecture videos**; (2) go over the quizzes you've worked on and make sure that you know how to solve all the questions in those quizzes. (3) pay special attention to the topics listed below, especially the underscored topics; (4) ask questions on Piazza if you have trouble with the materials, and email us or visit our office hours if your questions cannot be resolved on Piazza.

You are expected to know the following for Exam 2 (**underscored topics are especially important**):

1. **know what is a paradigmatic relation and what is a syntagmatic relation**. Can you give an example of each?

   Paradigmatic and syntagmatic relations are two basic relations between words in natural language. Two words have a paradigmatic relation if they tend to occur in the *same* or similar positions in a sentence. Two words have a syntagmatic relation if they tend to *co-occur* with each other. When two words have a paradigmatic relation, if we replace one word with another word, the sentence would usually still be meaningful; in contrast, if we replace a word by another word that has a syntagmatic relation with it, the sentence would generally become non-grammatical or not make much sense.

2. **Know the general ideas for discovering paradigmatic and syntagmatic relations** from text.

   The general idea for discovering paradigmatic relations is to compute the similarity between the contexts of two words and assume that if their contexts have high similarity, there would be a paradigmatic relation between them. The general idea for discovering syntagmatic relations is to check whether two words tend to co-occur together often and assume that those words that "always" co-occur together have a syntagmatic relation. In other words, we can assess how easily we can predict presence or absence of word A based on knowledge about presence or absence of word B. Such a contextual representation of a word is also used in automated word sense disambiguation where the context of a particular mention of an ambiguous word would be compared with "typical" context of each possible sense of the word to determine which sense is the most likely sense intended in the current context. In addition, the representation of a word by a word context vector is also related to the word embedding technique (notably Google's word2vec) where the vector is an "implicit" vector learned to optimize prediction of co-occurrences of words in a large corpus instead of an explicit context as discussed here in

the lectures (if you are interested in word embedding, which is an active research area, you may read Section 14.3.3 in Chapter 14 of the reference textbook).

3. What are the **two deficiencies of the Expected Overlap of Words in Context (EOWC)** approach? Why do we want to do **Term Frequency Transformation** when computing similarity of context? Why do we want to do **Inverse Document Frequency (IDF)** weighting when computing similarity of context?

   While EOWC intuitively measures the overlap of the two context word vectors, it may "over-count" the overlap of a term that occurs many times in the two context vectors, letting such a term dominate the overall similarity score. This deficiency can be addressed by perform a sublinear term frequency transformation. Another deficiency of EOWC is that it doesn't distinguish matching a common word such as "the" from matching a content-carrying word like "computer". The problem can be addressed by adding IDF weighting.

4. **Know the concept and formula of entropy**. **For what kind of random variables does the entropy function reach its minimum and maximum, respectively**?

   Entropy is a function defined on a discrete probability distribution of a random variable. It measures the uncertainty (or randomness) of the random variable. The most random/uncertain variable is one with a uniform distribution, which has the maximum entropy, whereas a completely skewed distribution with probability 1.0 on a particular value has no uncertainty and thus has minimum entropy which is always zero. The formula of entropy function can be more easily remembered by interpreting it as the expected number of bits used to optimally compress values following the corresponding distribution, a point not mentioned in the lectures, but worth knowing. Specifically, with optimal variable-length encoding, the length of a code representing value x is $\log(1/p(x))$, resulting in a shorter code for frequent values (high $p(x)$) at the cost of using a longer code for a rare value (small $p(x)$).

5. **Know the concept of conditional entropy? What is the relation between conditional entropy H(X|Y) and entropy H(X)? Which is larger?**

   Conditional entropy $H(X|Y)$ measures the uncertainty of random variable X under the condition that we know values of another random variable Y. When we consider a specific value of Y, say y0, we may consider entropy $H(X|Y=y0)$, which is simply the entropy of the conditional distribution $p(X|Y=y0)$. However, $H(X|Y)$ is the expectation of $H(X|Y=y0)$ with respect to the distribution of random variable Y, i.e., we'll consider all the possible values of Y, leading to a summation over all the possible values of Y, thus no longer depending on any particular value of Y. $H(X|Y)$ can never be larger than $H(X)$ because knowing information about Y can only possibly reduce the uncertainty on X (making $H(X|Y)$ smaller than $H(X)$ in such a case), but can never increase our uncertainty on X. If X is completely determined by Y, what is $H(X|Y)$? If X and Y are completely independent, what is $H(X|Y)$?

6. **How can conditional entropy be used for discovering syntagmatic relations**?

   We may use a binary random variable Xw to denote presence and absence of a word w in a text segment (e.g., a sentence) and consider the conditional entropy H(Xw1|Xw2). If H(Xw1|Xw2) is smaller than a threshold, we may assume the two words w1 and w2 have syntagmatic relation (since in such a case, knowing presence/absence of word w2 helps a lot to predict whether word w1 is present or absent, which only happens when w1 and w2 are semantically related).

