CMPE561 Application Project #2

Mine Melodi Çalışkan
2015705009
minemelodicaliskan@gmail.com

Emrah Budur
2016800036
emrah.budur@yahoo.com

17 May 2017

Introduction

In this project, we built an author identification system using a generative model (Naïve Bayes) and a discriminative model (Support Vector Machine) and evaluated its performance on “69 Authors” corpus that includes 910 articles (documents) from 69 different Turkish authors. As a result, we reported 25 % accuracy for "Naive Bayes" and 78 % accuracy for "Support Vector Machine" on the test that is obtained by randomly selection 40 % of the corpus.

Program Interface

- The **naive_bayes()** function is the main function for author identification using Naive Bayes method.
- The **svm()** function is the main function for author identification using Support Vector Machine method.

  In both functions input is the provided with the "authors" corpus, the user should change the "input_file_name" if different data set is desired.

These functions returns:

- Accuracy
- Micro-averaged Precision, Recall, F-measure
- Macro-averaged Precision, Recall, F-measure
- Confusion Matrix

based on the provided corpora.

Use of $SVM^{Multiclass}$:

- » svm_multiclass_learn [options] train_file model_file
- » svm_multiclass_classify test_file model_file prediction_file
- **Learning Options** :
  - -c float → C: trade-off between training error and margin (default 0.01)
  - -p [1,2] → L-norm to use for slack variables. Use 1 for L1-norm, use 2 for squared slacks. (default 1)
  - -o [1,2] → Rescaling method to use for loss. 1: slack rescaling 2: margin rescaling (default 2)
  - -l [0..] → Loss function to use. 0: zero/one loss
- **Kernel Options:**

- -t int → type of kernel function:
  - 0: linear (default)
  - 1: polynomial $(sa * b + c)^d$
  - 2: radial basis function $exp(-gamma||a - b||^2)$
  - 3: sigmoid $tanh(sa * b + c)$
  - 4: user defined kernel from kernel.h
- -d int → parameter d in polynomial kernel
- -g float → parameter gamma in rbf kernel
- -s float → parameter s in sigmoid/poly kernel
- -r float → parameter c in sigmoid/poly kernel
- -u string → parameter of user defined kernel

For other options see the main website of $SVM^{Multiclass}$ [1]

Program Execution

Method

Naive Bayes

The Naive Bayes classifier is a statistical classifier that proposes a Bayesian hypotheses based upon the maximum likelihood of the data supporting a specific hypothesis.

The Bayes' theorem proposes the following formula to quantify the relationship between an independent variable x and its dependent variable y [3].

$$P(y|x_1, ..., x_n) = \frac{P(y) * P(x_1, ..., x_n|y)}{P(x_1, ..., x_n)} \tag{1}$$

The formula is simplified by using the naive independence assumption such that:

$$P(y|x_1, ..., x_n) \cong \frac{P(y) \prod_{i=1}^{n} P(x_i|y)}{P(x_1, ..., x_n)} \tag{2}$$

In the classification context we choose y values that maximize Eq(2) given the set of input variables $x_1, ..., x_n$ as formulated in Eq(3).

$$\underset{y}{\text{argmax}} \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1, ..., x_n)} = \underset{y}{\text{argmax}}\, P(y)\prod_{i=1}^{n} P(x_i|y) \tag{3}$$

Note that the denominator in the maximization expression is completely ommitted since it is not a function of the maximizing term y.

There are a couple of variant of Naive Bayes methods which vary mainly on the assumption that is made to calculate *the posterior probability* $P(x_i|y)$.

Gaussian Naive Bayes

In this variant of Naive Bayes, the posterior probability is assumed to be Gaussian as formulated below.

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\Big(- \frac{(x_i - \mu_y)^2}{2\sigma_y^2}\Big) \tag{4}$$

where σ_y and μ_y are calculated by means of maximum likelihood estimation.

Bernoulli Naive Bayes

In this variant of Naive Bayes, the features are required to be binary valued. A distinct probability P(i|y) for each feature i given the dependent variable is calculated over the training dataset. Then, the posterior probability is calculated as follows.

$$P(x_i|y) = P(x_i|y) * x_i + \Big(1 - P(x_i|y)\Big) * (1 - x_i) \tag{5}$$

Note that the feature x_i contributes to the resulting posterior probability when not only its occurrence but also its non-occurrence.

Multinomial Naive Bayes

This variant is suitable for classification of text where labels are multinomials. Since we have multinomial author name labels in this application, we implemented Multinomial Naive Bayes method.

