# Programming Assignment: Instructions

Help Center

Submit Now

## Assignment Overview

In this assignment, you are going to have hands-on experience with several important text mining techniques that you have seen in the lectures. The assignment is composed of the following four tasks:

- Task 1 (Part of Speech Tagging) where you will apply POS tagging on a document
- Task 2 (Word Association Mining) in which you will implement mutual information and extract the top syntagmatically related words from the corpus
- Task 3 (Topic Modeling) where you will code a part of PLSA and experiment with its parameters
- Task 4 (Text Mining Competition) where you are going to use Yelp restaurant reviews to predict whether each restaurant is clean or not as judged by health inspectors

Working on these tasks will allow you to more accurately understand some important concepts such as mutual information and algorithms such as EM algorithm. To enable you to get the maximum benefit of learning from this programming assignment while minimizing your effort, we have provided a partial implementation of all the algorithms so that you only need to spend minimum effort on coding once you have completely understood the relevant concepts and algorithms. We hope this also make it more feasible for those of you who are not very familiar with coding in C++ to work on the programming assignment, thus allowing as many people to benefit from the programming assignment as possible.

If you have questions about the programming assignment, use the Programming Assignments Forum. This is a great place to ask questions and also to help your fellow classmates.

## Downloading the Assignment

In what follows, we assume that you have already installed MeTA and the directory **meta** is located in ~/Desktop/. If you haven't installed MeTA yet, go to the Programming Assignment Overview page and follow the installation instructions. If you choose to install MeTA in a different location, you should change ~/Desktop/ to the parent directory of the directory **meta**. If you are using the provided virtual image, you do not have to make any changes.

1. Download Assignment.tar.gz and extract it into the parent directory of **meta**. If **meta** is in ~/Desktop/, then you should extract the assignment in ~/Desktop/.
2. Open a terminal. In the provided VM (and Ubuntu in general), you can open a terminal using the Ctrl+Alt+T keyboard shortcut.
3. Change the directory to Assignment and run the bash script **Setup.sh**. This can be done using the following two commands:

```
cd ~/Desktop/Assignment/
./Setup.sh
```

   **Setup.sh** will copy the assignment files and the datasets to MeTA.

4. Recompile MeTA:

```
cd ~/Desktop/meta/build/
make
```

   After running the make command, you might get several warnings about unused variables from the compiler; you can safely ignore them.

## MeTA Overview

Before starting with the assignment, you are highly encouraged to read MeTA's System Overview to gain a high-level understanding of how MeTA operates. Throughout the assignment, if you want to know more about a certain class or function, you can use the search toolbar in MeTA's Documentation, which provides a brief explanation of the different modules. Below we will quickly go over some concepts that will help you understand how MeTA works.

### Inverted and Forward Indices

Two important data structures frequently used in text mining and retrieval applications are the **inverted index** and the **forward index**. The inverted index is a mapping that stores terms as keys and document IDs as values, thus allowing efficient lookup of all the documents that contain a specific term. The forward index, on the contrary, has document IDs as keys and words as values, which allows us to efficiently extract the terms in a specific document. MeTA stores all datasets in form of an inverted index or a forward index, depending on the requirements of the application. In the assignment's code, we will use both indices to process the datasets. You can find out more information on these indices in this Wikipedia article. See Disk_Index Class Reference for the API of the indices in MeTA (you can click on the inverted or forward index in the inheritance diagram to access their specific APIs).

### Configuring MeTA

Most of the parameters in MeTA can be changed using a single file called **config.toml** located in **meta/build/**. Open the file and have a look at its content. For example the following snippet:

```
corpus-type = "line-corpus"
dataset = "yelp"
forward-index = "yelp-fwd"
inverted-index = "yelp-inv"
```

tells MeTA that we are going to use a dataset called “yelp”. The corpus type is set to “line corpus”, meaning that it is stored in one file where each document occupies a single line. “yelp-fwd” and “yelp-inv” are the names of forward and inverted indices to be created. We are going to use the Yelp dataset for word association mining later on.