7. **Know the concept of mutual information I(X;Y)? How is it related to entropy H(X) and conditional entropy H(X|Y)?**

   Mutual information I(X;Y) measures the association of two random variables X and Y. It is equal to the reduction of entropy of one of the variables due to knowledge about the other variable. I(X;Y)=H(X) –H(X|Y) = H(Y)-H(Y|X). Note that mutual information is symmetric.

8. What's the **minimum value of I(X;Y)**? Is it **symmetric**?

   I(X;Y) reaches its minimum when the conditional entropy H(X|Y) (or H(Y|X)) reaches its maximum, which would be H(X) (or H(Y)). In such a case, it's easy to see that I(X;Y)=0, since I(X;Y)=H(X)-H(X|Y)=H(X)-H(X)=0. It's very easy to see mutual information is symmetric.

9. **For what kind of X and Y, does mutual information I(X;Y) reach its minimum? For a given X, for what Y does I(X;Y) reach its maximum?**

   As discussed above, I(X;Y) reaches its minimum (zero) when one of the conditional entropies is the same as the non-conditional entropy. For example, when H(X|Y)=H(X), I(X;Y)=0. In such a case, knowing Y does not help reduce uncertainty of X at all; in other words, in such a case, X and Y are independent. Note that in such a case, H(Y|X) must also be the same as H(Y), and I(X;Y)=H(Y)-H(Y|X)=0, reaching the same conclusion.

   For a given X, it's useful to note that I(X;Y) is upper-bounded by H(X), the entropy of X. Thus in general, if a variable X has less uncertainty, its mutual information with *any* other variable Y cannot be very large since I(X;Y)<=H(X). In contrast, if X is more uncertain, there would be more "room" for I(X;Y) to be larger. To make I(X;Y)=H(X)-H(X|Y) larger when X is fixed, we need to make H(X|Y) as small as possible. The smallest value of H(X|Y) is zero which happens when X is completely determined by Y. Thus in such a case, I(X;Y)=H(X) reaches its maximum. Note that I(X;Y) can have two potentially different maximum values, i.e., H(X) and H(Y). Indeed, it's upper-bounded by both H(X) and H(Y). If the two upper bounds are different, then only the smaller one is attainable. (Can you see why? Hint: if H(X) and H(Y) are different, then one of them must be more random than the other. What can we say about H(X|Y) and H(Y|X) in such a case?)

10. **Why is mutual information sometimes more useful for discovering syntagmatic relations than conditional entropy**?

    While for a given X, ranking Y according to H(X|Y) and I(X;Y) gives the same results (since I(X;Y)=H(X)-H(X|Y) and H(X) is a constant), if we want to rank pairs of (X,Y) (i.e., pairs of words) according to H(X|Y), it would be inappropriate because H(X|Y) may not be comparable across different word pairs. In such a case, ranking pairs of (X,Y) based on I(X;Y) which is more comparable. I(X;Y) can be seen as normalizing H(X|Y) by H(X).

11. **Know** that **the two basic tasks in mining and analyzing topics** in a collection of text documents are 1) discover the major covered in the collection; and 2) figure out to what extent each document covers each topic.

    Mining and analyzing topics in a collection of text documents has many applications. The two basic tasks involved in mining and analyzing topics are to 1) discover the major topics covered in the collection and 2) figure out the coverage of each topic in each document. The basic assumption made here is that there are a certain number of topics covered in the entire collection, and different documents may cover different topics and may also cover multiple topics. The coverage of topics in a document can usually be represented as a distribution over all the possible topics. In the extreme case, the distribution may be entirely concentrated on one topic, which would mean that the document only covers one single topic. In general, however, a document may cover multiple topics. Once we know the coverage of topics in each topics, we may compute the coverage of topics in any subset of documents (e.g., all the documents with a particular time stamp or written by authors from the same location), thus enabling many interesting ways to analyze patterns of topics (e.g., temporal trends of topics). The topics discovered can tell us the content of the collection, and can allow us to compare the topic content in two different collections (e.g., two collections corresponding to two different time periods).

12. **How can we heuristically solve the problem of topic mining and analysis by treating a term as a topic? What are the main problems of such an approach?**

    The simplest way to define a topic is to define it as one term. Under this assumption, the problem of discovering K topics would be to identify K terms representing K different topics, which can be done heuristically by scoring a term based its frequency in the collection etc. The task of computing topic coverage in each document would be reduced to estimate the coverage of each of the K terms in a document, which can be based on counts of terms in the document. Such a simple approach to representing a topic has a few problems. One is that a single term can only represent a simple topic; a complex topic would require multiple terms. The approach can't address the ambiguity problem either. Finally, when estimating the coverage of a topic in a document, we can't easily consider related terms to the term representing a topic.

13. **What are the advantages of representing a topic by a word distribution over representing a topic by one single term?**

    Representing a topic by a word distribution can be regarded as a generalization of representing a topic by one single term since when the distribution gives a probability of 1.0 to one single term, it would be equivalent to representing a topic by one term. The generalization addresses all the three problems of representing a topic by a term mentioned above since we can now bringing in additional words to describe a topic, thus can also characterize and distinguish complex topics. This also addresses word ambiguity by allowing an ambiguous word to contribute to multiple topics (with different weights).

