Department of Statistics 2019

Tag Recommendation for Stack Overflow Questions

Sahra Ghalebikesabi

Submitted for the Master of Science,
London School of Economics,
University of London

May 2019

Contents

Abstract **1**

1 Introduction **1**

2 Related Work **1**

3 Data **2**

4 Classification **3**
Multinomial Naïve Bayes 4
Discriminant Analysis 4
Logistic Regression 5
Neural Networks. 5

5 Clustering **7**
K means 7
Mixture Models 8
Latent Dirichlet Allocation 9

6 Discussion **10**

References **12**

Abstract

With the rise of online communities where labeling of web content got indispensable, automatic tag recommendation has arisen interest in recent machine learning literature.

This paper aims to present and compare competing methods to predict tags of questions asked on Stack Overflow in a non-technical manner. We start by an introduction into the topic and related work, and proceed by explaining crucial text preprocessing steps. We then approach the problem of tag recommendation from two perspectives: (1) tag recommendation as classification problems and (2) as clustering problems.

1 Introduction

In the last years human-based tagging of online content has been introduced in a wide-range of online-communities. Song et al. (2011) define tagging as the action of associating a relevant user-defined keyword to an entity such as a document. While users create hashtags for their Twitter tweets or authors define keywords for their research papers, the question-answering platform for developers, Stack Overflow, asks their users to manually assign tags to questions they post. Benefits of tags are twofold (Krestel and Fankhauser, 2009). They help other people to filter questions to a specific topic and experts to get notifications when new questions to topics they are interested in are posted. A growing body of literature (Diakopoulos and Shamma, 2010; Tsur and Rappoport, 2012) points out the importance of human-based tagging in query-based searches.

Recently, research also focused on understanding the underlying processes of choosing particular tags (Stanley and Byrne, 2013). Accurately modeling tag creation could help to propose users tags based on the text of their post. As a result, users could be introduced to tags of their interest. Further, different orthographic versions or synonyms of a topic can be mapped to the same tag. In order to assure appropriate tagging, it is thus desirable to have a system that automatically suggests relevant tags to a user. There are two approaches to this problem: user-centered approaches, where the recommendation is based on past user behavior (Wang et al., 2010) and document-centered approaches, where only the post's content is taken into account (Song et al., 2011).

This paper aims to compare different methods to predict tags of Stack Overflow questions. We approach this problem from a document-centered view since information on users is sparse (Wang et al. (2013) show that only 23.1% of the users ask two ore more questions). More formally, we assign tag Y_i from a limited number of tags T to question i based on its title and body consisting of words $X_i = X_{i1}, ..., X_{im}$.

The paper is organized as follows. The next section elaborates on related work that serves as the basis of our study. Section 3 describes the data and elaborates on the necessary pre-processing steps. In section 4 and 5, different tag recommendation methods are applied to the data. There are two approaches to this problem. First we explore classification models (i.e. Naïve Bayes, discriminant analysis, logistic regression and neural networks) where each tag is regarded as a class. Then, we use clustering methods (i.e. k means, mixture models and latent Dirichlet allocation) where tags to a new question are recommended based on the most frequent tags of questions in the same cluster. Finally we conclude in section 5 with some relevant discussion.

2 Related Work

Automatic content tagging has been vastly analyzed as a sub-problem to Web page text classification. Youquan et al. (2011) propose an improved Naïve Bayes algorithm which uses information contained in HTML tags for classification. Song et al. (2011) create two bipartite graphs and predict topics based on leveraging graph partitioning algorithms. Chirita et al. (2007) use cosine similarity and latent semantic analysis in order to measure the similarity of two documents and propose tags of the most similar training documents for the test case. Begelman et al.

(2006) represent tags as nodes of a graph and connect them if the tags were used for the same document. The nodes are clustered and new tags of the same cluster as the already assigned tags are recommended. In the ECML PKDD Discovery Challenge 2009, 24 researchers were asked to submit a paper on tag recommendation. Not a single paper did incorporate Bayesian methods except for latent Dirichlet allocation (Eisterlehner et al., 2009).

Tag recommendation for Stack Overflow has only arisen recent interest. Treude et al. (2011) were the first to analyze Stack Overflow questions by manually labeling 385 questions. Kuo (2011) predicts tags based on a next word prediction algorithm applied on the question body where the tag is the next word. More simply, they predict the likelihood of a tag for a question based on the tag's history of prior use and the strength of association between the tag and the content of words in the post. Hanrahan et al. (2012) cluster questions on Stack Overflow with respect to question complexity, while Nasehi et al. (2012) analyze the quality of Stack Overflow posts based on code snippets. Allamanis and Sutton (2013) present a topic modeling analysis that combines question concepts, types and code. Stanley and Byrne (2013) use a cognitively-inspired Bayesian probabilistic model based on ACT-R's declarative memory retrieval mechanisms (Anderson et al., 2004). Similarly to Barua et al. (2014), Wang et al. (2013) determine the topics of posts based on latent Dirichlet allocation. Finally, motivated by the observation that a lot of posts contain code snippets and that the programming language is often used as a tag, Rekha et al. (2014) combine programming language detection based on a multinomial Naïve Bayes classifier and a support vector machine for prediction.

