ADVANCED MACHINE LEARNING
Assignment 1
Text Classification On Reuters Dataset

By , Arindam Adak
Ramakrishna Mission Vivekananda University
Roll No. : B16755
Program Name : BDA
Semester : Third

Problem Release date:- 12/09/2017
Date of Submission :- 23/10/2017

1. Introduction

1.1 Dataset Description

The given dataset is the ModApte version of the Renters-21578 data set. It is new wire of 1987. The documents were originally assembled and ordered with categories by Carnegi Group Inc. and Returns Ltd. In this assignment we will consider only the following 10 categories of the ModApte version of Returns-21578 corps: [1]
alum, barley, coffee, dmk, fuel, livestock, palm-oil, retail, soybean, veg-oil

The Dataset given, consists two parts train and test. The Train set and the Test set contains 499 and 185 documents respectively, in total, from all the 10 classes.

2. Classification Methods

In our following text classification task we will use the following four types of classifiers:
1.Naive Bayes 2.Logistic Regression 3.Support Vector Machines 4.Multi Layer Perceptron

2.1 Naive Bayes

As the name suggests it is based on the Bayes' Theorem. Bayes' Theorem provides a way that we can calculate the probability of a hypothesis given our prior knowledge. [2]
It is called naive Bayes or idiot Bayes because the calculation of the probabilities for each hypothesis are simplified to make their calculation tractable. Rather than attempting to calculate the values of each attribute value, they are assumed to be conditionally independent given the target value and calculated as simple products. This is a very strong assumption that is most unlikely in real data, i.e. that the attributes do not interact.
We will be using Python's Scikit-Learn through out. [3].
We will run Naive Bayes with Multinomial Algorithm mostly. Since the rest Bernoulli and Gaussian assumes a distribution of the dataset, and Multinomial Naive Bayes' assumes independent classes which is true in this case. More over Bernoulli algorithm takes into account only if a word occurs in the document or not rather than considering word frequency which is exactly the case for Multinomial Naive Bayes' algorithm. Also for same reason I did not use GaussianNB. And historically Multinomial Naive Bayes has worked better on text classification(the references to this are give in the footnote [4]) Also we will show this by result (see Appendix).
The parameter to be tuned here case is the value of alpha,for which we will use Grid Search.

[1] For further references regarding the dataset see UCI Repository Reuters Dataset Collection
[2] For further references regarding Naive Bayes' classifiers I used A Comparison of Event Models for Naive Bayes Text Classification
[3] For further references regarding Scikit-Learn's Naive Bayes' algorithms see Scikit-Learn Naive Bayes'
[4] Reference for using Multinomial above the two othersSome Effective Techniques for Naive Bayes Text Classification Sang-Bum Kim, Kyoung-Soo Han, Hae-Chang Rim, and Sung Hyon Myaeng

The smoothing priors alpha accounts for features not present in the learning samples and prevents zero probabilities in further computations.

2.2 Logistic regression

Since this is a type of regression problem with classes being categorical we will try Logistic Regression. We know generally Logistic Regression in a binary classifier but as our problem is Multiclass(i.e. more than one class labels), so we will use Scikit-Learns Logistic Regression with parameter multi-class ='multinomial' and solver = 'lbfgs' [5]
If Y_i be the class labels and X_i be the vector of explanatory variables describing observation i, and β_k be vectors of weights, then using the fact that all K (K is the number of classes) of the probabilities must sum to one, we find
P(Y_i = K) = $\frac{1}{1+\sum_{k=1}^{K-1}\exp(\beta_k * X_i)}$,gives the probability of Y_i to be in the (K-1)-th class and for the K-th class we do 1 - P(Y_i = K). We choose the maximum of the probabilities to assign the data points to that class. [6]

2.3 Support Vector Machines

[7] SVMs are a generally applicable tool for machine learning. Suppose we are given with training examples x_i, and the target values $y_i \varepsilon$(-1,1). SVM searches for a separating hyperplane, which separates positive and negative examples from each other with maximal margin, in other words, the distance of the decision surface and the closest example is maximal. The equation of the hyperplane is : w^T*x+b = 0. For Separable Case(which is almost never) the decision rule is :

$$w^T * x + b \geq 1 \implies y_i = +1 \qquad\qquad w^T * x + b \leq 1 \implies y_i = -1$$

In case of Non-Separable Datasets we need introduce concepts of trade off between margin and training error denoted by C. And Kernels, a function K(x_i, x_j) which transforms the data point to a different space where they may be linearly separable and is called a kernel-function if it satisfies the Mercers's condition. [8] These are the two parameters will be tuning to find the best possible result. We us Scikit-Learns' SVM tool [9]