14. **know what is a statistical language model, what is a unigram/bigram language model**

    A statistical language model (SLM) is a distribution over word sequences. Intuitively, it gives us a probability for any sequence of words, thus allows us to compare two sequences of words to see which has a higher probability. In general, SLMs help capture the uncertainties associated with the use of natural language. For example, in general, non-grammatical sentences would have much smaller probabilities than grammatical sentences. Specialized language models can be used to answer many interesting questions that are directly related to many information management tasks.

    While there are many different kinds of SLMs, we are particularly interested in the simplest one, i.e., the unigram language models. This model corresponds to a multinomial distribution over words. According to this model, a piece of text is "generated" by generating each word *independently*. As a result, the joint probability of generating all the words in a document D=w1 w2 ... wn is simply the product of generating each individual word, i.e., p(D)=p(w1)p(w2)...p(wn). Note that in general, the generation of one word may depend on another. For example, having seen "web search" being generated would make the probability of further generating a word like "engine" much higher. This means that p(w3="engine" |w1="web", w2="search") is much higher than p(w3="engine"). Thus the independence assumption made by the unigram language model doesn't really hold in reality. Indeed, with a bigram LM, we'd have p(D)=p(w1)p(w2|w1)p(w3|w2)...p(wn|wn-1), which would capture local dependency between two adjacent words.

15. **know the idea of maximum likelihood (ML) estimator, and how to estimate a unigram language model with an ML estimate**:

    You should know that the Maximum Likelihood (ML) estimator is to find an optimal setting of the language model p(w|theta) (i.e., optimal setting of the probability of each word in our vocabulary p(w|theta)) so that p(d|theta), or equivalently log p(d|theta), would achieve the maximum value. In other words, if we set these word probabilities to different values than their ML estimate, p(d|theta) would be smaller. The ML estimate is

optimal in the sense that it maximizes the likelihood of the observed data, i.e., it finds the parameter setting that best explains the data. However, when the observed data sample is too small (e.g., the title of a document), it may be a biased representation of the entire population (e.g., the whole article), so if we overfit the observed data as the ML estimator would do, our estimated parameter values may not be optimal. For example, we would assign zero probability to all the unseen words (since the ML estimator would try to give as much probability mass to the observed words as possible in order to maximize the likelihood of data).

You should know the fact that the ML estimate of a multinomial distribution (i.e., a unigram language model) would give each word w a probability equal to the relative frequency of the word. That is, if the word distribution is theta, and the observed data is a document d, according to the ML estimator, we would have p(w|theta)=c(w,d)/|d| where c(w,d) is the count of word w in d, and |d| is the length of document d (i.e., total counts of words in d). From topic discovery perspective, this also means that if we assume that one document covers precisely one topic, then we may use the ML estimate to "discover" the single topic covered in a given document.

16. **Know the basic idea of Bayesian estimation. What is a prior distribution? What is a posterior distribution? How are they related with each other? What is Bayes rule?**

The main idea of Bayesian estimation is to assume that every parameter is itself a random variable (thus has a distribution over all possible values of the parameter). We can then consider a prior distribution over the possible values of a parameter, which can represent any additional knowledge we'd like to bring in for estimating the parameters. That is, we can pretend we already have a bias toward which value is likely the value of the parameter *before* we even observe the data, and this belief/bias is captured by a prior distribution. We would then be interested in computing the posterior distribution of the parameter which denotes our updated belief about which value is more likely the value of our parameter *after* we observe the data. The posterior distribution and prior distribution are related through the likelihood of the observed data, according to Bayes Rule. Specifically, if P(H) is the prior on our hypothesis H, p(D|H) is the probability of observing data D given that hypothesis H is true, then the posterior distribution p(H|D) is proportional to p(D|H)*p(H). More precisely, p(H|D)=p(D|H)*p(H)/p(D), where p(D) is a sum over all possible ways to generate data D, i.e., p(H1)*p(D|H1)+p(H2)*p(D|H2) +…. p(Hk)*p(D|Hk), where H1, …, Hk are the set of all hypotheses we consider. In the case of parameter estimation, each hypothesis can be a possible value of the parameter, but in other applications of Bayes Rule, hypotheses can be anything that we are interested in inferring.

In Bayesian estimation, once we have a posterior distribution p(H|D), we may further compute a point estimate, i.e., converting the posterior distribution into one single value (e.g., its mean or its mode). But we may also preserve the uncertainty by retaining the entire posterior distribution for "future" use to infer other variables that depend on the parameter. For example, if there's another variable Y = f(H) depending on hypothesis H, we can compute the expected value of Y, by computing the sum f(H1)p(H1|D) +

f(H2)p(H2|D) + …. f(Hk)p(Hk|D), which is the expected value of Y with respect to the posterior distribution over H.