3 Data

The Stack Overflow dataset was chosen because of its large scale, the rather restricted range of topics and its ease of access. The questions posted on Stack Overflow range across different topics of computer programming and can be posted by anyone with a free account. Each question is then assigned to one up to five tags by the author.

We downloaded the first $50,000$ questions asked on Stack Overflow in 2019 from `https://data.stackexchange.com/stackoverflow` (Atwood, 2009). The dataset contains the tags and the HTML markup of the body and the title of the questions asked in the period from January 1^{st} to February 18^{th} with an overall of $13,948$ different tags.

Since users are allowed to create new tags, the possible number of tags is boundless. For simplicity, we focus on classifying posts with exactly one tag being among the 20 most frequent tags. If we now extract those posts with only one tag and remove duplicates, we are left with $23,458$ observations. This way, we simplified the problem from a multi-label to a multiclass classification problem.

Please refer to Table 1 for a list of the most frequent tags with their occurrences in brackets. We notice that we have highly imbalanced classes. In order to encounter this problem we could, for example, sample 70 observations from each tag class which would leave us with 1.400 instances only. Since our goal is however to recommend the most probable tag, we keep all our observations in accordance to the common literature on tag recommendation.

`python`	(3.838),	`java`	(2.987),	`c#`	(2.458),	`javascript`	(2.357),	`android`	(1.961),
`php`	(1.585),	`r`	(1.337),	`c++`	(1.144),	`angular`	(1.024),	`sql`	(868),
`node.js`	(693),	`reactjs`	(656),	`python-3.x`	(595),	`mysql`	(441),	`ios`	(368),
`swift`	(365),	`css`	(269),	`jquery`	(231),	`html`	(211),	`pandas`	(70)

Table 1: 20 most frequent tags with frequency in brackets.

In a next step, we aim to extract meaningful features from the body and the title of the questions.

First, we discard any code snippets by removing anything enclosed in <code> HTML tags since source code syntax in-

troduces noise into the analysis phase (Thomas, 2012) especially in the case of the short code snippets found on Stack Overflow (Kuhn et al., 2007). Since a majority of tags are programming languages, we also tried to include the code in the analysis. However, since we did not employ any specific programming language detection model, not discarding code snippets led to deteriorated results.

Second, we strip any HTML tags, links and urls from the question body, replace any contractions with expansions, remove all digits and tokenize the texts from question body and title, i.e. split the texts into words. In order to tokenize we use a sentiment-aware tokenizer that does not split frequent tags such as c# and c++.

Third, we found that subsequent lemmatisation of the words, that is transforming inflected words to their dictionary form (i.e. turning 'studies' into 'study') worked better than stemming, that is reducing inflected words to their word stem (i.e. turning 'studies' into 'studi'). After lemmatization, we converted all words to lower-case.

Forth, based on the dictionary provided by the Python NLKT package, we remove common English-language stop words such as "a" and "the", which do not help to create meaningful topics (Manning et al., 2008). We further delete all words that appear in more than 40% of the posts. Please refer to Table 2 for an example post before and after pre-processing.

' If I build an image using the high-level docker-py sdk, I get a BuildError on failure., eg \n \n <pre> <code> try:\n client. images.build(...)\nexcept: \n print("Hey something wrong with image build!")\n</code></pre>\n \n How can I detect when docker-py client.build() fails. '
'if', 'build', 'image', 'use', 'high-level', 'docker-py', 'sdk', 'builderror','failure', 'eg', 'how', 'detect', 'docker-py', 'clientbuild', 'fail'

Table 2: Example post before (top) and after (bottom) pre-processing.

From these lists of words, we now extract our features with a simple bag-of-words approach considering the $1,500$ most frequent words. In simple terms, we count the occurrences of the $1,500$ most frequent words of the whole dataset and save these $1,500$ features for each question. We also experimented with other maximal number of words incorporated in the model, but found that $1,500$ gives the highest validation accuracies for the majority of classification models.

Further, since simply counting the number of words, gives more weighting to long documents and common words which appear in many posts, we use Term Frequency times inverse document frequency (Tf-idf). As a result, the features of each post are divided by the total number of words and the weighting of words that appear in many questions is reduced. In this process, we weight the words contained in the title twice as much as those contained in the question body since the title seems to have higher predictive power in tag classification than the body.

4 Classification

Classification models classify a question i as belonging to a tag t by choosing that t for which $P(Y_i = t|X_i)$, i.e. the probability that question i belongs to tag t given the words X_i, is maximized.

The basic metric for assessing the model performance on text data with highly imbalanced classes, is the micro-averaged F_1 score which is defined as $F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$ where $precision$ is the sum of all correctly classified tags for each tag over the sum of all predicted tags for each tag. $Recall$ is the ratio of the sum of all correctly classified tags over the sum of the number of questions with that tag for all tags. In our case this measure simplifies to the accuracy (number of correctly specified tags/number of all tags).