In this case, the posterior probability is calculated by means of maximum likelihood as formulated below.

$$P(x_i|y) = \frac{N_{yi}}{N_y} \tag{6}$$

where

N_{yi} = number of times word w_i appears in articles of author y during training time.

N_y = total number of words in articles of author y during training time.

Add-k Smoothing:

In Eq(6), estimation of $P(w_i|c)$ becomes a problem in test phase for each word w_i which doesn't appear in papers of author y in training dataset but exists in test dataset. This problem can be basically solved by smoothing the posterior probability as follows.

$$P(x_i|y) = \frac{N_{yi} + k}{N_y + kV} \tag{7}$$

where

k = smoothing factor.

V = the size of vocabulary in training dataset

This method is known as add-k smoothing. Although the value of k ranges from zero to infinity in theory, its typical value is 1 which is specially called as Laplace Smoothing.

Pseudo-code for Training

Algorithm 1 *Naive Bayes Algorithm*

for each document **do**
    if document.author=target **then**
        add document to corpus A
    else
        add document to corpus B
    end if
end for
Set Vocabulary=
for each Word in corpus A and B **do**
    if Word in Vocabulary **then**
        Add word to Vocabulary
    end if
end for
Set N=number of words in Vocabulary
Set Nd=Number of Documents
Using Corpus A
Set nd=number of documents in corpus
Set nw= number of distinct words in corpus
Set pc= nd /Nd
for each distinct Word in corpus **do**
    ni=number of times Word appears in corpus
    pi= $\eta \frac{(n_i+1)}{(nw+1)}$
end for
End Using
Repeat for corpus B

Pseudo-code for evaluation

```
T= Test Document
Using Corpus A
Set P=pc
for each word in Test-Document that is in Vocabulary do
    P=P*pc
end for
End Using
Repeat for Corpus B
if A≥B then choose A else choose B
end if
```

Training

We have executed two steps in training time which were explained below.

- **Splitting the dataset:** We splitted the dataset such that 60% of the articles of each author reserved in training dataset and the remaining 40% articles of each author is kept as test dataset.
- **Calculating word likelihood probabilities:** We calculated the word probabilities for each author using the following formula:

$$P(w_i|a) = \frac{N_{ai} + k}{N_a + kV} \tag{8}$$

where

N_{ai} = number of times word w_i appears in articles of author a during training time.

N_a = total number of words in articles of author a during training time.

k = smoothing factor.

V = the size of vocabulary in training dataset

We experimented various values for $0 < k < 1$. Since the dataset was balanced, we didn't take into account the prior probabilities.

Prediction

In prediction phase, we are given a paper without any information about its author and the application is expected to predict its author based on its content. Hence, in prediction phase, we use the word likelihood probabilities that were obtained in training phase as follows.

For each author a that were seen in training time, we iterate paper word by word and sum up $P(w_i|a)$ values for each word w_i. Then, the resulting sums were sorted descendingly and the author having maximum sum is returned as predicted author of the given paper.

Support Vector Machine

This is a discriminative classifier formally defined by a separating hyperplane. Given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples.

Method

$SVM^{Multiclass}$ uses the Multiclass Kernel-based Vector Machines [4], and finds the soltion of the following optimization problem during training. It is claimed in [1] that the optimization algorithm is very fast in linear case and our experiment results support this.

Let $(x_1, y_1), \ldots, (x_n, y_n)$ be a training set with labels $y_i \in \{1, \ldots, k\}$

$$\begin{aligned} \textbf{Task:} \qquad & min \frac{1}{2} \sum_{i=1}^{k} w_i * w_i + \frac{C}{n} \sum_{i=1}^{n} \xi_i \\ & \text{s.t. for all y } \in \{1, \ldots, k\} : \\ [x_1 * w_{yi}] \geq [x_1 * w_y] + 100 * \Delta(y1, y) - \xi_1 \\ & \ldots \\ [x_n * w_{yn}] \geq [x_n * w_y] + 100 * \Delta(y_n, y) - \xi_n \end{aligned} \tag{9}$$

where

- C: the usual regularization parameter that trades off margin size and training error.

- $\Delta(y_n, y)$: the loss function that returns 0 if y_n equals y, and 1 otherwise.

Training

- We trained our model with "svm_multiclass_learn" in SVM_Multiclass using our train file.