17. <u>Know the concept of a mixture model. In general, how do you compute the probability of observing a particular word from a mixture model? What is the general form of the expression for this probability?</u>

A mixture model is a probabilistic model with multiple component models "mixed" together, which has two types of parameters. One type is a "switch" to determine which component model to use when generating a data point. The other type is the parameters characterizing each individual component distribution, which control what kind of data would be generated from each component model. When generating a data using such a mixture model, we would first decide which component model to use (using the component model selection "switch"), and then sample a data point from the chosen component model. In our applications, we are most interested in a mixture of unigram language models (i.e., word distributions). Each unigram language model denotes a topic. Thus if we have k topics, we would have k component models, each being a unigram language model. Let these k unigram language models be p(w|theta1), p(w|theta2), …, p(w|theta_k). Our mixture model thus needs to have a parameter to control the frequency of each component model to be used, which we denote by p(theta1), p(theta2), …., p(theta_k), (they sum to one since we assume we must always choose one of these k topics). The probability of observing a word w from such a mixture model is the sum over all the possible k ways of generating the word, each corresponding to using one of the k component models, i.e., p(theta1)p(w|theta1)+p(theta2)p(w|theta2)+… p(theta_k)p(w|theta_k). In the sum, each term is a product of probability of choosing a component model p(theta_i) and the probability of observing word w from the chosen model p(w|theta_i). It's very important that you know this general form of the likelihood function of a mixture model as it's the key to understanding any mixture model, and definitely essential to understanding a topic model such as PLSA or LDA. If all the parameters of a mixture model are given, you should know how to compute the probability of observing a word from such a mixture model.

From another perspective, the probability of observing word w from a mixture model (i.e., the sum above) can be seen as the probability of observing our data p(D) when using Bayes Rule. Specifically, suppose we take each theta_i as a hypothesis H_i, then we see p(theta_i) is simply our prior on the hypothesis p(H_i), which indicates our prior belief of which model has been used to generate a word (or any data) *before* we actually see the word/data. In such a case, the posterior probability p(H_i|D) given an observed data allows us to "guess" (infer) which component has been used to generate an observed data D. According to Bayes Rule, p(H_i|D)=p(H_i)p(D|H_i)/p(D), where p(D) is precisely the sum mentioned above, i.e., p(H_1)p(D|H_1)+….+p(H_k)p(D|H_k). Can you see what is p(D|H_i) in our case when D is a word? The inference of which component model has been used to generate a data point is key to the Expectation-Maximization (EM)

algorithm; indeed, the E-step is precisely to use Bayes Rule to make this inference to "decode" which component model has been used to generate a data point.

18. **What does the maximum likelihood estimate of the component word distributions of a mixture model behave like? In what sense do they "collaborate" and/or "compete"? Why can we use a fixed background word distribution to force a discovered topic word distribution to reduce its probability on the common (often non-content) words?**

    The multiple components in a mixture model work together to maximize the likelihood of the observed data when computing the Maximum Likelihood (ML) estimate. They "compete" in the sense that they all want to give a higher probability to the data points that occur very frequently in the data (so as to maximize the data likelihood). However, they collaborate by placing high probabilities on somewhat different words to collaboratively assign high probabilities to frequent data points in the observed data. If one component model (e.g., model theta_B) already gives high probabilities to some frequent words, another component model would tend to give high probabilities to other frequent words that have not received high probabilities from theta_B. This is why using a fixed background word distribution would "explain away" the common words (stop words) in the collection, thus encouraging other component models to be more discriminative and assign high probabilities to content-carrying words instead of common words such as those stop words which can already be "accounted for" by the background model.

19. **What is the basic idea of the EM algorithm? What does the E-step typically do? What does the M-step typically do? In which of the two steps do we typically apply the Bayes rule? Does EM converge to a global maximum?**

    The EM algorithm is an algorithm for computing the Maximum Likelihood estimate of a mixture model (or in general, a model with latent variables). As such, its output is the values for all the parameters that we want to estimate. It is an iterative hill-climbing algorithm that would start from a random initialization of all the parameters and gradually improve the parameter values by executing an E-step followed by an M-step. There is a theoretical guarantee that by doing E-step and M-step, we would obtain a new generation of parameter values that would increase the likelihood function. This way, it would eventually converge to a local maximum of the likelihood function. Note that it cannot be guaranteed to converge to a global maximum. Thus we often need to try different trials starting from different random initializations and see which one gives us the highest likelihood value. The E-step of the EM algorithm is to infer the values of latent/hidden variables (which indicate which word distribution has been used to generate a word in our topic model applications) by using Bayes Rule. Since at any iteration, we always have a set of (tentative) values for all the parameters, with Bayes Rule, we can obtain a distribution on all the possible topics representing how likely a topic has been

used to generate a word. This distribution is then used to split the counts of the word among all the topics, and the split counts would then be re-collected appropriately to compute a re-estimate of all the parameters in the M-step.

20. What is PLSA? How many parameters does a PLSA model have? How is this number affected by the size of our data set to be mined?

