CSE 5243 Lab-2 Report

Harsh Gupta (gupta.749), Dhanvi Athmakuri (athmakuri.1)

October 9th 2016

1 Problem Statement

The purpose of this assignment is to implement and analyze the performances of KNN, Naive Bayes and Decision Tree classification algorithms, on the dataset produced from the pre-processing done in lab-1. This report also discusses the challenges faced in the implementation and the way they were handled.

2 Assumptions made in the implementation

2.1 Naive Bayes and Decision Tree

1. Documents which had multiple topic labels were creating a problem, so we thought of replacing it with any one of the topic label out of all labels assigned to it, as discussed in class. After analyzing the document we realized it will be wrong to assign any random topic label out of the list and therefore assigning the label with maximum frequency in the entire data set will essentially over represent that label. Hence we didn't change the multi topic labels and concatenated it into one single label.

2. To combat the assumption for multi label problem, we have created 8 samples of uniform width to run the training and testing set model. This will help to restrict the over representation of any particular label and hence will give more accurate result.

2.2 KNN

No major assumptions about the data have been made in implementing KNN.

3 Implementation Details

Project Participants : This implementation has been accomplished by the collaboration of Dhanvi Athmakuri (athmakuri.1) and Harsh Gupta (gupta.749).
Participant Contributions : Dhanvi Athmakuri implemented and analyzed the KNN algorithm and Harsh Gupta implemented and analyzed the performance of Naive Bayes classifier and Decision Tree.
Implementation language and libraries used: We have used *python-2.7* programming language to implement our pre-processing methodology. The following libraries have been used to help process the files :

- `sklearn` - Provides the implementation of KNN, Naive Bayes, Decision Tree and several other machine learning algorithms. Also, provides libraries to analyse the performances of these algorithms.
- `cross_validation` - Belongs to the `sklearn` library. Provides methods to split a given data file into training and testing sets. This implementation randomly assigns 70% of the data file as the training set and the remaining 30% as testing set.
- `MultiLabelBinarizer` -Belongs to the `sklearn` library. Provides methods to handle Multi Label Classification tasks.
- `KNeighborsClassifier` - Provides methods to run the KNN algorithm, when input is given in a specified format.
- `NaiveBayesClassifier` - Provides methods to run the Naive Bayes algorithm, when input is given in a specified format.
- `DecisiontreeClassifier` - Provides methods to run the Decision Tree algorithm, when input is given in a specified format.

Other libraries like `nltk, random, sys, csv, time, math, collections, gc, operator` have been used for utility functions.

3.1 KNN Implementation

This report presents two different implementations of the KNN algorithm. The first implementation (Implementation-I) uses textttsklearn library to implement the algorithm, where as the second implementation (Implementation-II) is stand alone code written by Dhanvi Athmakuri.

3.1.1 Implementation-I Details

The first implementation is given a data file as input. This is precisely the file which was generated by the preprocessing code of lab-1.

Following is the step-by-step procedure of how the first implementation proceeds.Below every individual point, we have included the line(s) of code that execute the task presented by the point.

1. Read the appropriate file, given as the input. Store the data vectors and the list of topics for every instance, in specific data structures.
2. Handle the multi-label problem, by binarizing the topics. This essentially associates every data vector, with a binary string with 0&1 indicating the absence and the presence of a certain label in the list of topics for a particular instance.This is accomplished by
   `MultiLabelBinarizer().fit_transform()`
3. Randomly split the data into training and testing sets. In this implementation, we have consistently used the $70\% - 30\%$ split. This is accomplished by the following line of code:
   `cross_validation.train_test_split(data vectors, labels, test_size=0.3,random_state=1)`
4. fit the model using the training data sets using an appropriate value for k and predict the classification labels of the testing dataset.This is accomplished by
   `KNeighborsClassifier(n_neighbors=k).fit(train_vectors,train_labels)`
   `accuracy = classifier.score(test_vectors,test_labels)`
   `predictions = classifier.predict(test_vectors)`
   Here, `classifier` is a reference for `KNeighborsClassifier(n_neighbors=k)`

