Introduction:

We have a training dataset and a test dataset in the project. These two datasets contain several web pages and have been divided into 3 class: student, faculty and course. For the data preprocessing, (1) we use two shell files to extract all the words form the given dataset at the beginning. Then we use two java programs preprocessing.java and stemmer.java to deal with the stop words, stemming and rename each page. (2) After that we put the renamed folder for both datasets to Hadoop, using four python code running by one shell scripting file to calculate the TF-IDF value of each word among all the webpage for two datasets respectively (463 pages for training data and 206 pages for testing data). (3) Later, we use java program CreateTable to transfer the result of training data that we get from the step 2 to an inverted index file. We put that inverted file to weka using Information Gain method to selected the first 305 words from word of training. (4) Create the CreateTestTable.java to generate the test table based on the two file: TF-IDF file of test data from step 2 and word selected file from step 3. After data preprocessing, we put both train table from step 3 and test table from step 4 to Weka and learn the classification model.

Data Preprocessing

A. Preprocessing for Training Data
    a. Split words:
        Input file folder: the original folder named train & test data set that professor gives
        i. Language & Tools: shell
        ii. Corresponding Class: extractmultfiles.sh, extractwords.sh
        iii. Output file folder: **words-train & words-test folder**
    b. Rename the name of each web pages
        i. Input file folder: files in word-train & word-test folder
        ii. Languages & Tools: Java, Eclipse
        iii. Corresponding Class: PreProcessing.java
        iv. The reason for doing this is every webpage has special characters, that will pop out error message of the Hadoop if we don't rename it
    c. Remove the stopwords

Introduction:

We have a training dataset and a test dataset in the project. These two datasets contain several web pages and have been divided into 3 class: student, faculty and course. For the data preprocessing, (1) we use two shell files to extract all the words form the given dataset at the beginning. Then we use two java programs preprocessing.java and stemmer.java to deal with the stop words, stemming and rename each page. (2) After that we put the renamed folder for both datasets to Hadoop, using four python code running by one shell scripting file to calculate the TF-IDF value of each word among all the webpage for two datasets respectively (463 pages for training data and 206 pages for testing data). (3) Later, we use java program CreateTable to transfer the result of training data that we get from the step 2 to an inverted index file. We put that inverted file to weka using Information Gain method to selected the first 305 words from word of training. (4) Create the CreateTestTable.java to generate the test table based on the two file: TF-IDF file of test data from step 2 and word selected file from step 3. After data preprocessing, we put both train table from step 3 and test table from step 4 to Weka and learn the classification model.

Data Preprocessing

A. Preprocessing for Training Data
    a. Split words:
        Input file folder: the original folder named train & test data set that professor gives
        i. Language & Tools: shell
        ii. Corresponding Class: extractmultfiles.sh, extractwords.sh
        iii. Output file folder: **words-train & words-test folder**
    b. Rename the name of each web pages
        i. Input file folder: files in word-train & word-test folder
        ii. Languages & Tools: Java, Eclipse
        iii. Corresponding Class: PreProcessing.java
        iv. The reason for doing this is every webpage has special characters, that will pop out error message of the Hadoop if we don't rename it
    c. Remove the stopwords

i. Languages & Tools: Java, Eclipse
ii. Corresponding Class: PreProcessing.java

d. Stemming
   i. Languages & Tools: Java, Eclipse
   ii. Corresponding Class: PreProcessing.java, Stemmer.java
   iii. Output file folder: **word-train** & **word-test** folder

e. Build an inverted index file for the training data set.
   i. Iutput file folder: **word-train** & **word-test folder** & **Python programm**
   ii. Using MapReduce to Calculate TF/IDF
      1. Languages & Tools: Python, Shell Scripting, Docker, Hadoop
      2. Main Method: MapReduce. (Learned from EECS4415 Big Data)
         a. Wrote python codes to calcualte TF/IDF values for each <word, filename> pair.
         b. Wrote a shell scripting file to running the python codes in Hadoop container. So that the MapReduce mechanism allow us to calculate the TF/IDF values and generating the outputs parallelly.
         c. Store and sort the output values in one file: word-train.txt
      3. Corresponding Files:
         a. Python: tfMap.py, tfReduce.py, idfMap.py, idfReduce.py
         b. Shell Scripting: job.sh
         c. Using instruction: cover.txt
      4. Output File: **word-train.txt**
      5. Output Format:

         word1,directory/filename1 tfidf1
         word2,directory/filename2 tfidf2
         word3,directory/filename3 tfidf3
         word4,directory/filename4 tfidf4
         ……..