Results of SVM Multiclass model training
Final epsilon on KKT-Conditions
Upper bound on duality gap
Dual objective value: p_{val}
Total number of constraints in final working set
Number of iterations
Number of calls to find most violated constraint
Number of SV
Norm of weight vector $\|\|w\|\|$
Value of slack variable on working set, ξ
Value of slack variable on global set, ξ
Norm of longest difference vector: $\|\|\Psi(x, y) - \Psi(x, y_{bar})\|\|$
Runtime in cpu-seconds
Final number of constraint in cache

Testing (Prediction)

- For prediction we used "svm_multiclass_classify"

Results of SVM Multiclass model testing
Runtime without IO in cpu-seconds
Average loss on test set
Zero/one-error on test set
Number of correct, incorrect predictions and total number of targets

Additional evaluation results in our implementation
Confusion matrix
Accuray
Micro-averaged precision, recall, f-measure
Macro-averaged precision, recall, f-measure

Input and Output

Naive Bayes

Input

The application processes the authors dataset as a zipped archive given in the main method of our Naive Bayes implementation.

Output

The application outputs the following measurements.

- Overall success metrics that were examplified in Table 1.
- Confusion matrix which were illustrated in Figure 2 and Figure 3.

Support Vector Machine

Input

Like in Naive Bayes implementation, the application takes the input the authors dataset as a zipped archive given in the main method of our SVM implementation.

Data preprocessing

First we created a vocabulary using "set" data structure and train data set. Then we assigned unique id for each token using a loop in range of vocabulary size and adding 1 to each term.

Similary we assigned unique id to each author. Then we created a modified version of the training file corresponding the following form using dict of dict data structure:

authorid f1:v1 f2:v2 ...

In input file every line represents an article. Authorid corresponds to the author who wrote that article. f_i and v_i pairs corresponds to word id's and their binary valued weights.0 weights are skipped and feature ids are ordered.

Additional Features

In SVM classification, new features are added with $f_n + i$ for $i = 1, 2, \ldots, m$ where m is the number of new features, f_n is the greatest id value of the tokens.

We used following 4 new features:

- Article length
- Average sentence length in an article
- Number of three dots in an article
- Number of semicolons in an article

Then we sorted features with respect to their id's using "OrderedDict" in python's "collections" module.

```
1 1:1 73:1 76:1 239:1 411:1 464:1 658:1 677:1 716:1 807:1 1025:1
1054:1 1138:1 1336:1 1447:1 1457:1 1548:1 1941:1 2104:1 2184:1
2214:1 2250:1 2346:1 2741:1 3026:1 3431:1 3618:1 3911:1 3929:1
4065:1 4515:1 4787:1 5299:1 5353:1 5457:1 5489:1 5611:1 5777:1
6001:1 6193:1 6203:1 6211:1 6217:1 6270:1 6293:1 6580:1 7060:1
7408:1 7886:1 8042:1 8043:1 8122:1 8193:1 8319:1 8947:1 8951:1
8976:1 9246:1 9308:1 9385:1 9564:1 10087:1 10344:1 10358:1
10445:1 10592:1 11023:1 11141:1 11221:1 11326:1 11440:1 11525:1
11632:1 11782:1 11963:1 12439:1 12634:1 12800:1 12852:1 12943:1
13076:1 13562:1 13638:1 13738:1 13785:1 14117:1 14664:1 14727:1
14979:1 15229:1 15394:1 15611:1 15808:1 16001:1 16084:1 16253:1
16325:1 16502:1 16515:1 16542:1 16626:1 16633:1 16992:1 17500:1
17613:1 17712:1 17717:1 17951:1 18013:1 18064:1 18075:1 18251:1
18317:1 18432:1 18565:1 18651:1 19033:1 19111:1 19647:1 19752:1
19753:1 19906:1 19976:1 20069:1 20386:1 20478:1 20605:1 20852:1
21190:1 21191:1 21194:1 21278:1 21336:1 21642:1 21950:1 22227:1
22437:1 22509:1 22656:1 22925:1 23060:1 23496:1 23651:1 23997:1
24126:1 24177:1 24305:1 24534:1 24538:1 24556:1 24724:1 24788:1
25010:1 25127:1 25489:1 25535:1 25655:1 25829:1 25900:1 26177:1
26185:1 26235:1 26250:1 26339:1 26411:1 26553:1 26716:1 26784:1
26970:1 27023:1 27120:1 27145:1 27152:1 27193:1 27242:1 27562:1
27854:1 27919:1 28375:1 28529:1 28611:1 28658:1 28674:1 28710:1
28722:1 28912:1 29101:1 29261:1 29280:1 29317:1 29382:1 29512:1
29522:1 29527:1 29562:1 29777:1 29976:1 29978:1 29987:1 30075:1
30109:1 30241:1 30538:1 31120:1 31168:1 31465:1 31535:1 31648:1
32170:1 32236:1 32375:1 32629:1 32639:1 32664:1 32774:1 32820:1
33036:1 33075:1 33154:1 33413:1 33496:1 33750:1 33821:1 33869:1
```

Figure 1: The snapshot of training file in word wrapped form

Output

The program saved the prediction file, containing predicted authors, to user specified location and returns the explained results in the Method section of SVM.