3.1.2 Performance Evaluation(Accuracy Calculation)

Accuracy is calculated using a **strict evaluation criterion**. That is, if and only if the prediction vector matches completely with the target vector, it is counted as a correct prediction. This measure has been chosen, because it is a more conservative measure and any other accuracy calculation would definitely yield better results. Accuracy is calculated as the number of right prediction to the total number of prediction.
For example:
Accuracy Calculation in Naive Bayes and Decision tree :
testSet = [[1,1,1,'a'], [2,2,2,'a'], [3,3,3,'b']] **predictions** = ['a', 'a', 'a']
Output: **Accuracy**: 66.67 % **Accuracy calculation in KNN**
Here, prediction for an instance is a vector, to handle multi label classification
testSet = [[1,1,1],[0,0,1]], [[2,2,2],[0,1,1]], [[3,3,3],[0,0,1]] **predictions** = [[1,1,0],[0,1,1],[1,0,0]]
Output: **Accuracy**: 33.33%

3.1.3 Results of Implementation-I (KNN)

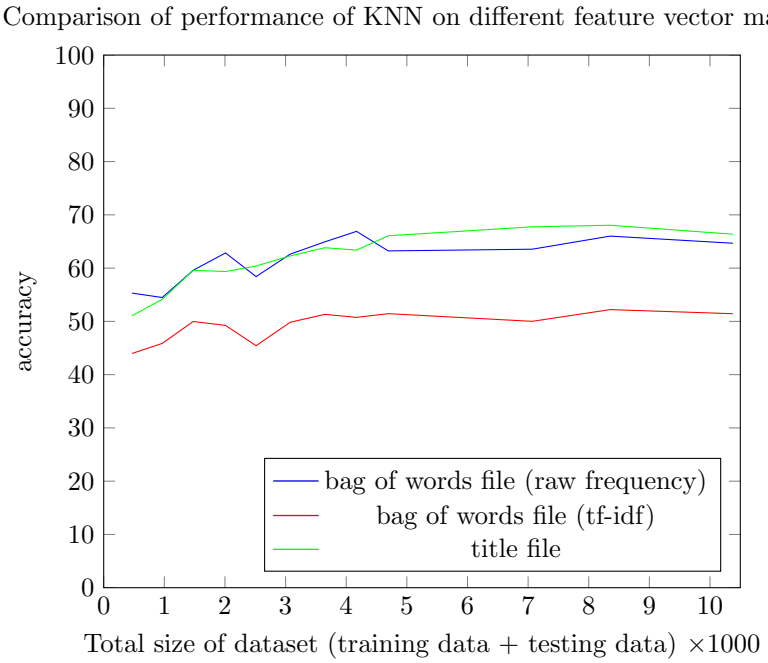
In this sections, we present the results obtained by running the KNN algorithm provided by `sklearn.KNearestNeighbors` library, over datasets of different sizes and types and over different k values. We present most of our results graphically or in tabular form. Detailed discussion is omitted due to paucity of space. The x-axis represents the size of the input dataset. There are over $21,000$ individual articles in the reuters dataset, out of which only $10,377$ have non-empty topics. Only, these files have been used for prediction.

From the figure below, we can see that the accuracy is ranging from $40\% - 70\%$, which is rather good, considering that documents have multiple labels. It is clear that the accuracy is consistently lower when using tf-idf measure than when compared to using raw frequency. Thus, from this analysis, we conclude that raw frequency should be preferred over tf-idf measure, for this particular dataset, for further processing. Also, we can see that the accuracy obtained using the *title file* is comparable to that of the *bag of words file* , upto a certain size and becomes even better after that. This aligns with our expectation that the words in the title of a document are more relevant in predicting the classification label of that document. The title file(s) being smaller in size, are also easier to process. Thus feature vector files, created from the title words are to be preferred over the bag of words feature vectors.

Table 1: Accuracy and Coverage Error over different k values

.

k-value	accuracy	coverage error
3	67.43%	37.22
5	64.67%	41.11
7	63.10%	43.30
9	61.8%	44.89
11	60.85%	46.2
15	58.54%	49.27



The primary measure of performance used, is accuracy. However, we have also used *coverage error*, since it is relevant to multi label classification. Coverage error gives us the average number of labels to be included in the final prediction, so that all true labels are predicted. The table 1, shows the values of accuracy and coverage errors for different k-values, over all the $10,377$ files.

From table 1, we can see that the accuracy is steadily decreasing and the coverage error is steadily increasing with increasing k-value. This can be explained by the possible addition of noise with the addition of extra neighbors. 2 shows the accuracy and online-offline execution times over different training-testing split ratios, for a k-value of 5.