We save 60% of the observations for training ($14,074$ instances) and 30% ($7,038$ instances) for testing. The remaining 10% ($2,345$ instances) serve as validation set. Since we compare different methods with respect to their performance, we preserve against unintentional

hyperparameter hacking by sweeping over different hyperparameter sets and choose those with the best performance on the validation dataset.

Multinomial Naïve Bayes

Naïve Bayes is a generative model which determines the probability that a question belongs to a tag by assuming that a user's decision to choose a word for his text is not influenced by the other words he has chosen given he has decided on a tag. The probability that a question i is assigned a tag t is then proportional to $P(Y_i = t) \prod_{k=1}^{m} P(X_{ik}|Y_i = t)$.

If we define the probabilities as the respective relative frequencies, our model achieves an accuracy of only 66.03% on the test data. One flaw of the model is that questions which contain a word that was not used in any other question of that tag cannot be classified as that tag. In order to overcome this obstacle, we add a smoothing parameter to the probability estimates such that the test accuracy increases to 73.97%. In comparison, we would only achieve a test accuracy of 5% if we would randomly guess the class.

Naïve Bayes is thus a simple approach which already achieves high test accuracies. However, this classifier has two flaws: (1) when a new question is classified, information on the frequency of the words contained in the question is discarded and (2) it is reasonable that the choice of words is not independent given a tag. For a question with tag `python` the probability that the word `dataframe` is included, likely to increase if the word `pandas`, namely the basic package for working with data frames, appears.

Discriminant Analysis

In discriminant analysis, we assume that the user picks words according to a multivariate normal distribution whose parameters (mean vector and covariance matrix) depend on the tag that is assigned. The parameters for these distributions are chosen such that they maximize the likelihood, or more simply the density functions of the normal distributions at the observed word vectors. The probability of a tag given the word vectors can then be calculated based on Bayes' Theorem. Compared to before we now allow the features to be correlated.

If we assume that the covariance matrix is the same across all tag classes, this method is called linear discriminant analysis. It gives a test accuracy of 73.97% and a train accuracy of 83.14%. We notice that the test accuracy has not increased compared to the results obrtainded previously. Apparently there is no benefit in allowing the features to be dependent. We can support this assumption by the observation that the average absolute correlation of the features is only 0.0072. Another difference to Naïve Bayes is that X is now modelled by a continuous distribution (compared to binary distribution). Although the Tf-idf score of the words is thus also considered now, our accuracy does not increase. Therefore, Bernoulli distributions are a reasonable approximation of these features. We can support this statement by considering the histogram of the occurrences of the most frequent word in the training dataset and the corresponding normal distribution as illustrated in Figure 1. We see that the distribution of this feature (and also of other features) is highly right-skewed. It is further limited to the interval $[0, 1]$ since we normalized the features. A more appropriate approximation would thus be the Pareto distribution.

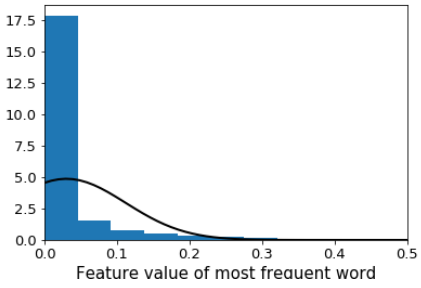

Figure 1: Histogram of the Tf-idf scores of the most frequent word and density of fitted normal distribution

We speak of quadratic discriminant analysis, if the covariance matrix can vary across classes. This method only yields a test accuracy of 25.81% and a train accuracy of 71.86%. First, we note that the train accuracy decreased since 19 additional covariance matrices of size $1.500 \times$

1.500 had to be computed. Since the parameter space thus increased as well, the model overfits the training data and yields low test accuracies. Therefore it is reasonable to assume that the covariance matrix does not vary across classes. As a result, we also assert that it is more suitable to separate the feature vectors of questions with different tags with a linear instead of a quadratic hyperplane.

Since discriminant analysis methods (and also Naïve Bayes) specify a distribution for the features, they are generative. For example, we could generate a list of words for a question with tag `python` by drawing from the conditional multivariate distribution of the word vector. An alternative discriminative model is logistic regression where generation is unattainable but as a trade-off the number of parameters is shortened and we make fewer assumptions.

Logistic Regression

Multinomial logistic regression specifies the probability of a question i to belong to tag class $t \in \{1, ..., 20\}$ as

$$P(Y_i = t|X_i) = \frac{e^{\beta_t \cdot X_i}}{\sum_{k=1}^{20} e^{\beta_k \cdot X_i}}.$$

Simply put, we define a regression function for each class and normalize this number to the interval $[0, 1]$ by taking the exponential of the output and dividing by the exponential functions of all regressions. The higher the value β_{tk}, the more probable it is that question i has tag t if the word X_k is included.