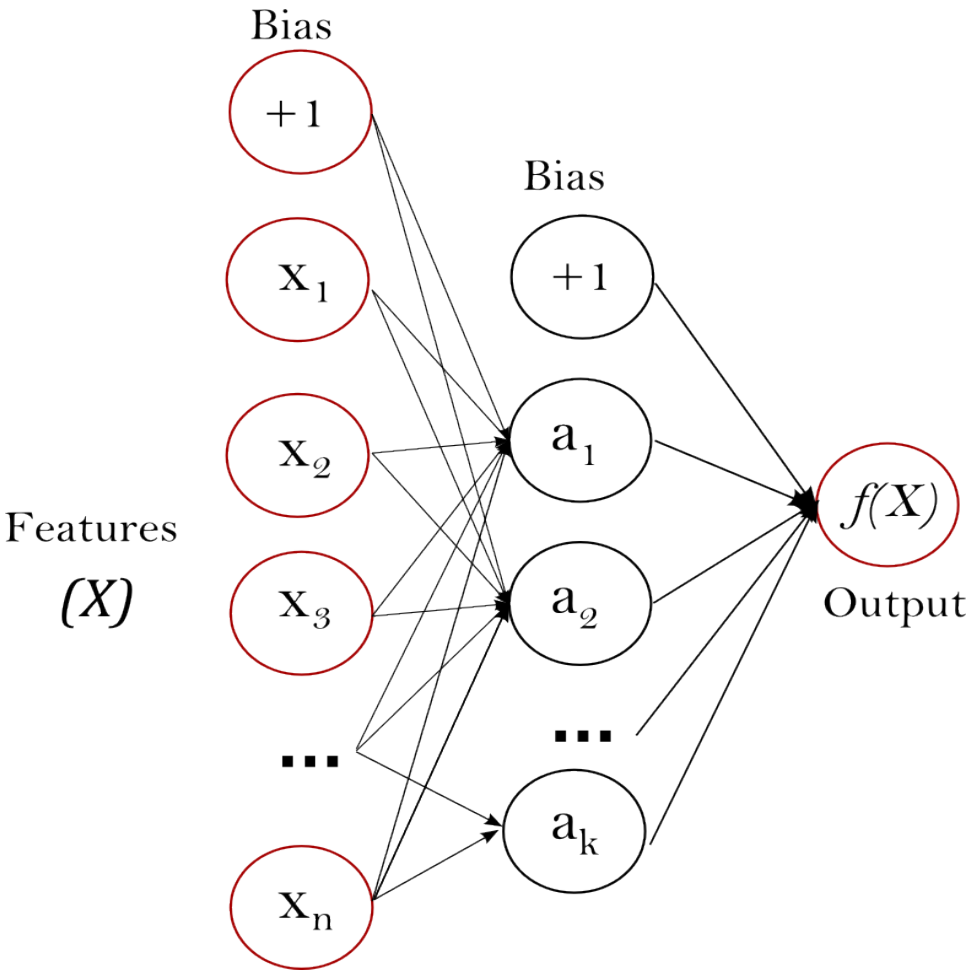
2.4 Multilayer Perceptron

Next, we look at Multilayer Perceptrons for classification and use the Scikit-Learn's Implementation. Multi-layer Perceptron (MLP) is a supervised learning algorithm that learns a function, f : $R^m \rightarrow R^o$ where m is the number of dimensions for input and o is the number

[5] For further references regarding Scikit-Learn's Logistic Regression see Scikit-Learn Logistic Regression
[6] For further references regarding MultiClass Logistic Regression I referred to Wikipedia Multinomial Logistic Regression
[7] For further references regarding Support Vector Machines Support Vector Machines Theory
[8] For further references regarding Mercers Theorem refer to Mercers Theorem
[9] I used Scikit-Learns SVM Support Vector Machines

of dimensions of the output.

A one hidden layer MLP with scalar output.



The leftmost layer, known as the input layer, consists of a set of neurons $(x_i|x_1,x_2,...,x_m)$ representing the input features. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation $w_1x_1+w_2x_2+....+w_mx_m$ followed by a non-linear activation function $g : R \rightarrow R$ like hyperbolic tan function(tanh) or logistic etc known as the activation function. In this case we tune the parameters that is the number of hidden layers, the activation function and the penalty parameter. [10]

[10]Scikit Learns MLP Scikit-Learn MLP Documentation

3. Evaluation Criteria

We use Confusion Matrix,Precision, Recall and the F-measure, for evaluation of the binary classifiers. Confusion matrix:

		Prediction outcome	
		yes	**no**
Actual value	**yes**	True Positive	False Negative
	no	False Positive	True Negative

The metrics and their formulae :

Metric	Formula
Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
F-Measure	$\frac{2*Precision*Recall}{Precision+Recall}$

where, TP means true positive, TN true negative, FN false negative, FP false positive instances, defined above in the confusion matrix.