Another important snippet is:

```
[[analyzers]]
method = "ngram-word"
ngram = 1
        [[analyzers.filter]]
        type = "whitespace-tokenizer"
        [[analyzers.filter]]
        type = "lowercase"
        [[analyzers.filter]]
        type = "alpha"
        [[analyzers.filter]]
        type = "length"
        min = 2
        max = 35
        [[analyzers.filter]]
        type = "list"
        file = "../data/lemur-stopwords.txt"
        [[analyzers.filter]]
        type = "porter2-stemmer"
```

This tells MeTA how to process the text before indexing the documents. “ngram=1” configures MeTA to use unigrams (single words). Each “[[analyzers.filter]]” tag defines a text filter that applies a special function on the text. These filters are being “chained” together; text will first be processed by a whitespace tokenizer which separates words based on white spaces, then all the tokenized words will be converted to lowercase. This is followed by a couple of filters that end up with stopword removal and stemming. These filters can be usually changed depending on the application. For more information on how to use and configure the filters in MeTA see MeTA's Analyzers and Filters documentation.

Note: Do not change any settings in **config.toml** when doing Tasks 1, 2, and 3 unless told to do so. In task 4 (competition), you are free to explore and change any parameters you want.

## Task 1 (Warm Up): Part-of-Speech Tagging (10%)

MeTA supports several shallow text analysis techniques such as stemming and frequency analysis in addition to natural language processing techniques like part-of-speech tagging and parsing that allow a deeper understanding of text. These methods can be applied to individual text documents as well as to the entire corpus without doing any coding.

Part-of-Speech (POS) tagging refers to the process of assigning part-of-speech tags, such as “noun” or “verb,” to words in documents. MeTA has a built-in part-of-speech tagger that has been already trained on an English corpus.

In this task, you are going to run POS tagging on a text document. Go to **meta/build/Assignment/** and have a look at the content of **doc.txt**. This document contains the description found on the syllabus of this course. To perform POS tagging on **doc.txt**, execute:

```
cd ~/Desktop/meta/build/
./profile config.toml Assignment/doc.txt --pos
```

The second command runs a built-in program in MeTA called **profile** and passes to it the configuration file **config.toml** and **doc.txt** as arguments. It is very common to pass **config.toml** as an argument to applications in MeTA.

Now in **meta/build/Assignment/** you should find a new file called **doc.pos-tagged.txt** which was output by **profile**. Open this file and examine its content. You should see how the POS tags are attached to each word after the underscore. Note that the tags are abbreviated. For example, the first word “this” has been assigned the tag “DT” which refers to a determiner, whereas the second word got “NN” which refers to a noun. For a description of the commonly used POS tags see Penn Treebank List of POS Tags.

Submit your output file by executing:

```
cd ~/Desktop/meta/build/Assignment/
python submit.py
```

The submission script will ask for your email and **one-time password** (Note: this is different from the password you use to sign in to Coursera). Enter the email associated with your Coursera account. You can find your one-time password in the Assignments Page. When prompted about the task number to submit, enter 1. If you experience problems with the submission script, upload **doc.pos-tagged.txt** to the Assignments Page.

## Task 2: Word Association Mining (30%)

In this task, you will mine the words with the strongest syntagmatic relations in a corpus of 10,000 restaurant reviews extracted from the Yelp Academic Dataset. The corpus is located in **meta/data/yelp/**. If you want to have a quick look at the dataset, go to its directory and open **yelp.dat**, which contains each review on a separate line. We have already configured MeTA to use the Yelp dataset through **meta/build/config.toml**, as discussed previously.

Words with strong syntagmatic relations usually tend to co-occur frequently together while having relatively low individual occurrence. There are many applications where syntagmatic relation mining is important. For example, in retrieval, words that have strong syntagmatic relations with the original query words can be used to expand the query in order to enhance retrieval results. Another application is opinion summarization; for example, we can extract the top K syntagmatically related words to “iPhone 6” from a corpus of customer reviews in order to summarize the users' feedback.