We maximize the likelihood of this model and get a train accuracy of 100%, while the test accuracy is only 55.34%. The basic logistic regression approach is clearly overfitting. Alternatively, we can include an $L2$ or $L1$ regularization term ($|\beta|^2$ or $|\beta|$) which penalizes high values of betas in the estimation process. The former is the same as assuming the beta coefficients are distributed according to a normal distribution before seeing the data (Hoerl and Kennard, 1970), while the latter assumes that the beta coefficients follow a Laplace distribution before seeing the data (Tibshirani, 1996). These two alternatives have proven robust in the case of text categorization (Genkin et al., 2007). The intuition is that we tell the model that the parameters of the model are likely to be near zero such that the actual estimates are shrinked towards zero. This indeed reduces overfitting since our train accuracies decrease to 79.01% when including an $L1$ penalty resp. 80.97% when including $L2$ penalty. Since the former overfits less, its test accuracy is also higher (75.89% compared to 73.57%).

Please refer to Figure 2 where we plotted the boxplots for the beta coefficients. We see that the range of values is the largest for logistic regression without penalty term. Coefficients range between -279 and 371 which results from the fact that this model overfits. Including the $L1$ penalty term shrinks the coefficients' range to $[-4.31, 13.59]$, while $L2$ leads to a range of $[-10.35, 33.27]$ which lets us suggest that in our case the Laplace prior puts more probability around zero than the Gaussian prior.

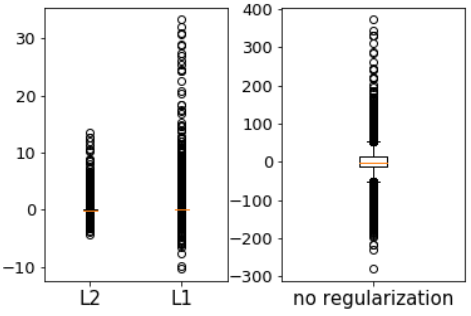

Figure 2: Boxplots for the distributions of the coefficients for logistic regression with no, $L1$ and $L2$ penalty

We also tried to implement a Bayesian logistic regression with other prior distributions where the posterior was approximated by Monte Carlo sampling using the package `PYMC3`. However, we ran out of memory (on a single machine with 16 GB RAM) such that the model can only be trained if a small fraction of the features (i.e. the 100 most frequent words) are included. In this case the test accuracy however drops to 52.53% and the estimation of the model parameters takes over 662 seconds compared to the 3.78 seconds needed for $L1$ regularized logistic regression.

Neural Networks

Last but not least, we implement a perceptron with one hidden layer with 256

neurons. It can be expressed similarly to a two-stage regression where the second stage $Y_1 = b_1 + X_1 W_1$ is applied on the 256-dimensional output of the first stage $Y_0 = b_0 + X_0 W_0$. All in all we have

$$P(Y) = \sigma(b_1 + (b_0 + X W_0) W_1),$$

where the softmax function $\sigma(z_i) = \frac{e^{z_i}}{\sum_{i=0}^{20} e^{z_i}}$ normalizes the output of the second stage to the interval $[0, 1]$. The coefficients W_0 and W_1 are called weights, while we refer to b_0 and b_1 as bias vectors.

Such a model achieves a train accuracy of 1 since it overfits the data. As before we use an $L2$ regularization term in the estimation process which penalizes large weights. Furthermore we include a dropout layer (Srivastava et al., 2014) after the first layer (which is the first stage regression). This layer drops some of the output of the first layer such that not all of the information is carried over to the second layer. We only achieve a test accuracy of 75.23% which is smaller than the best accuracy achieved with logistic regression.

As an alternative regularization method we can implement Bayesian neural networks where we specify the uncertainty about the predictors of the weights and biases by prior distributions (Neal, 2012). Priors such as the multivariate standard normal distribution, as implemented here, shrink the predictors towards zero. We approximate the posterior distribution with the Flipout Monte Carlo estimator (Wen et al., 2018) and fit a model on the test data using the maximal values of the posterior distributions. This perceptron achieves the highest test accuracy of 79.7% among all our models.

The main difference between a neural network with one hidden layer and two models is the first stage of the regression. Since the Bayesian neural network predicts the tags of the test data more accurately than the Bayesian logistic regression, we assume that the additional complexity of the neural network does capture the process of tag choice. However since the plain neural network performed worse, $L2$ regularization does not shrink the weights enough.

Another drawback of neural networks is that the weights and bias vectors are hard to interpret and the Bayesian model takes 1.613 seconds to train while Naïve Bayes achieves a similar test accuracy of 73.97% in 0.17 seconds. Moreover, Naïve Bayes outputs probabilities for all words given a tag and all tags given word features which are useful for the interpretations of tag use.