PLSA stands for Probabilistic Latent Semantic Analysis, It's a mixture model with unigram language models as component models that can be used to discover k topics from a collection of documents and compute coverage of these k topics in each document in the collection. These results can further be used to develop many interesting applications as we covered later in the course.

If there are k topics and N words in the vocabulary and M documents in the collection, then a PLSA model would have the following parameters: 1) k word distributions with each distribution having N parameters, corresponding to N word probabilities (strictly speaking, N-1 free parameters since they must sum to one), giving a total of k*(N-1) free parameters. 2) Each of the M documents has a distribution over k topics, giving a total of M*k parameters for topic coverage (more precisely M*(k-1) free parameters due to constraints on probabilities). The data set size directly affects the second set of parameters (i.e. topic coverage), but doesn't really affect the first set (topic characterization), which is affected by the vocabulary size and the number of topics to discover.

21. How is LDA different from PLSA? What is shared by the two models?

LDA stands for Latent Dirichlet Allocation. It is a Bayesian version of PLSA, which means that it adds prior distributions on all the parameters of PLSA. LDA is meant to address two deficiencies of PLSA: 1) PLSA is not a generative model (for generating (new) documents), and sometimes, we need a generative model as in the case of text categorization (see discussion later about this topic). It is impossible to compute the probability of a new (unseen) document D given a PLSA model since its parameters are tied to specific documents and can't be used to generate a new unseen document. 2) PLSA has many parameters, leading to a likelihood function with many local maxima (making it hard to compute the ML estimate). Recall that PLSA has two types of parameters: 1) topic coverage in each document; 2) word distribution characterizing a topic. LDA uses a Dirichlet prior distribution on both types of parameters and assumes that these parameters are themselves values drawn from a Dirichlet prior distribution. Thus LDA has only two sets of parameters corresponding to the two Dirichlet prior distributions respectively, significantly fewer parameters than PLSA. Other than the use of prior, however, LDA is exactly the same as PLSA. That is, they share the mixture model with multiple unigram language models.

Note that although LDA has fewer parameters, after parameter estimation, LDA cannot give us directly the word distributions characterizing topics, nor the coverage of topics in

each document (since they are no longer parameters!). Thus, in order to obtain such knowledge that we need for topic mining and analysis, we must use posterior inference. That is, we will need to infer this additional information based on the estimated parameters. This posterior inference, unfortunately, runs into the same difficulty as in PLSA where we have many parameters, and thus both PLSA and LDA tend to perform similarly in practice. They often produce very similar results.

22. What is clustering? What are some applications of clustering in text mining and analysis?

Clustering is a general data mining technique for discovering patterns in data, where data points that are similar would be grouped together to form a cluster. The problem is clearly not well defined until we also define what we meant by "similar". Depending on the applications, we may need to use a different notion of similarity even for clustering the same data set. For text clustering, the objects to be clustered can be terms, sentences, passages, articles, or even collections. Text clustering can also be used to cluster other entities that are associated with text, e.g., clustering authors based on the clustering their publications. Clustering is often very useful for getting an overview of what the data looks like. For example, clustering of documents in a collection can reveal what are the main topics covered in the collection. Clustering also enables creation of structures over a collection of text documents, thus facilitating browsing.

23. How does hierarchical agglomerative clustering work? How do single-link, complete-link, and average-link work for computing group similarity? Which of these three ways of computing group similarity is least sensitive to outliers in the data?

Hierarchical Agglomerative Clustering (HAC) is a bottom-up approach that requires a similarity function to be given for measuring similarity between any two data points. It gradually merges data points that have the highest similarity to form a hierarchy of clusters. It needs to assess similarity of two groups of data points based on the provided similarity function defined on the original data points, and there are multiple ways to do this. Single-link defines the similarity between two groups as the similarity of the closest pair of data points from the two groups, whereas complete-link uses the similarity of the farthest pair of data points. Average-link takes the average of the similarity of all the pairs formed by the data points from the two groups. Average-link is more robust against outliers than single-link or complete-link since both single-link and complete-link use just one single pair of data points to compute the similarity and thus can be affected significantly by an outlier pair that is extremely close (for single-link) or extremely far apart (for complete-link).

24. How does K-means clustering work?

K-means clustering is an iterative algorithm for producing k clusters of data points. It starts with randomly chosen k centroids for representing k tentative clusters respectively. It would then take two steps, an allocation step and a step to re-calculate centroids, very similarly to the E-step and M-step of the EM algorithm. In the allocation step, it would allocate every data point into the cluster whose centroid is the closest to the data point. Then in the re-calculation step, it would use the allocated data points in each cluster to re-calculate its centroid, leading to a new generation of centroids that can be guaranteed to generate better clusters than the previous generation. The algorithm is guaranteed to converge to a local optimum for an objective function that would minimize the average distances of data points in each of the k clusters. Overall, the algorithm is very similar to the EM algorithm for clustering with a mixture model in that the allocation step is similar to the E-step only that in the E-step, the allocation of data points into different clusters is done in a weighted manner (i.e., "soft" allocation allowing a data to belong to potentially multiple clusters) whereas in K-Means a data point can only be assigned to one cluster.