The techniques used to mine word associations can be generally classified into two categories. The first is hypothesis testing, where statistical tests are used to determine if the co-occurrence of two words happened by chance or due to an actual correlation. The second category is information-theoretic and is based on measures such as mutual information, which was discussed in the lectures. For a survey of the different methods used for word collocation mining see this chapter on collocation mining.

We are going to focus on implementing Mutual Information as the association measure to mine the top K syntagmatically related words in the Yelp corpus. Let $X_{w_a}$ and $X_{w_b}$ be two binary random variables that denote whether the words $w_a$ and $w_b$ have appeared in a certain document. The mutual information, $I(X_{w_a};X_{w_b})$, measures the dependence between the two random variables associated with the two words. More specifically, it measures the reduction in the entropy (randomness) of one of the words when we have knowledge about whether the second word is present in the document or not. The mutual information between the two random variables is defined as follows:

$$I(X_{w_a};X_{w_b}) = \sum_{u\in\{0,1\}}\sum_{v\in\{0,1\}} P(X_{w_a}=u, X_{w_b}=v)\log\frac{P(X_{w_a}=u, X_{w_b}=v)}{P(X_{w_a}=u)P(X_{w_b}=v)} \tag{1}$$

The probabilities in (1) can be estimated based on the frequency of the words in the corpus while using additive smoothing to avoid zero probability assignments. The main probability estimates can be written as:

$$P(X_{w_a}=1) = \frac{\text{count}(w_a)+0.5}{N+1} \quad \text{and} \quad P(X_{w_a}=0) = 1 - P(X_{w_a}=1) \tag{2}$$
$$P(X_{w_b}=1) = \frac{\text{count}(w_b)+0.5}{N+1} \quad \text{and} \quad P(X_{w_b}=0) = 1 - P(X_{w_b}=1) \tag{3}$$
$$P(X_{w_a}=1, X_{w_b}=1) = \frac{\text{count}(w_a,w_b)+0.25}{N+1} \tag{4}$$

where $\text{count}(w)$ is the number of documents the word $w$ appears in (i.e., the document frequency of $w$) and $\text{count}(w_a,w_b)$ is the number of documents where both $w_a$ and $w_b$ occur. By applying the law of total probabilities, we can write the rest of the probability estimates as follows:

$$P(X_{w_a}=0, X_{w_b}=1) = P(X_{w_b}=1) - P(X_{w_a}=1, X_{w_b}=1) \tag{5}$$
$$P(X_{w_a}=1, X_{w_b}=0) = P(X_{w_a}=1) - P(X_{w_a}=1, X_{w_b}=1) \tag{6}$$
$$P(X_{w_a}=0, X_{w_b}=0) = P(X_{w_a}=0) - P(X_{w_a}=0, X_{w_b}=1) \tag{7}$$

We have created a special program for this task called **association.cpp** located in **meta/src/tools/**. Before proceeding to implement mutual information, we will first use **association** to print the top 50 words with the highest co-occurrence in the corpus, i.e., **association** will rank each two words $w_a$ and $w_b$ based on $\text{count}(w_a, w_b)$ and will then print the top 50 word pairs. To run **association**, in the terminal execute:

```
cd ~/Desktop/meta/build/
./association config.toml --words 50
```

You should see the top 50 word pairs printed along with their numbers of occurrence (the program might take some time to produce the output). As you have noticed, many of the top terms are very common and not interesting. This can be explained by the fact that a high number of co-occurrences alone does not imply high correlation; for words to be highly correlated, they must co-occur frequently and have relatively low individual occurrences.

To get more informative word pairs, we will use mutual information as the association measure. Go to **meta/src/tools** and open **association.cpp**. **association.cpp** contains an incomplete implementation of mutual information. Your task is to complete the code of the function **MutualInformation**, which is located at the top of the file. Equations (2), (3), and (4) have already been implemented. Specifically, you should enter the correct formulas for equations (5), (6), and (7) and implement the mutual information formula, i.e., equation (1), in the return statement. Make sure to read and strictly follow the comments in the code and to use log base 2 for implementing the logarithm (you can write $\log_2 x$ as log2(x) in C++). Also, do not forget to save the file after you finish coding.