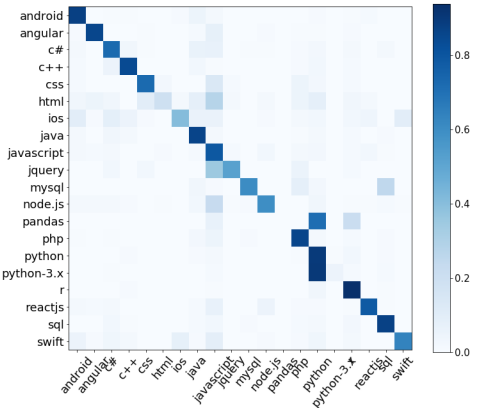

Figure 3: Confusion matrix for Bayesian neural network (left) with color bar (right) specifying share of questions with predicted label (as specified by column labels) for each true tag (as specified by the rows).

Please refer to Figure 3 for a confusion matrix of the test results. The darkness of a cell is proportional to the share of questions with true label (as specified in the respective row) and predicted label (as specified by the respective column label) from all questions with true label as specified in the rows. Thus it would be desirable that the cells in the diagonals are the only ones with coloring. However, we see that the majority of questions with tags `pandas` and `python-3.x` have been wrongly classified with the tag `python`. This results from the thematic similarity of the tags (`python-3.x` being a python version, while `pandas` is a python package) and from the imbalance in classes (`python` has 3.838 instances while the other two tags have only 665 instances together). Further some questions relating to web applications (with tags `node.js`, `jquery`, `html` and `css`) have been wrongly classified as being tagged with `javascript`. Notice that these tag groups are clustered together e.g. when we use latent Dirichlet allocation. The confusion matrices of other methods such as linear discriminant anal-

ysis have lower values in the diagonal and higher values in the column corresponding to questions with true label `python` and `javascript`. Apparently, those methods suffer even more from the imbalanced classes. Thus, techniques combating imbalanced data such as resampling could boost their performance.

5 Clustering

In the following we assume that each question belongs to a cluster Z which is unobserved but influences the word choice X. We can think of Z as a topic. If we know to which cluster a question belongs, we can recommend the user tags that frequently appear in the questions of that cluster.

In order to visualize our results we reduce the dimension of the features from 1.500 to 2 using principal component analysis where the axes are chosen such that the variance of the data in these two dimensions is maximized. We also experiment with axes chosen according to linear discriminant analysis such that also the separation between the tag classes is maximized. This latter dimensionality reduction method takes also information on the classes into account which are not considered in clustering methods but help us in our aim of accurate tag recommendation. However, we found linear discriminant analysis was (due to the additional information it takes into account) not suitable for visualizing the clusters found by our models.

In Figure 4 we see the data using the first two principal components for representation. We find that the first principal component captures the variance in the word vectors belonging to tags web and mobile applications, while the second principal component captures the variance in word choice of questions related to data analysis.

K means

An intuitive (not probabilistic) way to cluster data is to minimize the distance between the feature vectors of instances within a cluster. K means is an iterative algorithm where all questions are assigned a random cluster in the initial step before they are re-assigned to that cluster that

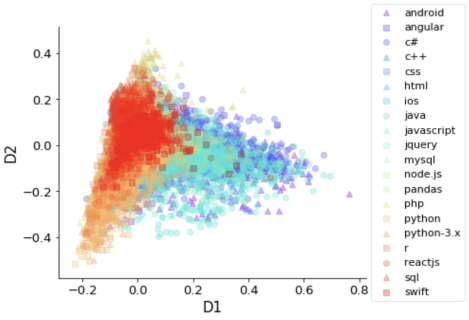

Figure 4: Data reduced to two dimensions. $D1$ resp. $D2$ denotes the first resp. second axis computed by the principal component analysis.

minimizes the squared euclidean distance of the word vector and the respective centroid. The centroid of a cluster is the mean over the word vectors that belong to the cluster.

This model requires the number of clusters K to be predetermined. We use the Bayesian Information Criterion $BIC = K \cdot log(n) - 2 \cdot ln(P(X|K))$, where n denotes the number of questions, to choose that parameter. The BIC contains on the one hand the likelihood $P(X|K)$, that is the probability to observe the questions that were posted on Stack Overflow given a K means clustering. Since an observation is better explained by a cluster if it is the only instance of that group, the likelihood increases with K. On the other hand, it penalizes large values of K by the second term.

Since K means largely depends on the initial assignment to the clusters, this algorithm has to be repeated several times. We achieve the lowest BIC score for $K = 20$ among all $K \in \{1, ..., 20\}$. Thus even higher values for K should be chosen if the aim is to accurately find all clusters. This can be explained by the high dimensionality of the data. Since the interpretability of the results however decreases with the number of clusters, we will use 5 clusters in the following. The centroids of these clusters can be used to find the words that are on average most frequent in the questions that were grouped together. The results are displayed in Table 3 on the next page.

We refer to the clusters by the row number they are presented in in the table.