Program Structure

Packages

zipfile, math,re, random, sys, collections, codecs, $SVM^{Multiclass}$

Evaluation

Below are the details of the metrics we calculated for evaluating the output quality of the implementations.

Accuracy: The number of correct predictions divided by the number of documents.

TP (True Positive): Number of documents whose author is A and predicted as A

FN (False Negative): Number of documents whose author is A but predicted as another author

FP (False Positive): Number of documents whose author is another author but predicted as A

Let π: Precision, ρ: Recall , F: F-measure

Micro-averaged Precision, Recall, F-measure

$$\begin{aligned} \pi &= \frac{TP}{TP+FP} = \frac{\sum_i TP_i}{\sum_i (TP_i + FP_i)} \\ \rho &= \frac{TP}{TP+FN} = \frac{\sum_i TP_i}{TP_i + FN_i} \\ F &= \frac{2 * \pi * \rho}{\pi + \rho} \end{aligned} \tag{10}$$

Macro-averaged Precision, Recall, F-measure

$$\begin{aligned} \pi_i &= \frac{TP_i}{TP_i + FP_i} \\ \rho_i &= \frac{TP_i}{TP_i + FN_i} \\ F_i &= \frac{2 * \pi * \rho}{\pi + \rho} \\ \pi &= \frac{\sum_i \pi_i}{M} \rho \qquad = \frac{\sum_i \rho_i}{M} F = \frac{\sum_i F_i}{M} \end{aligned} \tag{11}$$

k	0	1	2	3	4	5	5.25	5.5	5.75	6	7	8	9	10
accuracy	0.151	0.184	0.214	0.231	0.239	0.250	0.250	**0.253**	0.247	0.247	0.234	0.228	0.228	0.220
micro_avg_precision	0.151	0.184	0.214	0.231	0.239	0.250	0.250	**0.253**	0.247	0.247	0.234	0.228	0.228	0.220
micro_avg_recall	0.151	0.184	0.214	0.231	0.239	0.250	0.250	**0.253**	0.247	0.247	0.234	0.228	0.228	0.220
micro_avg_f_measure	0.151	0.184	0.214	0.231	0.239	0.250	0.250	**0.253**	0.247	0.247	0.234	0.228	0.228	0.220
macro_avg_precision	0.154	0.176	0.181	0.223	0.237	0.261	0.260	**0.260**	0.261	0.260	0.296	0.279	0.270	0.284
macro_avg_recall	0.135	0.165	0.190	0.210	0.226	0.240	0.240	**0.246**	0.244	0.244	0.242	0.239	0.246	0.239
macro_avg_f_measure	0.105	0.127	0.145	0.168	0.182	0.197	0.196	**0.200**	0.198	0.197	0.195	0.195	0.201	0.204

Table 1: The experimentation over different values of k

Results

Naive Bayes Results

Table 1 shows the results for different values of k that is used in add-k smoothing. A fine grained grid search where 4<k<6 let us find a maximum value for $k = 5.5$ as highlighted in Table 1. In addition, by using the best k value (k=5.5), we obtained a confusion matrix for all authors which can be analyzed in Figure 2 and Figure 3. Note that micro averaged precision, recall and f-measures are identical since this is a multimodel classification of a balanced dataset.

We obtained also author based success metrics whose head is illustrated in Figure 4 (We obtained the same table also for our SVM implementation but we didn't report it for brevity). In this table, it can be observed that predicting papers of some particular authors is way more easy for the application than the other. For example, the application identified the papers of Abbas Guclu with a 100% success rates. On the other hand, it was confused when predicting the papers of Ayse Arman.

	abbasGuclu	abdullahAymaz	ahmetAltan	ahmetHakan	aliBulac	atillaDorsay	ayseArman	balcicekPamir	bekirCoskun	bulent
abbasGuclu	6.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
abdullahAymaz	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ahmetAltan	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
aliBulac	0.0	1.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0
atillaDorsay	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0
bulentKorucu	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
cemSuer	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0
cemilErtem	0.0	0.0	0.0	1.0	0.0	2.0	1.0	0.0	1.0	2.0
cengizCandar	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
cetinAltan	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ekremDumanli	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
elifSafak	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0
emreKongar	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ertugrulOzkok	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
fatihAltayli	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
fikretBila	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
hasanPulur	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
hasmetBabaoglu	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 2: Some part of the confusion matrix which was obtained as a result of Naive Bayes implementation

Support Vector Machine Results

Figure 5 is a picture of confusion matrix. Most of the predictions were mapped into diagonal elements, which indicate correct predictions, while few elements were mapped into the non-diagonal elements which means incorrect predictions.