After you finish coding the mutual information function, you should recompile MeTA by executing

```
cd ~/Desktop/meta/build/
make
```

If the compilation process terminates with an error, make sure that your syntax is correct. Before proceeding with the exercise, submit your code and make sure that you have received full credit. You can do so by executing:

```
cd ~/Desktop/meta/build/
./association config.toml --submission
cd Assignment
python submit.py
```

Note that "./association config.toml --submission" might take up to a minute to finish executing. When prompted for the task number, enter 2. Your grade should instantly appear on the Assignments Page. If you are experiencing problems with the submission script, go to **meta/build/Assignment/** and upload the file **mutual-information.txt** to the Assignments Page.

Now you will run **association** to print the top 50 syntagmatically related terms based on the mutual information function you coded. You can do this by executing:

```
cd ~/Desktop/meta/build/
./association config.toml --words 50 --MI
```

where the argument "--MI" specifies that we want to use mutual information as the association measure. The top 50 word pairs will be printed along with their mutual information values. As you can see, the words are highly correlated and provide a good summary of the Yelp reviews. Note: some of the words might seem to be misspelled; this is due to stemming, which strips off some of the characters from words.

## Task 3: Topic Modeling (30%)

In the section, you will learn how to implement and experiment with Probabilistic Latent Semantic Analysis (PLSA), which is the most basic topic model, and also one of the most useful ones for mining topics and themes from text collections. PLSA assumes that each document in the corpus is generated from a set of topics represented by a unigram langauge models. Every document is also associated with a set of mixing weights that determine the proportions of topics in the document. In addition to the topic models, a fixed background language model can be also used to explain the generation of very frequent words.

Let $\theta_1, \ldots, \theta_k$ be unigram language models representing each of the $k$ topics and $\theta_B$ be the background language model which is estimated based on the frequency of the words in the corpus. Then, the probability of generating a word $w$ in a document $d$ is given by:

$$P(w) = \lambda_B P(w|\theta_B) + (1 - \lambda_B) \sum_{j=1}^{k} \pi_{d,j} P(w|\theta_j) \tag{8}$$

where $\lambda_B$ is a fixed proportion that determines the extent to which the background language model explains the generation of words. $\pi_{d,j}$ is the mixing weight of the jth topic that determines the proportion of topic $j$ in document $d$.

Our goal is to find the distributions of the topics and the mixing weights of documents that maximize the likelihood of observing all the documents in the corpus. The log-likelihood of the documents is given by:

$$\log P(C|\boldsymbol{\theta}, \boldsymbol{\pi}) = \sum_{d \in C} \sum_{w \in V} c(w,d) \log \left( \lambda_B P(w|\theta_B) + (1 - \lambda_B) \sum_{j=1}^{k} \pi_{d,j} P(w|\theta_j) \right) \tag{9}$$

where $c(w, d)$ is the frequency of $w$ in $d$. (9) is hard to maximize due to the presence of the log inside the summations. One way to maximize (9) is to use the EM algorithm, which is an iterative algorithm usually used to solve intractable maximum likelihood problems. In the E-step it tries to “guess” the probability that each word $w$ was generated using the topic $j$ by using latent variables. In the M step, it optimizes the parameters (i.e., the topic distributions and mixing weights) assuming that the guessed probabilities in the E-step are correct.

The update formulas for the E-step are given by:

$$P(z_{d,w} = j) := \frac{\pi_{d,j} P(w|\theta_j)}{\sum_{j'=1}^{k} \pi_{d,j'} P(w|\theta_{j'})} \tag{10}$$

$$P(z_{d,w} = B) := \frac{\lambda_B P(w|\theta_B)}{\lambda_B P(w|\theta_B) + (1 - \lambda_B) \sum_{j=1}^{k} \pi_{d,j} P(w|\theta_j)} \tag{11}$$

where $z_{d,w}$ is the latent variable associated with word $w$ in document $d$.