25. How do we evaluate clustering results?

Clustering results can be evaluated directly by comparing the clustering results of an algorithm with the "ideal clustering results" produced by humans. When there's a mismatch between them there are multiple ways to quantify the errors, leading to different measures. Clustering results can also be evaluated indirectly in terms of their contribution to an application. For example, if clustering is to be used for improving a search interface (clustering search results), we may compare an interface with clustering support with a baseline interface that doesn't have clustering support to see which one helps a user finish a retrieval task more quickly or more easily.

26. <u>What is text categorization</u>? What are some applications of text categorization?

Text categorization is to classify a text object into one or many of the pre-defined categories. The categories can vary according to different applications, ranging from categories that can be easily predicted based on presence or absence of keywords (as in the case of subject categorization of a research article) to categories that may be very weakly correlated to surface features in text data (as in the case of categorizing a debate speech of a politician into different ideological parties). Thus there are many applications one can think of. From text data analysis perspective, it might be interesting to distinguish two kinds of applications: annotations vs. text-based prediction. In the case of annotations, the categories are used to enrich the representation of text data. For example, topic categories and sentiment tags can enrich semantic representation of documents and support semantic queries such as finding all positive articles about a topic. In other words, they help us understand the meaning of text data in more depth. In the case of prediction applications, the categories may have little to do with the text content (at least on the surface). One example is to classify the news articles today into two categories: positive for stock market vs. negative for stock market. In such a case, we are leveraging

the text data as "signals" we can observe about the world to infer the value of another variable that might be correlated with text data (i.e., stock prices in this case). It is more natural to call this kind of categorization a prediction task.

27. What does the training data for categorization look like?

The training data for a text categorization task contains a set of text documents, each labeled with one or more categories that should be assigned to a document. That is, in the training data, we already know the correct categories that should be assigned to each document.

28. How does the Naïve Bayes classifier work?

Naïve Bayes classifier is a special case of a generative model for classification where the main idea is to use Bayes Rule to infer the category of a text object. Specifically, let H1…Hk be the k categories (think about each category as a "hypothesis") and D be a document (i.e., observed evidence about which hypothesis is true). We can compute the posterior probability of hypothesis given D, i.e., p(Hi|D) and assign to D the Hi that has the highest posterior probability. For this purpose, we can note that p(Hi|D) is proportional to p(Hi)*p(D|Hi) where p(Hi) is the prior distribution of hypothesis, and p(D|Hi) is a generative model of text data (i.e., a statistical language model). Thus in such a generative model approach to categorization, we mainly need to estimate two kinds of parameters. The first is the prior p(Hi) which can be estimated based on the popularity of categories in our training data. The second is whatever parameters we have in the generative model p(D|Hi). They generally can also be estimated by using training data. For example, with Maximum Likelihood estimation, we would set the parameters of the model p(D|Hi) to values that would give the observed training documents with category Hi the highest probability. Note that we can do the estimation for each category Hi separately here by using *only* the documents labeled with category Hi. That is, p(D|Hi) and p(D|Hj) are estimated independently.

The simplest generative model is a unigram language model where we assume that a sequenced of words in our text object D=d1 d2 …dN (where di is a word), are generated by generating each word independently. Thus the generative model p(D|Hi) in this case would be a word distribution and thus has as many parameters as the number of words in the vocabulary. When we use such a simple model for categorization, the classifier is called Naïve Bayes. The word "Naïve" indicates that the model is indeed overly simplistic because there are always dependencies between the words in a document and thus assuming unigram language models is far from accurate, thus "Naïve".

Note that the generative model p(D|Hi) can also be based on a topic model like LDA. In such a case, each category has its own LDA model with its own parameters. For such an application, PLSA cannot be used because it is not a generative model and thus LDA

clearly has an advantage. (As mentioned before, for the purpose of discovering topics from text data, LDA and PLSA work similarly, though)

29. Why do we often use logarithm in the scoring function for Naïve Bayes?

Using logarithm helps preserve precision in the computation since computation of p(Hi|D) involves multiplication of many small numbers, which can cause underflow. By taking logarithm, we can preserve the precision while still maintaining the ordering of p(Hi|D) (thus not affecting the decision regarding which category to assign to D) as logarithm is a monotonic transformation. Such a technique is often used when we face the problem of underflow or overflow. Another way to solve such a problem is to use an appropriate normalizer to scale the numbers to a "comfortable" range, thus also avoiding losing precision.

30. What's the general idea of the logistic regression classifier? How is it related to Naïve Bayes? Under what conditions would logistic regression cover Naïve Bayes as a special case for two-category categorization?