Table 2: Accuracy and Computation time over different split ratios

training% - testing%	**accuracy**	**offline cost(sec)**	**online cost(sec)**
$60\% - 40\%$	64.23%	21.70	867.855
$70\% - 30\%$	64.65%	13.99	661.95
$80\% - 20\%$	64.35%	19.55	587.56

3.2 Implementation-II Details (KNN)

Implementation-II follows the same program structure, as given in `http://www.kdnuggets.com/2016/01/implementing-your-own-knn.html`. However, to handle multi label classification, the following research paper has been referred to : `Zhang, Min-Ling, and Zhi-Hua Zhou. "ML-KNN: A lazy learning approach to multi-label learning." Pattern recognition 40.7 (2007): 2038-2048.` The paper follows *maximum aposteriori principle*, to predict the target labels. If there are N labels in total, then each target label vector would be of the form $< 0/1, 0/1, ...0/1, >$, where the '0/1' indicates 0 or 1, which tell whether or not the vector belongs to a particular label class. The maximum aposteriori principle computes the final prediction label vector as follows(notation adapted from the mentioned research paper) :

$$\vec{y_t}(l) = arg\ max_{b \in \{0,1\}} P(H_b^l | E^l_{C_t\vec{(l)}})$$

where, H_b^l denotes the event that the target vector belongs to class l if $b = 1$, and does not belong to label class l, if $b = 0$. $\vec{C_t}(l)$ is called the *membership counting vector*. It tells how many of the k neighbors of t belong to the label l. $E_(j)^(l)^l$ denotes the event that among the k nearest neighbors of t, exactly j belong to the label class l. Then target label vector can be computed from the bayesian rule as :

$$\vec{y_t}(l) = arg\ max_{b \in \{0,1\}} P(H_b^l) \times P(E^l_{\vec{C_t}(l)} | H_b^l)$$

The above probabilities can be computed from the training set. Due to paucity of space, further details of implementation are not presented in this report. The code pertaining to this implementation is included in the submission.

3.2.1 Results of Implementation-II

Implementation-II requires to compute the k-nearest neighbors of all the examples in the training set, before running the test set, in order to compute the probabilities. This consumes significant computation time. We have considered this time as offline cost for KNN in this implementation. The script `offline_processor.py` computes all the neighbors of the training instances and stores them in the `neighbours.csv` file. It also initializes the files `training_data_file.csv` and `testing_data_file.csv`. The script

`ml_knn.py` read these files and performs prediction operation by computing the above probabilities. Further details are given in the README file. The off-line cost for initializing the neighbors of the training instances, is over **518 seconds** for only 467 files, and over **2000 seconds**, for 965 files. The online cost for a thousand files is **91.83 seconds**. with an **accuracy of 56.02%**, for 467 files. The online cost for 965 files is found to be 609.46*seconds*, with an accuracy of 56.55%.The online and offline cost increase dramatically, for larger files. Thus, the results for the larger files have not been computed. However, this research paper, presents a novel approach to tackle the multi-label classification problem.

3.3 Naive Bayes and Decision Tree Implementation

This report presents implementations of Naive Bayes and Decision Tree algorithm. The first implementation (Implementation-I) uses GaussianNBsklearn and the second implementation (Implementation-II) uses DecisionTreeClassifiersklearn library to implement the algorithm.

3.3.1 Naive Bayes Implementation

The first implementation is given a data file as input. This is precisely the modified version of the file which was generated by the preprocessing code of lab-1.

Following is the step-by-step procedure of how the first implementation proceeds.Below every individual point, we have included the line(s) of code that execute the task presented by the point.

1. Read the appropriate file, given as the input. Store the data vectors and the list of topics for every instance, in specific data structures.
2. Randomly split the data into training and testing sets. In this implementation, we have consistently used the $70\% - 30\%$, $75\% - 25\%$ and $80\% - 20\%$. This is accomplished by the following line of code:
   `splitData(filename,split_ratio, i)`
3. Fit the model using the training data sets using an appropriate value for k and predict the classification labels of the testing dataset.This is accomplished by
   ```
   nb_model.fit(TrainingSet, Training_labels)
   predicted = nb_model.predict(TestingSet)
   accuracy= nb_model.score(TestingSet,Testing_labels,sample_weight=None)*100
   ```
4. Finally a report and a confusion matrix is created according to the predicted value in the testing phase and the actual values. This is accomplished by
   ```
   report = metrics.classification_report(Testing_labels, predicted)
   confusionMatrix= metrics.confusion_matrix(Testing_labels, predicted)
   ```

3.3.2 Results

In this sections, we present the results obtained by running the Naive Byes algorithm provided by `sklearn.naive_bayes` library, over data set over a single sample out of 8 samples which can be created. We present most of our results in tabular form and files which contains the report and confusion matrix respectively. Detailed discussion is omitted due to paucity of space.