The update formulas for the M-step are given by:

$$\pi_{d,j} := \frac{\sum_{w \in V} c(w,d)(1 - P(z_{d,w} = B))P(z_{d,w} = j)}{\sum_{j'=1}^{k} \sum_{w \in V} c(w,d)(1 - P(z_{d,w} = B))P(z_{d,w} = j')} \tag{12}$$

$$P(w|\theta_j) := \frac{\sum_{d \in C} c(w,d)(1 - P(z_{d,w} = B))P(z_{d,w} = j)}{\sum_{w' \in V} \sum_{d \in C} c(w',d)(1 - P(z_{d,w'} = B))P(z_{d,w'} = j)} \tag{13}$$

After completing each M-step, the EM algorithm is guaranteed to increase the likelihood of the data until it converges to a local maximum. Practically, we usually run the algorithm several times using randomly selected parameters and choose the parameter values that yield the highest likelihood.

The EM algorithm is a very important general algorithm with many applications, thus it is essential that you understand exactly how it works for PLSA. However, asking you to implement the whole EM algorithm would require too much coding effort. As a compromise, we thus have created a special program for this task called **plsa** to minimize your effort while retaining the benefit of learning about the EM algorithm. Go to **meta/src/tools/** and open **plsa.cpp**. This file contains a naïve implementation of the PLSA algorithm that uses equations (10), (11), (12), and (13) as they are. The algorithm is almost complete with the exception of equation (11) in the E-step, which you are expected to complete.

Start from the top of the file by reading the comments inside the PLSA class to familiarize yourself with the different variables involved. Then, proceed to reading the comments inside the functions **PLSA::Estep()** and **PLSA::Mstep()** and try to understand how equations (10), (12), and (13) are implemented. Equation (11) has not been fully implemented and a part of its code is missing. Your task is to complete the implementation of equation (11) in the E-step.

The table below shows a mapping between some of the important variables that you are going to use in implementing the code and the variables used in equation (11):

| **Variable Name in plsa.cpp** | **Variable Name in Eq (10) to (13)** |
|---|---|
| pi[d][j] | $\pi_{d,j}$ |
| Pw[j][w] | $P(w\|\theta_j)$ |
| PzB[d][w] | $P(z_{d,w} = B)$ |
| lambda_B | $\lambda_B$ |
| num_topics | $k$ |
| | |

| | |
|---|---|
| PB[w] | $P(w\|\theta_B)$ |

Make sure to strictly follow the comments in the code when implementing equation (11).

We are going to run PLSA on a dummy dataset containing reviews of four types of products: cars, boats, laptops, and wearables. Go to **meta/data/reviews** and have a look at some of the documents. You should configure MeTA to use this dataset when running PLSA. To do so, open **config.toml** from **meta/build/** and replace:

```
corpus-type = "line-corpus"
dataset = "yelp"
forward-index = "yelp-fwd"
inverted-index = "yelp-inv"
```

with

```
corpus-type = "file-corpus"
list = "reviews"
dataset = "reviews"
forward-index = "reviews-fwd"
inverted-index = "reviews-inv"
```

Save the changes to the file. After coding equation (11), you should recompile MeTA by executing:

```
cd ~/Desktop/meta/build/
make
```

If you get errors from the compiler, recheck you code and make sure that your syntax is correct. Before proceeding with the exercise, submit your code by executing:

```
cd ~/Desktop/meta/build/
./plsa config.toml --submission
cd Assignment
python submit.py
```

and enter task 3 when prompted. If you experience problems with the Python submission script upload **plsa.txt**, which is located in **meta/build/Assignment**, to the Assignments Page. Make sure that you got full credit before proceeding with the instructions below.

Now you should run PLSA with the number of topics set to 2, the number of iterations to 300, and $\lambda_B$ to 0.8 by executing:

```
cd ~/Desktop/meta/build/
./plsa config.toml --topics 2 --lambda 0.8 --iter 300 --seed 11
```

where the "--seed 11" is used to randomly initialize the parameters of PLSA. While the algorithm is running, it will print out the values of the log-likelihood. You should observe how the log-likelihood always increases in the beginning before it reaches a value where it becomes constant; this means that the algorithm has converged.