Logistic regression is a widely used discriminative classifier, which attempts to compute the conditional probability p(Hi|D) directly without using Bayes Rule. Instead, it assumes that p(Hi|D) is a function of a set of features to be defined in specific applications. Specifically, it assumes that log [p(Hi|D)/(1-p(Hi|D))] is a linear combination of a set of features: beta0+beta1*X1+…beta_n*Xn with beta0, beta1, …, beta_n parameters that would have to be estimated based on training data. In the case of binary categorization, we can show that the classifier function of Naïve Bayes also boils down to a linear combination of word-based features. Thus logistic regression can be regarded as a generalization of a Naïve Bayes classifier since it not only covers Naïve Bayes as a special case, but also can accommodate many other features that we'd like to put into a classifier. Try to figure out what is beta and what is X when we write the Naïve Bayes classifier for binary categorization in the form of a logistic regression classifier. If you do this, you will see that Naïve Bayes uses words as features and provides a specific way of setting the parameters beta0, beta1, …, beta_n.

31. What's the general idea of the k-Nearest Neighbor classifier? How does it work?

K-Nearest Neighbor (K-NN) is another popular discriminative classifier that estimates p(Hi|D) based on the training data similar to D. Specifically, it would first find the K training documents that are most similar to D (i.e., the K nearest neighbors of D) and look at their (known) labels of category to see which is most popular which would then be assigned to D. Normalizing the counts of different category labels in the K nearest neighbors would essentially give us an estimate of p(Hi|D) with the highest probability assigned to the H_i that is most popular among the K nearest neighbors, and this category with the highest estimated probability would be assigned to D. Clearly, the performance of K-NN is highly affected by the choice of

the similarity function which further affects which data points would be counted as the nearest neighbors.

32. How do we evaluate categorization results?

Categorization results are usually evaluated by comparing the categorization decisions made by a system with the correct categorizations that are created by humans. Depending on how we treat the errors made by the system, we may have potentially many different ways to design a metric for quantitative evaluation of text categorization results.

33. How do we compute classification accuracy, precision, recall, and F score? Why is classification accuracy not suitable for evaluating binary categorization when the two categories are unbalanced (one with many more instances than the other)?

Classification accuracy is the percentage of correct decisions made by a categorization method among all the categorization decisions. When the two classes in a categorization problem are unbalanced (i.e., one with far more instances than the other), classification accuracy is not a good measure because a simple baseline where we simply label everything as the majority label would do quite well. In such a case, it's important to look at the results for each document or in each category. When looking at the results for each document (or each category), we may also compute precision, recall and F score by treating the categorization task as a "retrieval" task (i.e., retrieving "relevant categories" if we focus on the results for each document, or retrieving "relevant documents" if we focus on the results for each category).

34. Why is harmonic mean as used in F better than the arithmetic mean of precision and recall?

Because the harmonic penalizes extreme values such as a zero precision of zero recall. Imagine if we look at the results for each document and we assign all categories to the document, this would result in a perfect recall (Recall =1.0), but with a very low precision. However, if we use an arithmetic mean, we'd still have a mean value of at least 0.5!

35. **What's the difference between macro and micro averaging**?

Macro and micro averaging are different ways to aggregate classification metrics when we first compute a metric on a per-document or per-category basis. If we view each category (or document) as equally important, we would first compute a metric such as precision or F1 for each category (or document) and then take the average of all these category-level (or document-level) metrics. This is macro averaging. However, if we view each document-category combination as a unit and treat all such combinations as equally important, then we'd pool all these decisions together without respecting boundaries of documents or categories. This is called micro averaging. It's often useful to

look at both macro and micro averaging since they provide different perspectives of the performance.

36. Why is it sometimes interesting to frame a categorization problem as a ranking problem?

Categorization results are often given to humans for further validation, and humans can generally only examine one result at each time. Thus prioritizing the categorization results so that humans would first examine the most accurate categorization results would be important. In such an application, the categorization problem can be framed as a ranking problem and should be evaluated as a retrieval task by using ranking measures such as Mean Average Precision.

37. What is an opinion? How is it different from a factual statement? What's an opinion holder? What's an opinion target?

An opinion is a subjective statement by someone about something, where "someone" is the opinion holder, and "something" is the opinion target. An opinion differs from a factual statement in that an opinion cannot be proved to be correct or wrong.

38. What's the goal of opinion mining?

The goal of opinion mining is to extract a detailed representation of opinions in text data, specifically including the identification of opinion holder, opinion target, opinion content and opinion context. Opinion content may also be summarized as positive vs. negative opinions to indicate sentiment. Such a basic opinion representation can then be aggregated to provide more informative opinions about a target (e.g., obtaining the percentage of positive opinions on the battery of a laptop).

39. What is sentiment classification? How is it similar to and different from a text categorization task such as topic categorization?

Sentiment classification is to annotate a text object with a label to indicate sentiment (e.g., positive vs. negative vs. neutral or multiple levels of rating). It can be regarded as a special case of text categorization with sentiment polarity as categories, thus all the text categorization methods can be potentially used for sentiment classification. However, it's different from topic categorization in that more sophisticated features than just words are often needed for effective sentiment classification. Also, the categories form an order and thus have interesting dependency relations.

