DS222: MLLD 2018
Assignment 1 : Naive Bayes classifier implementation

Sourabh Balgi
Sr. No. - 14318
EE Department
M. Tech. (Systems Engineering)
sourabhbalgi@gmail.com

Abstract

Naive Bayes Classification is one of the simplest, yet effective and most commonly-used machine learning classifier. Since the variables are considered independent, it's easier for performing computations in case of large number of variables. This document summarize the results and observations of dealing with very large datasets DBPedia and distributed computing using HADOOP mapreduce. As a part of the assignment, we implement two variants of Naive-Bayes classifier one in-memory and the other using HADOOP mapreduce.

1 Introduction

Naive Bayes classifier is a defacto method to go for when a large number of data is not available. It's one of the easiest classifiers as it makes no assumptions on the interdependence of the features in the corpus. Considering the availability of the huge corpus of data now-a-days, It's much required to go for simple methods which can be implemented without much feature engineering and usage of massive computational resources. With large data, in memory computation becomes much harder and we need to explore distributed computing. One such tools we are exploring in this assignment is HADOOP.

1.1 Dataset

We use DBPedia 2015 dataset which consists of 50 classes based on the labels obtained from mapping based type of the documents.

The columns are separated by space-tab and the list of labels are comma seperated. Each line is a uunique document with labels and document data.

2 Naive Bayes classifier

Naive Bayes involves maximizing the posterior probability assuming Independence of the conditional density. The given problem instance to be classified, represented by a vector $s = (w_1, ..., w_n)$ representing some n words w_i in a sentence (independent variables), it assigns to this instance probabilities

$$p(L_m|w_1, ..., w_n)$$

for each of m possible outcomes or labels L_m.

The problem with the above formulation is that if the number of features n is large or if a feature can take on a large number of values, then basing such a model on probability tables is infeasible. We therefore reformulate the model to make it more tractable. Using **Bayes' theorem**, the conditional probability can be decomposed as,

$$p(L_m|s) = \frac{p(L_m)p(s|L_m)}{p(s)}$$

Thus, the above equation can be written as,

$$posterior = \frac{prior * conditional}{\Sigma prior * conditional}$$

2.1 Naive Bayes Algorithm

To eliminate zeros, we use add-one or Laplace smoothing, which simply adds one to each count

$$\hat{P}(w|class) = \frac{C_w + 1}{\sum_{w' \in V}(C_{w'} + 1)} = \frac{C_w + 1}{(\sum_{w' \in V} T_{w'}) + |V|}$$

where *C* indicates actual count of word *w*, *V* is set of vocabulary and —*V*— is the size of vocabulary

Algorithm 1: Naive Bayes Classifier : Training

1 Pre-process the training data using stopword removal, lemmatizer and tokenizer.
2 Group the documents belonging to the same class and count the number of documents for each class to obtain the prior probability using

$$prior = \frac{class\_count}{total\_class\_count} \quad (1)$$

3 Find the conditional density of each class with add-one smoothing for numerical stability and unknown vocabulary handling.

$$cond_{w,m} = \frac{wordcount_m + 1}{totalword_m + vocabsize} \quad (2)$$

Algorithm 2: Naive Bayes Classifier : Inference

1 Pre-process the training data using stopword removal, lemmatizer and tokenizer.
2 For each document, find the posterior densities for all the classes using Bayes rule.

$$p(L_m|s) = \frac{p(L_m)p(s|L_m)}{p(s)} \quad (3)$$

3 Consider the class with the maximum posterior probability as the predicted class.
4 Find and report accuracy using

$$Accuracy = \frac{total\_correct\_prediction}{total\_number\_of\_documents} \quad (4)$$

2.2 Naive Bayes on Hadoop MapReduce framework

We implement the mapreduce using Hadoop streaming. Hadoop streaming is a utility that comes with the Hadoop distribution. This utility allows us to create and run Map/Reduce jobs with any executable or script as the mapper and/or the reducer. In this assignment, both the mapper and the reducer are python scripts that read the input from standard input and emit the output to standard output. We control the number of reducers used by the mapreduce per job to observe the run-times

3 Hadoop Approach

3.1 Map Reduce model description

3.1.1 Training Phase :

1. **Get class word count** : In this job, we get the counts of individual classes and counts of individual words present in the training corpus. The reducer phase sums these individual counts and emits the total used by the next Job to find the prior and conditional densities. Here key is class label and word. Value corresponding to them are the class count and the word count per class.