After the algorithm converges, the top 20 terms in each topic will be printed. You should be able to see that the first topic summarizes the reviews on laptops and wearables and that the second topic summarizes the reviews on boats and cars. This is expected since laptops and wearables are more related to each other and can fit into the category of "technology" and, similarly, boats and cars can fit into the category of "transportation". k=2 allows us to get an overview or a coarse description of the dataset. To get a finer summary, let us set k=4 by executing:

```
./plsa config.toml --topics 4 --lambda 0.8 --iter 300 --seed 6
```

After convergence, the top 20 terms in each of the 4 topics will be printed. Clearly, each of the mined topics corresponds to exactly one of the four products in the reviews dataset. This shows how PLSA can be used to explore and summarize datasets at multiple granularities.

Note that the implementation of PLSA we used is inefficient since it copies the algorithm directly as it is without doing any optimizations. The main problem lies in the fact that we are using a huge multidimensional array to store the values of $P(z_{d,w} = j)$ whose space complexity is $O(k|C||V|)$. To make the algorithm more efficient, the E and M steps can be combined to avoid storing the multidimensional array. We made the choice of using the naïve version of the algorithm to make it easier for you to understand the code; however, in any serious implementation of the algorithm, the steps should be combined.

If you want to experiment further with topic modeling on large datasets, MeTA contains an efficient implementation of LDA (Latenet Dirichlet Analysis) which is an extension of PLSA. Refer to MeTA's Tutorial on Topic Models for more details on how to use LDA (you can run it without any coding!).

## Task 4: Text Mining Competition (30%)

In this task, you are going to use Yelp restaurant reviews to predict whether a set of given restaurants will pass the public health inspection tests. This is a representative task for some of the most important applications of text mining, i.e., using relevant text data to help make predictions on some important decision factors that are only loosely connected with the text data. In this case, for example, the hygiene condition of a restaurant can be an important decision factor when people decide where to go for a dinner. Thus by working on this task, you will gain hands-on experience with solving a text-based prediction problem and have an opportunity to explore all kinds of ideas, particularly your own new ideas, and study their effectiveness in a fun way. You will be competing with the rest of the class for a chance to appear on the leaderboard and to get the Text Mining Competition Leader Badge if you manage to be in the top 30%. You should make at least 1 submission to get the 30 points on this task (your position on the leaderboard does not affect your grade). Note: The relevant material for this task will be out in week 3. If you do not have prior knowledge about supervised learning, you can work on this task later on when the relevant video lectures are published.

The dataset is composed of Yelp reviews of restaurants along with labels that tell whether each restaurant is clean or not based on the latest health inspection test. The dataset is found in **meta/data/hygiene/**. Go to this directory and explore the files:

- **hygiene.dat** is a line corpus which contains on each line the reviews corresponding to one restaurant. There are a total of 746 restaurants in this corpus (and consequently 746 lines).
- **hygiene.dat.labels** contains the labels of the first 546 restaurants in **hygiene.dat**. A number 1 indicates that the restaurant is **not** clean as deemed by the health inspectors, while a 0 indicates that the restaurant is clean. You are **not** given the labels of the remaining 200 restaurants.

Your task is to train a classifier based on the first 546 restaurants (which you are given labels for), and then predict the labels of the last 200 restaurants. Your position on the leaderboard will be determined by the classification accuracy of your classifier on the last 200 restaurants.

You are allowed to use any programming language and toolkit for this task, although we encourage you to use MeTA since we are already providing code that can train the classifier and perform prediction, and thus you can start testing any ideas for improving classification accuracy immediately. Below we provide instructions on how to participate in the competition using MeTA or other toolkits.

### MeTA

If you are going to use MeTA, you should first configure it to use the Hygiene dataset. To do so, open **config.toml** from **/meta/src/build/** and replace:

```
corpus-type = "file-corpus"
list = "reviews"
dataset = "reviews"
forward-index = "reviews-fwd"
inverted-index = "reviews-inv"
```

with:

```
corpus-type = "line-corpus"
dataset = "hygiene"
forward-index = "hygiene-fwd"
inverted-index = "hygiene-inv"
```

Then, go to **meta/src/tools/** and open **competition.cpp**. Read and try to understand how the code is working. On a high level, the code starts by training a classifier based on the first 546 restaurants and then predicts the labels of the remaining 200 restaurants and outputs them to a file called **competition.txt** in the directory **meta/build/Assignment/**.