   iii. Using Notepad to Format the Output File and Save as .csv File

1. Output File: **word-train.csv**
2. Output Format:

   word,filename,TF/IDF,Class

   word1,directory/filename1,tfidf1

   word2,directory/filename2,tfidf2

   word3,directory/filename3,tfidf3

   word4,directory/filename4,tfidf4

   ……...

f. Select a "bag of words"
   i. Languages & Tools: Excel.
   ii. Method: In our inverted index file(word-train.csv), sort the rows by their TF/IDF values. Select the first 20000 rows.
   iii. Output: **word-train-selected.csv**
g. Generate the training data table.
   i. Languages & Tools: Java, Eclipse
   ii. Corresponding Class: CreateTable.java
   iii. Main Method:
      1. Iterate the inverted index file(word-train.csv) line by line, store each <word, filename> pair as key and their TF/IDF number as value in a Hashtable.
      2. When creating the table, we set the words as attributes(column name), and set filenames as the element id(row name). For each element value in $<word_j, filename_j>$, if this pair can be found in Hashtable, the value is their TF/IDF. otherwise, it is 0.
   iv. Output: **word-train-table.csv**
h. Using Weka to Select Attributes and Create .arff File
   i. Languages & Tools: Weka
   ii. Main Method:
      1. Open our word-train-table.csv in Weka
      2. Choose Using filters-supervised-attribute-AttributeSelection as our filter.
      3. Use **Information Gain** algorithm to sort our attributes.

4. Only Remain the attributes with ranked > 0. (First 305 attributes.)
5. Save it as .arff file.

iii. Output File: **word-train-table-305.arff**

B. Preprocessing for Testing Data

a. Split words:
   i. Same as [A. Preprocessing for Training Data]

b. Rename the web pages
   i. Same as [A. Preprocessing for Training Data]

c. Remove the stopwords
   i. Same as [A. Preprocessing for Training Data]

d. Stemming
   i. Same as [A. Preprocessing for Training Data]

e. Build an inverted index file for the testing data set
   i. Same as [A. Preprocessing for Training Data]
   ii. Output File: **word-test.csv**

f. Check our "bag of words", remove the un-selected words from the inverted index file for testing data set.
   i. Languages & Tools: Java, Eclipse, Notepad
   ii. Corresponding Class: PreprocessingTestData.java
   iii. Main Method:
      1. Using notepad to store our selected words into a txt file: **selected-words.txt**
      2. Run class PreprocessingTestData.java: Check our inverted index file for testing data line by line. If the first word of the line is not contained in our selected "bag of words", remove it.
   iv. Output File: **word-test-selected.csv**

g. Generate the testing data table.
   i. Language & Tools: Java, Eclipse
   ii. Corresponding Class: CreateTestTable.java
   iii. Main Method:
      1. The logic is same as A.g

2. The column-words should be the same as training data table after selecting.

iv. Output File: **word-test-table.csv**

h. Using Weka to Create .arff File

i. Language & Tools: Weka

ii. Main Method:

1. Remove the filename attribute, save as .arff file.

iii. Output File: **word-test-table-305.arff**

Algorithms

We tried to imply 6 algorithms in the project and compare their accuracy to get best classification model.

The table below represent the percentage for correctly classified instances in two situations: use 10 folder cross-validation and use the test dataset .