2. **Get model parameters** : Here, the output from the previous job **Get class word count** is used to find the prior probability and the conditional probability of each word per class.

3.1.2 Inference Phase :

This involves 3 separate mapreduce phases.

1. **Get document word count** : We consider each document with a unique document id i.e. line number as the key and the words in this document in the mapper phase. In the reduce phase, we sum all the key and emit the key - (doc id, word) : value - (word count)

2. **Calculate posterior probability** : We use the outputs from **Get model parameters** Job i.e. probabilities (prior and conditional) and Get document word count Job i.e. word count in the test document.

3. **Calculate accuracy** : We use the outputs from **Calculate posterior probability** Job.

We sum the prior and conditional probabilities according to documents and find the label corresponding to the maximum posterior probability and emit as predicted label for the document.

4 Observations and Results

1. **In-Memory** :
   Vocab Size :
   1059468(without stemming and stopword removal).
   967298(with stemming (porter stemmer) and stopword removal (nltk).
   In-Memory Run Time :

Table 1: Run Time

Datset Size	Train	Train+Test
Very Small	1 min 16 sec	3 min 11 sec
Small	3 min 34 sec	5 min 20 sec
Full	10 min 36 sec	16 min 30 sec

2. **Hadoop Mapreduce** :
   Vocab Size : (Same as above)
   1059468(without stemming and stopword removal).
   967298(with stemming (porter stemmer) and stopword removal (nltk).
   Mapreduce Run Time :

Table 2: Run Time

Method	Training Time
Map Reduce (R=1)	5 min 10 sec
Map Reduce (R=2)	4 min 41 sec
Map Reduce (R=5)	2 min 42 sec
Map Reduce (R=8)	2 min 10 sec
Map Reduce (R=10)	1 min 15 sec

Observation :

1. Mapper takes almost same time with any number of reducers.
2. Time taken reduces as the number of reducers increase.
3. Each reducer file creates separate output file with prefix **'part-'** in the output folder.

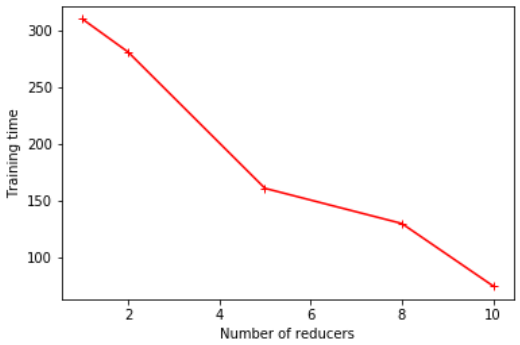

Figure 1: Training time vs Number of reducers

4. The successful completion of the Job is indicated by **'_SUCCESS'** file in the output folder on hdfs.
5. In case a job fails, the Tasker reassigns the job with multiple attempts to successfully complete the job. This robustness to failure was also observed as some of the jobs failed near completion were restarted.
6. The number of reducers has to be chosen according to our input file size, our algorithm for parallel computation and execution time.

4.1 Accuracy

Table 3: Accuracy

Method	Train(%)	Devel%)	Test(%)
In-memory full	85	73	73.4
In-memory small	83.8	85.1	84.4
In-memory very small	86.7	87.6	86.1
Map Reduce (full)	83.2	73.6	72.8

References

[1] I. Arieli, M. Koren, and R. Smorodinsky, "The Crowdfunding Game," Conference on Web and Internet Economics, 2017.

[2] R. Strausz, "A Theory of Crowdfunding - a mechanism design approach with demand uncertainty and moral hazard," American Economic Review , 2017

[3] S. Alaei, A. Malekian and M. Mostagir, "A Dynamic Model of Crowdfunding," ACM Conference on Economics and Computation, 2016.

[4] Y. Yang, H.J. Wang and G. Wang, "Understanding Crowdfunding processes: a dynamic evaluation and simulation approach," Journal of Electronic Commerce Research, 2016.

[5] M. Ellmann and S. Hurkens, "Optimal Crowdfunding Design," *NET Institute Working Papers*, 2016.