I will be using micro average to calculate these metrics since,micro-averaged measures add all the tp, fp and fn (for each label), whereafter a new binary evaluation is made. Macro-averaged measures add all the measures (Precision, Recall, or F-Measure) and divide with the number of labels, which is more like an average.[11]

4. Data Pre-processing

Before starting to use classifiers we need to process the given Dataset to a structure which can be passed as input to the classifiers. And also to clean the data so that there are no meaningless features passed onto as input. For this purpose I did the following step by step.
STEP 1 : Tokenized each document of the entire dataset(both test and train).
STEP 2 : Removed the Punctuations because they give away nothing about the class of the

[11] For more information on Micro and macro average I referred these Evaluation of text classification

document in which they are contained.
STEP 3 : Removed the Stop-words (e.g. a, an, the, were, with, will etc), again for the same reasons.
STEP 4 : Then Stemmed the word tokens using PorterStemmer[12] from the NLTK Stem package[13] in Python. Stemming is the process of reducing a word into its stem, i.e. its root form. The root form is not necessarily a word by itself, but it can be used to generate words by concatenating the right suffix. For example, the words fish, fishes and fishing all stem into fish, which is a correct word
STEP 5 : Next, converted the collection of text documents to a matrix of token counts using sklearn.feature extraction.text.CountVectorizer [14] from python. What this does is count the occurrences of a stemmed word in a document . Basically, we define the term-frequency as a couting function: tf(t,d) = $\sum_{x \varepsilon d} fr(x,t)$, where

$$fr(x,t) = \begin{cases} 1 & if, x = t \\ 0 & otherwise \end{cases} \tag{4.1}$$

And then goes on into the creation of the document vector, which is represented by:$\overrightarrow{v_{d_n}}$ = $(tf(t_1,d_n), tf(t_2,d_n), ..., tf(t_n,d_n)$, where n is the number of documents.
Then deleted those feature columns which were all zero among the train documents. Because even though they may have been features of the test they not being present among the train cannot be considered as features as they are supposed to be unknown to us. There were 834 such terms for example : 'miti', 'mitsubishi', 'mitterand', 'model', 'modestli', 'modif', 'mohler', tuberculosisfre', 'tug', 'uff', 'unblock' etc.[15]
STEP 6 : But what we did in the previous step is the count the occurrences of a stemmed word in a document which did not take into account that the word itself may be common among all the documents. Hence an inverse document frequency factor is incorporated which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely..So we use tf-idf(Term frequency - Inverse document frequency) using sklearn.feature extraction.text.TfidfVectorizer[16]. The formula for the tf-idf is :tf-idf(t) = tf(t,d) $\times$ idf(t) ,where idf(t) = $\log \frac{|D|}{1+|\{d : t \in d\}|}$. Here | D | denotes the total number of documents and $|\{d : t \in d\}|$ is the number of documents where the term t appears, when the term-frequency function satisfies tf$(t,d) \neq 0$, we're only adding 1 into the formula to avoid zero-division.

[12] PorterStemmer
[13] For further references regarding Stemming I referred to Stemming
[14] CountVectorizer
[15] For further references regarding count term frequency matrix creation refer to Term Frequency Matrix
[16] For further references regarding tf-idf matrix creation by python refer to tf-idf

5. Results From The Classifiers

Here I present the best of the precision scored models on the test sets that I got for each of the classifiers, upto two decimal places, after doing grid search for the best parameters in each cases.For more of the results of models that I tried see the Appendix

Classifiers	Parameters	Precision	Recall	F-Measure
Naive-Bayes	**MultinomialNB alpha=0.01 unigram On the Term Frequency Matrix**	0.73	0.73	0.73
Logistic Regression	**C=100 unigram**	0.77	0.77	0.77
SVM	**kernel='linear' C=1 unigram**	0.79	0.79	0.79
MLP	**hidden layers=(100,100) activation function ='logistic' unigram**	0.73	0.73	0.73

Thus, we see SVM gives the best prediction on the dataset.
We see for Naive Bayes' when we take unigrams, using term-frequency matrix instead of tf-idf yields better result. But intuition ally that should not be the case. This may be happening due to availability of less number of training sample. Less number of training sample means the effect of the number of document is a bit neutralized thus tf-idf matrix doesn't give more information than the term frequency matrix.