Top 5 words	Top 5 tags
file, line, import, python, error	python, java, c#, php, python-3.x
code, error, work, import, like	python, java, android, c#, javascript
public, class, string, void, new	java, c#, android, c++, php
table, select, query, column, row	sql, mysql, r, python, c#
gt, array, var, function, return	javascript, php, reactjs, c#, angular

Table 3: 5 most frequent words and tags per cluster chosen by K means

We see that the first cluster groups questions related to `python` (word `python`, tag `python-3.x`), while the fourth cluster relates to data analysis (words `table`, `query`, `table` and tags `sql`, `mysql`, `r`). The last cluster refers to web application programming (tags `javascript`, `php`,`reactjs`, `angualr`) and cluster 3 contains questions on classes in object oriented languages (words `public`, `class`).

As can been seen in Figure 5, the clusters can be even visually separated when the data is projected to the two-dimensional space. This presentation however benefits from the fact that the data is displayed along the axes that maximize the variance of the data, while K means minimizes the within cluster variance.

One major drawback of this clustering model in our setting however is that K means tends to assign a similar area of the feature space (excluding the infinite empty space) to the clusters. Smaller topical group thus also include questions of larger groups.

Mixture Models

If we assume that the user chooses a topic Z among K alternatives according to a multinoulli distributed, and that the process f underlying the word choice X depends on this topic Z, we can express the question data as mixture.

Gaussian mixture models assume that

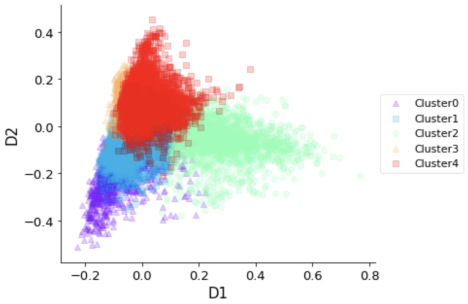

Figure 5: Clusters found by the K means model reduced to two dimensions

the we model the features as multivariate normal distributions. We find that a diagonal covariance matrix which assumes that the words within clusters are chosen independently from each other and that the variance can differ across clusters achieves the lowest BIC (even lower than that of K means). This observation is in accordance to our findings on the superiority of Naïve Bayes over linear discriminant analysis for classification.

In contrast to K means, we can now compute the probability of an observation belonging to a cluster and can thus also assess the uncertainty related to the clustering. Again we can find the most common terms within each cluster as tabulated in Table 4 on the following page. Compared to K means, we can also display probabilities of the words.

We see that the first cluster groups questions related to data processing (words `table`, `date`, tags `r`, `python`, `sql`) and the fifth cluster questions related to JavaScript (tags `javascript`, `angular` (platform for building web applications), `reactjs` (a JavaScript library)). According to Figure 6 on the next page the questions within a cluster are now more dispersed since we no longer minimize the squared euclidean distance between features of the same cluster. Further we now also have imbalanced clusters which mirrors the thematic range of the topics more accurately.

We now take a fully Bayesian approach and assume that the the variance vectors of the word distribution follow a Wishart distribution and the probabilities of the topics are Dirichlet distributed. However, BIC

Top 3 words	Top 5 tags
`table` (0.15), `date` (0.1), `select` (0.0809)	`python`, `r`, `sql`, `java`, `c#`
`file` (0.31), `line` (0.08), `python` (0.0353)	`python`, `java`, `c#`, `php`, `python-3.x`
`function` (0.03), `return` (0.03), `code` (0.0244)	`java`, `c#`, `android`, `c++`, `php`
`public` (0.25), `void` (0.11), `string` (0.0867)	`python`, `android`, `java`, `c#`, `javascript`
`gt` (0.22), `const` (0.12), `return` (0.0485)	`javascript`, `angular`, `php`, `reactjs`, `python`

Table 4: 3 most probable words (probabilities conditional on cluster in brackets) and 5 most common tags per cluster determined by a Gaussian mixture model assuming 5 clusters and a diagonal covariance matrix.

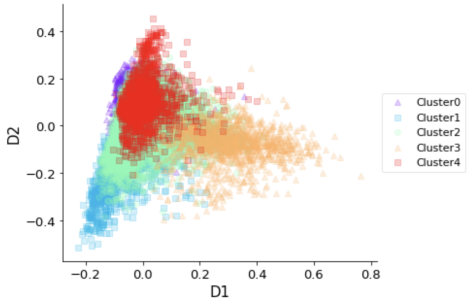

Figure 6: Clusters found by the Gaussian mixture model displayed in two dimensions

decreased and the clustering was no longer visually tractable.

As we have seen before, assuming a Gaussian distribution for X did not increase the performance of our classification models. For this reason, we will now assume that the word choice underlies a Bernoulli distribution with the probability parameter underlying a Beta prior. The probabilities of the multinoulli distribution of Z are given a Dirichlet prior. We estimate the model using variational Bayesian inference. After at least 13 iterations and a minimum of 336 seconds, the model converges. A benefit of this model is that it implicitly penalizes the number of clusters such that the model assigns questions to only three clusters. Nonetheless, this penalty is too large such that the model is too general for any suitable tag recommendation. Moreover, the algorithm is highly susceptible to local minima and achieved a maximal training time of $1,882$ seconds in five runs (compared to approximately 21 seconds needed for the other mixture models).