The primary measure of performance used, is accuracy. However, we have also used *precision, recall, f1 measure*, since it is important in classification. The output of these metrics are not listed here due to paucity of space. It can be seen in the folder written in ReadMe file. The table 3 and 4, shows the values of accuracy, offline and online time for different split ratio.

Table 3: Word Document Frequency Vector (Samples 1 out of 8)

Split	**Accuracy**	**Offline Cost**	**Online Cost**
$80 - 20$	79.16	15.81	5.79
$75 - 25$	75.75	14.26	3.52
$70 - 30$	75.50	14.12	4.26

From table 3 and 4, we can see that the accuracy is comparatively higher in case of word document frequency vector than tf-idf feature vector. Though TF-IDF is a more correct measure for preprocessing but for classification sometimes word frequency is a stronger measure for predicting labels.

Table 4: TF-IDF Document Frequency Vector (Samples 1 out of 8)

Split	Accuracy	Offline Cost	Online Cost
$80-20$	77.65	15.36	4.07
$75-25$	74.54	15.26	3.54
$70-30$	72.97	14.23	6.02

3.4 Decision Tree Implementation

The first implementation is given a data file as input. This is precisely the modified version of the file which was generated by the preprocessing code of lab-1.

Following is the step-by-step procedure of how the first implementation proceeds.Below every individual point, we have included the line(s) of code that execute the task presented by the point.

1. Read the appropriate file, given as the input. Store the data vectors and the list of topics for every instance, in specific data structures.
2. Randomly split the data into training and testing sets. In this implementation, we have consistently used the $70\% - 30\%$, $75\% - 25\%$ and $80\% - 20\%$. This is accomplished by the following line of code:
   `splitData(filename,split_ratio, i)`
3. Fit the model using the training data sets using an appropriate value for k and predict the classification labels of the testing dataset.This is accomplished by
   ```
   DT_model.fit(TrainingSet, Training_labels)
   predicted = DT_model.predict(TestingSet)
   accuracy= DT_model.score(TestingSet,Testing_labels,sample_weight=None)*100
   ```
4. Finally a report and a confusion matrix is created according to the predicted value in the testing phase and the actual values. This is accomplished by
   ```
   report = metrics.classification_report(Testing_labels, predicted)
   confusionMatrix= metrics.confusion_matrix(Testing_labels, predicted)
   ```

Please read the README file on how to execute the program, with different options.

3.4.1 Results

In this sections, we present the results obtained by running the Naive Bayes algorithm provided by `sklearn.DecisionTreeClassifier` library, over data set over a single sample out of 8 samples which can be created. We present most of our results in tabular form and files which contains the report and confusion matrix respectively. Detailed discussion is omitted due to paucity of space.

The performance metrics are same as Naive Bayes and can be seen in the folder as stated in ReadMe file.

Table 5: Word Document Frequency Vector (Samples 1 out of 8)

Split	Accuracy	Offline Cost	Online Cost
$80-20$	70.27	63.84	0.21
$75-25$	65.25	64.47	0.29
$70-30$	66.60	64.91	0.34

Table 6: TF-IDF Document Frequency Vector (Samples 1 out of 8)

Split	Accuracy	Offline Cost	Online Cost
$80-20$	77.27	16.76	0.01
$75-25$	76.66	15.29	0.02
$70-30$	77.77	15.80	0.02

From table 5 and 6, we can see that the accuracy is comparatively higher in case of tf-idf feature vector, the offline cost is comparatively lower and the online cost is almost half the value as compared with the document frequency vector classification metrics. As decision tree is more scalable hence it work better for tf-idf as tf-idf scales the document for the entire data set contrary to word frequency.
The most interesting outcome of decision tree classification is its online cost which is negligible when compared to Naive Bayes cost. This is again because decision tree is more scalable than Naive Bayes.