You can specify the classifier type from **config.toml** under the “[classifier]” tab. We have already configured **config.toml** to use Naive Bayes. If you prefer, you can choose other types of classifiers (see MeTA's Classification Tutorial).

The features to be used can also be specified using **config.toml**. Using the default settings, MeTA will use unigrams as features since ngram=1 under the [[analyzers]] tag. See MeTA's Analyzer and Filters Tutorial for instructions on how to use other kinds of features (for example, you can easily combine unigrams and bigrams without writing any code).

Now to run **competition**, execute:

```
cd ~/Desktop/meta/build/
./competition config.toml
```

The program will first ask you to enter your preferred nickname, which will appear on the leaderboard. Then it will train the classifier you specified in **config.toml** on the first 546 restaurants and predict the labels of the rest. It will also print the confusion matrix and classification accuracy on the training data.

To submit your output file, execute:

```
cd ~/Desktop/meta/build/Assignment/
python submit.py
```

Enter 4 when prompted for the task number. If you experience problems with the submission script, upload **competition.txt**, which is found in **meta/build/Assignment/**, to the Assignments Page. Check the feedback in the Assignments Page to see if your submission was accepted. In case your nickname is already in use, the submission will be rejected and you will be asked in the feedback to change your nickname. Also, check the leaderboard for your position.

### Other Languages/Toolkits

You can use any language and toolkit as long as you submit the output in the required format. You should prepare a text file that contains on the first line the nickname that you prefer to appear on the leaderboard. The rest of the lines should contain the labels of the 200 restaurants predicted by your system, i.e., each line should have the value 1 or 0, indicating whether the restaurant is clean or not. Your output file should look like:

```
nickname
label 1
label 2
.
.
.
label 200
```

Make sure that you strictly stick to this format and keep one empty line at the end of the file (i.e., the file should contain **exactly** 202 lines).

Name your output file **competition.txt** and place it in **meta/src/Assignment**. In the terminal, execute:

```
cd ~/Desktop/meta/build/Assignment/
python submit.py
```

Enter 4 when prompted for the task number. If you experience problems with the submission script, upload **competition.txt** to the Assignments Page. Check the feedback in the Assignments Page to see if your submission was accepted. In case your nickname is already in use, the submission will be rejected and you will be asked in the feedback to change your nickname. Also, check the leaderboard for your position.

### General Tips to Improve Classification Accuracy

Below we provide some general advice that may help you get started in improving the classification accuracy. You are encouraged to try whatever ideas you may come up with.

1. Try bigrams as features. You can do this in MeTA by setting “ngrams” in **config.toml** to 2. Make sure to delete **hygiene-inv** and **hygiene-fwd** from **meta/build/** before running competition again (this ensures that the dataset will be indexed again based on bigrams instead of unigrams). You can also try combinations of ngrams, for example a combination of unigrams and bigrams. Refer to MeTA's Analyzer and Filters Tutorial for more details.
2. Experiment with different types of classifiers. In MeTA, you can specify the classifier under the “[classifiers]” tag. For instance, you can try using a SVM or stochastic gradient descent with different loss functions. Refer to MeTA's Classification Tutorial for more information. You can also experiment with kernels or use non-linear classifiers such as decision trees and random forests (if supported by the toolkit).
3. Experiment with new features that carry richer semantics than ngrams. A simple example is to extract the top K syntagmatically related words to "dirty", $w_1, \dots, w_K$, and use $\mathrm{Count}(\mathrm{dirty}, w_i)$ for all $i \in \{1, \dots, K\}$ to augment your features. You can be more creative and come up with better features!
4. Use some of the ideas you will learn in the upcoming lectures on sentiment analysis.

### How Does the Leaderboard Operate?

- It will rank all submissions in descending order of the classification accuracy.
- In case you make multiple submissions, it will only show the highest accuracy you have achieved so far.
- In the event of ties, the contestant who submitted earlier will have the higher rank.

Submit Now