Latent Dirichlet Allocation

One drawback of mixture models is that each question can only belong to one topic. If we assume that a user can also choose more than one topic, before he starts writing his question, we draw Z (and the words X) from a multinomial distribution and model the clusters with an admixture model.

The intuition of latent Dirichlet allocation (Blei et al., 2003) is to set the priors of the parameters for the topic and word choice equal to Dirichlet distributions. This lets the model become an infinite admixture which covers a wider range of distributions than finite mixtures. The posterior of the model can then be estimated using collapsed Gibbs sampling by iterating multiple times over the posts and determining $P(Z_i|Z_{-i})$ until convergence where Z_{-i} is the set of all assigned tags excluding topic Z_i (Griffiths, 2002).

Similar to Krestel and Fankhauser (2009), we analyze the distribution of tags in the clusters. For illustration purposes we follow Wang et al. (2013) and let our model identify 5 topics. We noticed that including a number of $5,000$ in the bag-of-words approach gave results with the highest interpretability.

Please refer to Table 5 for the five words with the highest likelihood for each topic and the three most frequent tags per cluster. First we note that the likelihoods of the words are close together (range from 0.0005 to 0.0003). A wider range of likelihoods can be achieved by a smaller maximal number of words.

Questions grouped into topic 5 seem to be related to reading in data and pre-processing it because of high likelihood of words like `date`, `data` and `file`. Possible tags might thus be `sql` (programming language for managing data) or

pandas (Python library for handling especially DataFrames) which is indeed true.

Top 5 words	Top 3 tags
array, object, image, class, python	pandas, python, python-3.x
component, file, gt, angular, data	css, reactjs, angular
function, file, data, call, query	node.js, jquery, php
list, file app, object	ios, c++, swift
file, string, table, data, date	r, sql, pandas

Table 5: Five most likely words and three most frequent tags for each topic

Similarly, we notice that the topic 1 is strongly related to Python because of high likelihoods of words like python and array.

While topic 4 is apparently related to app programming (word app and tags swift (programming language of Apple) or ios), topic 2 and 3 surround issues concerning web applications (words angular, gt (selector in JavaScript)).

Latent Dirichlet allocation is with a fitting time of 22 seconds already considerably faster than K means (fitting time of nearly 390 seconds), although the feature space was more than three times larger than in the latter model. Wu et al. (2016) experiment with different parameter estimation methods and find that the CVB0 learning algorithm (Asuncion et al., 2009) converges faster than Gibbs Sampling in a similar setting. For only $1,500$ features however, latent Dirichlet allocation returns poor results with respect to interpretability and BIC.

6 Discussion

In this paper we examined the performance of both frequentist as well as Bayesian approaches to tag classification of Stack Overflow posts. The main motivation was to assess the performance of different methods and analyze their results.

When classifying the data, we found that frequentist approaches overfit the data, even when regularization terms were added. Including Bayesian priors is indispensable considering the large feature space which makes subset selection methods computationally unfeasible. As a result Bayesian neural networks achieved the highest test accuracy of 79.7%.

Nonetheless, accuracies of the models are close together. For better comparison it would be preferable to compare the predicted class probabilities using measures such as the log score.

Clustering the questions allowed us to recommend tags that were common to a cluster a question belonged to. We find that the high-dimensionality of the data entails that only a large number of clusters minimizes the BIC. In an effort to ensure interpretability, we implemented all models considering only 5 clusters.

The benefit of Bayesian approaches in the framework of text clustering and classification, is the possibility to interpret the probabilities of the word features in each cluster or the probabilities of tags given word features.

Our analysis is limited by the fact that we only consider the top 20 tags. Barua et al. (2014) for example find out that posts related to the iPhone technology are tagged in various ways (iphone-sdk- 3.2, iphone-3gs, iphone-sdk-documentation). It would be interesting to see whether our clustering approaches would cluster posts with these tags into the same cluster.

We started by tokenizing the question texts in order to obtain numerical features. One drawback of the bag-of-words approach however is that we discard any contextual information such as grammar or word order. However, we found that considering n-grams, that is a sequence of n words, instead of single words would deteriorate our results since for example models such as Naïve Bayes assume that words are chosen independently from each other. Incorporating dense features, such as continuous word representations computed by Word2Vec (Mikolov et al., 2013), might benefit the results of the analysis. Further, we could have incorporated some meta fea-

tures from the raw text considering especially the discarded code fragments.

We identify several directions for future work. While in our paper the tag `python` is obviously the most frequent, not so recent research identifies other programming languages as most frequent tags (e.g. c in the dataset of Stanley and Byrne (2013) from 2011). Apparently, temporal trends in tagging exist. Therefore it is important to find automatic tagging model that be continuously updated.

Further, latent Dirichlet allocation has proven a valuable clustering resp. topic model. Therefore it would be interesting to implement a supervised version (Mcauliffe and Blei, 2008) where the information contained in tags could improve current tag recommendation baselines.