	C4.5	RIPPER	Naive Bayesian Classification	Bayesian Network	k-Nearest Neighbor	Neural networks
10 folder cross validation	79.4372%	79.4372%	**83.9827%**	83.7662%	80.3030%	81.8182%
test data set	54.8544%	74.7573%	73.7030%	76.2136%	70.3883%	79.1262%

Since the accuracy of **Naive Bayesian** is the highest. We would like to choose it as our learning algorithm.

The testing statistic of the learned model on the test data:

=== Summary ===

Correctly Classified Instances 151 73.301 %
Incorrectly Classified Instances 55 26.699 %
Kappa statistic 0.5669
Mean absolute error 0.1804
Root mean squared error 0.421
Relative absolute error 48.6664 %
Root relative squared error 98.9114 %
Total Number of Instances 206

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.680	0.038	0.967	0.680	0.798	0.627	0.919	0.950	student
	0.676	0.070	0.657	0.676	0.667	0.600	0.812	0.573	faculty
	0.932	0.247	0.506	0.932	0.656	0.575	0.859	0.507	course
Weighted Avg.	0.733	0.088	0.817	0.733	0.746	0.611	0.888	0.793	

=== Confusion Matrix ===

```
  a  b  c   <-- classified as
 87 12 29 |  a = student
  0 23 11 |  b = faculty
  3  0 41 |  c = course
```

Discussion and Conclusion

When selecting data, we are confused with how to select our “bag of data”. Since the method we used is TF/IDF, we sort the total 90000 data by their TF/IDF and choose the first 20000 ones. However, we actually do not know what the threshold of TF/IDF we should use. Thus, we also used the total 90000 data to create our training data table, the accuracies are higher.

	C4.5	RIPPER	Naive	Bayesian	k-Nearest	Neural

			Bayesian Classification	Network	Neighbor	networks
10 folder cross validation	90.5172%	87.931 %	87.5 %	93.5345%	82.7586%	87.6182%
test data set	77.1845%	75.2427%	84.9515%	83.9806%	44.1748%	

If we use the total data without selection, the results looks better, and **Bayesian Network** becomes the best learning algorithm.

The testing statistic of the learned model on the test data:

=== Summary ===

Correctly Classified Instances	173	83.9806 %
Incorrectly Classified Instances	33	16.0194 %
Kappa statistic	0.7276	
Mean absolute error	0.1117	
Root mean squared error	0.3042	
Relative absolute error	30.1333 %	
Root relative squared error	71.4649 %	
Total Number of Instances	206	

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0.824	0.070	0.700	0.824	0.757	0.707	0.955	0.821	faculty
0.789	0.051	0.962	0.789	0.867	0.716	0.966	0.979	student
1.000	0.105	0.721	1.000	0.838	0.804	0.997	0.990	course

Weighted Avg. 0.840 0.066 0.867 0.840 0.843 0.733 0.971 0.956

=== Confusion Matrix ===

```
  a   b   c   <-- classified as
 28   4   2 |   a = faculty
 12 101  15 |   b = student
  0   0  44 |   c = course
```

==========

If using the data without selection can get better result, then why we need to select a "bag of data"? Is the selection method(Sort by TF/IDF) we used correct?

appendix

the way to run the program:

1. splitting words : mybox% extractmultfiles.sh
2. for the java program we can just use the eclipse to run it directly
3. The procedure for you to invoke python program in hadoop:
   1. open the Docker on you laptop. (this is the container for the Hadoop)

``

   2. open the terminal type the following words to start the docker
      mybox% docker start Hadoop
   3. then copy the file from the computer home filesystem to the docker container
      mybox% docker cp projectdata Hadoop:/home/projectdata
   4. to get the bash shell within the container type
      mybox% docker attach Hadoop
   5. if you want to check if the file you put inside is already inside the container, type
      bash$ bin/hadoop fs -ls
   6. copy the input file to Hadoop's hfs filesystem type:
      bash$ bin/hadoop fs -put /home/projectdata/word-train word-train
   7. to excuse the Hadoop MapReduce job, run the shell file
      bash# cd /usr/local/hadoop
      bash# sh /home/projectdata/job.sh 2>&1 | tee /home/runLog