5.1 Why Should linear kernel SVMs Work Well ?

1. When learning text classifiers, one has to deal with many (to be exact,in this case, 4615) features. Since SVMs use overfitting protection, which does not necessarily depend on the number of features, they have the better potential to handle these large feature spaces.

2.Training a SVM with a linear kernel is faster than with another kernel. Particularly when using a dedicated library such as LibLinear[17].

3.Less parameters to optimize than MLP and Naive Bayes.

4.Text data are naturally linearly separable[18] since the features i.e the words are independent of each other thus svm linear kernels which finds a separating hyperplane between the class tend to work better than others.

[17] LibLinear
[18] Why SVM linear kernels work well on text classification

6. Some Observations And Future Work

We first present the confusion matrices for the 0 best classifier on the test data given. 0,1,2,....,9 are the class labels of the documents, alum, barley, coffee, dmk, fuel, livestock, palm-oil, retail, soybean, veg-oil respectively :

Support Vector Machine with Linear Kernel:

	0	1	2	3	4	5	6	7	8	9
1	18	0	1	0	0	1	0	0	0	3
1	0	12	0	0	0	0	0	0	1	1
2	0	0	28	0	0	0	0	0	0	0
3	0	0	0	4	0	0	0	0	0	0
4	0	0	0	0	6	0	0	1	0	3
5	0	0	0	0	0	23	0	0	1	0
6	0	0	1	0	0	0	3	0	0	6
7	0	0	0	0	0	0	0	2	0	0
8	0	0	1	0	0	2	0	0	25	2
9	0	0	2	0	0	1	3	0	7	24

Now the Accuracy Measures per Class:

	Precision	Recall	F-Measure
alum	1.00	0.78	0.88
barley	1	0.86	0.96
coffee	0.85	1.0	0.92
dmk	1.0	1.0	1.0
fuel	1	0.60	0.75
livestock	0.85	0.96	0.90
palm-oil	0.50	0.30	0.37
retail	0.67	1.0	0.80
soybean	0.74	0.76	0.75
vegoil	0.57	0.65	0.61

Thus, we see that it is the class palm-oil which seems to be dragging us down and classified as veg-oil mostly(this is the case in each of classifier). Now lets look for a reason: the documents "009606", "009610", "009647", "0010519","0010637", "0011048","0011593", "0011721","0011820","0012526" are all the documents of both palm oil and veg oil class. The dataset is actually multi-label. Hence such miss classifications that we get are very natural.
As a future work we should consider the MultiLabel Classification.[19]. It is also implemented in python scikit-learn[20].

[19] Multilabel text classification via label propagation
[20] Multilabel classification Implementation in python

7. Appendix

Classifiers	Parameters	Precision	Recall	F-Measure
Naive-Bayes	**MultinomialNB alpha=0.01 bigram On the Term Frequency Matrix**	0.66	0.66	0.66
Naive-Bayes	**MultinomialNB alpha=0.01 bigram On the Tf-IDF Matrix**	0.67	0.67	0.67
Naive-Bayes	**BernoulliNB alpha=0.01 unigram On the Term Frequency Matrix**	0.62	0.62	0.62
Logistic Regression	**C=100 bigram**	0.75	0.75	0.75
Logistic Regression	**C=100 trigram**	0.75	0.75	0.75
SVM	**kernel='linear' C=1 unigram**	0.79	0.79	0.79
MLP	**hidden layers=(100,100) activation function ='logistic' unigram**	0.73	0.73	0.73
MLP	**hidden layers=(100,100,100) activation function ='relu' unigram**	0.72	0.72	0.72
MLP	**hidden layers=(60,60,60) activation function ='relu' unigram**	0.72	0.72	0.72

8. References

1. For references see the footnotes.
2.Text Categorization and Support Vector Machines István Pilászy Department of Measurement and Information Systems, Budapest University of Technology and Economics, e-mail: pila@mit.bme.hu .
3.Text Categorization with Support Vector Machines: Learning with Many Relevant Features, Thorsten Joachims, University, at Dortmund Informatik LS8, Baroper Str. 301 44221 Dortmund, Germany
4. Scikit-Learn Packages as mentioned in the footnotes. 5.Towards Multi Label Text Classification through Label Propagation Shweta C. Dharmadhikari PICT Pune, India Maya Ingle DAVV Indore, India Parag Kulkarni EkLaT Solutions Pune, India