Last but not least, tags are subjective and user-specific. An approach which combines user-based and document-based approaches (Chirita et al., 2007) could therefore perform superior.

References

Allamanis, M. and Sutton, C. (2013). Why, when, and what: analyzing Stack Overflow questions by topic, type, and code. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 53–56. IEEE.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., and Qin, Y. (2004). An integrated theory of the mind. *Psychological review*, 111(4):1036.

Asuncion, A., Welling, M., Smyth, P., and Teh, Y. W. (2009). On smoothing and inference for topic models. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 27–34. AUAI Press.

Atwood, J. (2009). Stack Overflow creative commons data dump.

Barua, A., Thomas, S. W., and Hassan, A. E. (2014). What are developers talking about? an analysis of topics and trends in Stack Overflow. *Empirical Software Engineering*, 19(3):619–654.

Begelman, G., Keller, P., Smadja, F., et al. (2006). Automated tag clustering: Improving search and exploration in the tag space. In *Collaborative web tagging workshop at WWW2006, Edinburgh, Scotland*, pages 15–33.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Chirita, P.-A., Costache, S., Nejdl, W., and Handschuh, S. (2007). P-tag: large scale automatic generation of personalized annotation tags for the web. In *Proceedings of the 16th international conference on World Wide Web*, pages 845–854. ACM.

Diakopoulos, N. A. and Shamma, D. A. (2010). Characterizing debate performance via aggregated twitter sentiment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1195–1198. ACM.

Eisterlehner, F., Hotho, A., and Jäschke, R. (2009). Ecml pkdd discovery challenge 2009 (dc09). *ceur-ws. org*, 497.

Genkin, A., Lewis, D. D., and Madigan, D. (2007). Large-scale bayesian logistic regression for text categorization. *Technometrics*, 49(3):291–304.

Griffiths, T. (2002). Gibbs sampling in the generative model of latent dirichlet allocation.

Hanrahan, B. V., Convertino, G., and Nelson, L. (2012). Modeling problem difficulty and expertise in StackOverflow. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work Companion*, pages 91–94. ACM.

Hoerl, A. E. and Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67.

Krestel, R. and Fankhauser, P. (2009). Tag recommendation using probabilistic topic models. *ECML PKDD Discovery Challenge*, 2009:131.

Kuhn, A., Ducasse, S., and Gírba, T. (2007). Semantic clustering: Identifying topics in source code. *Information and Software Technology*, 49(3):230–243.

Kuo, D. (2011). On word prediction methods. *Technical report, Technical report, EECS Department*.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). Text classification and naive bayes. *Introduction to information retrieval*, 1(6).

Mcauliffe, J. D. and Blei, D. M. (2008). Supervised topic models. In *Advances in neural information processing systems*, pages 121–128.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Nasehi, S. M., Sillito, J., Maurer, F., and Burns, C. (2012). What makes a good

code example?: A study of programming q&a in stackoverflow. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 25–34. IEEE.

Neal, R. M. (2012). *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media.

Rekha, V. S., Divya, N., and Bagavathi, P. S. (2014). A hybrid auto-tagging system for StackOverflow forum questions. In *Proceedings of the 2014 International Conference on Interdisciplinary Advances in Applied Computing*, page 56. ACM.

Song, Y., Zhang, L., and Giles, C. L. (2011). Automatic tag recommendation algorithms for social recommender systems. *ACM Transactions on the Web (TWEB)*, 5(1):4.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Stanley, C. and Byrne, M. D. (2013). Predicting tags for Stack Overflow posts. In *Proceedings of ICCM*, volume 2013.

Thomas, S. W. (2012). Mining software repositories with topic models. *School of Computing. Queen's University, Canada*.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288.

Treude, C., Barzilay, O., and Storey, M.-A. (2011). How do programmers ask and answer questions on the web?: Nier track. In *2011 33rd International Conference on Software Engineering (ICSE)*, pages 804–807. IEEE.

Tsur, O. and Rappoport, A. (2012). What's in a hashtag?: content based prediction of the spread of ideas in microblogging communities. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 643–652. ACM.

Wang, J., Clements, M., Yang, J., de Vries, A. P., and Reinders, M. J. (2010). Personalization of tagging systems. *Information Processing & Management*, 46(1):58–70.

Wang, S., Lo, D., and Jiang, L. (2013). An empirical study on developer interactions in StackOverflow. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1019–1024. ACM.

Wen, Y., Vicol, P., Ba, J., Tran, D., and Grosse, R. (2018). Flipout: Efficient pseudo-independent weight perturbations on mini-batches. *arXiv preprint arXiv:1803.04386*.

Wu, Y., Yao, Y., Xu, F., Tong, H., and Lu, J. (2016). Tag2word: Using tags to generate words for content based tag recommendation. In *Proceedings of the 25th ACM international on conference on information and knowledge management*, pages 2287–2292. ACM.

Youquan, H., Jianfang, X., and Cheng, X. (2011). An improved Naive Bayesian algorithm for Web page text classification. In *2011 Eighth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, volume 3, pages 1765–1768. IEEE.