40. What's the concern of using too many complex features such as frequent substructures of parse trees?

While complex features are very discriminative and thus would enable accurate separation of different categories in the training data, they do not generalize well in the unseen test data, causing overfitting (to training data). Ensuring a certain level of frequency of a feature in the training data is one way to avoid overfitting (i.e., eliminating very rare features).

41. What are some commonly used features to represent text data?

There are a wide range of features that one can extract from text data. The commonly used ones include word n-grams (including unigrams, bigrams, trigrams etc), part of speech tags, partial parsing structures, frequent patterns of words or a combination of words and syntactic structure information. Topics discovered by using a topic model like PLSA or LDA can also be used as features.

42. Why is text-based prediction interesting from an application perspective? Why are humans playing an important role in text-based prediction? What is the general "data mining loop"?

When we view text data as the data generated by humans as subject sensors of our world, we can see that text data can potentially contain signals for all kinds of prediction problems. Thus in principle, we can collect all the text data we can possibly collect and try to predict any interesting variable such as sales of products, stock prices or which presidential candidate might win or whether two companies might merge etc. Obviously, some prediction tasks are much easier (e.g. predicting whether a product would sell well is relatively easy based on product reviews), while others are very hard (e.g., predicting stock market trends). Humans play an important role in text-based prediction because humans are able to understand text data much better than computers, so to be successful, we must inject as much human intelligence as possible. Furthermore, we usually also have to combine all kinds of data, including both text data and non-text data in an application problem.

43. Why is it necessary and useful to jointly mine and analyze text and non-text data? How can non-text data potentially help in analyzing text data? How can text data potentially help in mining non-text data?

In general, the more data we use, the more knowledge we can potentially discover. Non-text data can provide context for analyzing text data. Specifically, non-text data can be used to partition text data in interesting ways and thus enable interesting comparison of topics discovered from different partitions of text data (e.g. ,text data may be partitioned based on time, location etc). Text data can help interpreting any patterns discovered from non-text data. For example, a pattern generally can be used to separate non-text data matching the pattern from those that don't match the pattern. To interpret the potential meaning of the pattern, we may look at the corresponding text data associated with each of the two subsets and try to find the differences in topics which may help interpret the pattern originally discovered from non-text data.

44. Can you give some examples of context of a text article? How can we partition text data using context information? Can you give some examples where we can leverage context information to perform interesting comparative analysis of topics in text data?

Any meta data of a text article (e.g., time, author, author's affiliation etc) can be regarded as the context of the text article. More generally, we may also view any other data associated with the meta data, or very generally, any data that can be linked to the text data in some way as context. When we view text data together with context, we may picture a table with the first column having text data and other columns having context variables (e.g. ,one column has time, another column has location etc). Then any way to cluster the context variables could potentially lead to the partitioning of text data on the first column (e.g., each combination of time and location can form a cluster, which could partition text data into many subsets each corresponding to a particular time and a particular location, enabling analysis of spatiotemporal trends of topics).

45. What's the general idea of Contextual Probabilistic Latent Semantic Analysis (CPLSA)? How is it different from PLSA?

The general idea of CPLSA is to further introduce context variables into PLSA so that the topic coverage distribution and the variations of word distributions characterizing a topic would both depend on the context variables, thus the model would give us a conditional model of text data in the form of p(TextData|ContextVariables). After estimating the parameters, we would be able to obtain topics conditioned on a particular context variable such as time or location.

46. Can you give some examples of interesting topic patterns that can be found by CPLSA?

This should be an interesting exercise for you to work on!

47. What's the general idea of using CPLSA for analyzing the impact of an event? Can you think of an interesting application of this kind?

The general idea of using CPLSA to analyze the impact of any event is to compare the text content before an event and the content after the event to examine the similarity and differences. You should be able to easily come up with examples of such applications. We encourage you to discuss potential ideas on Piazza with others.

48. What's the general idea of using the social network of authors of text data as a complex context to improve topic analysis for text data? Can you give an example of an interesting application of this kind?

The main idea of using the social network of authors of text data as a complex context to improve topic analysis is to assume that text produced by authors that are well connected on the social network would tend to cover similar topics. This network-based constraint

can influence the discovered topics and make the topics more aligned with the network structures (e.g., a group of authors that have strong connections with each other may have a set of "favorite" topics attached to the group). Again, we encourage you to think about interesting applications of this kind.

49. What's the general idea of using a time series like stock prices over time to supervise the discovery of topics from text data? Can you give an example of an interesting application of this kind?

The idea of using a time series to supervise topic discovery is to discover meaningful topics whose coverage in the text stream (over time) is strongly correlated with the provided time series data. This is useful for predicting the time series data based on text data and may also help explain the cause or impact of sudden changes in the time series data. For example, by looking at the topics that can predict (future) changes in the time series, we can predict the time series in the future, whereas the topics that tend to follow a sudden change in the time series may indicate the impact of the time series. Once again, we encourage you to think about interesting applications of this kind of joint analysis.