Table 3 shows the overall success metrics while Table 2 denotes the runtime performance of our SVM implementation.

Improvements and Extensions

Naive Bayes

Improvements: Designate special solution for misspredictions

In Figure 3, we noted a couple of vertical lines that belong to miss predictions of some authors. We may deeply analyze the data to find the root cause of these mispredictions and designate a special solution for it.

Extension: Alternative smoothing methods

We have implemented add-k smoothing since it is easy to implement and quick to test. We may extend our smoothing strategy by implementing alternative method such as GT smoothing.

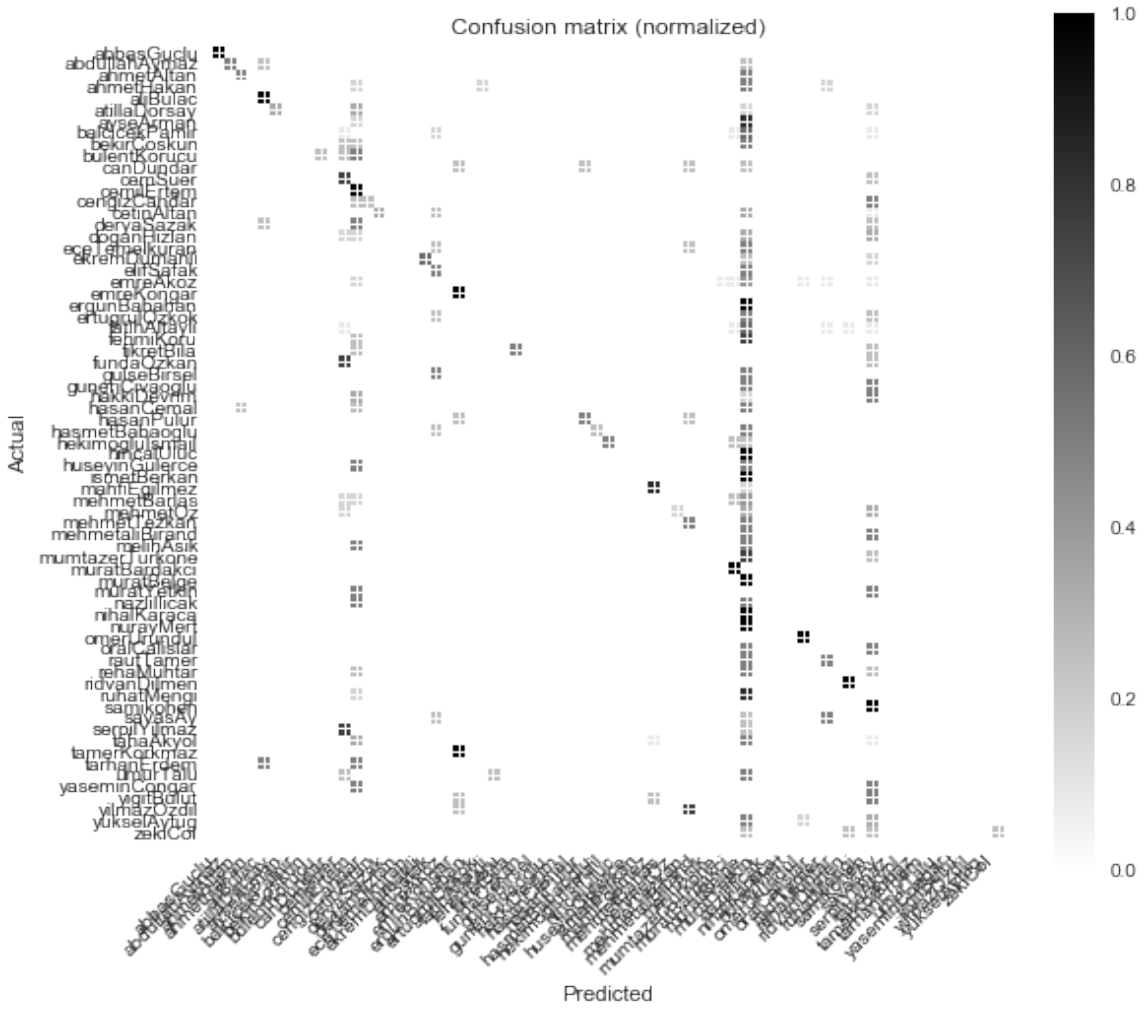

Figure 3: Normalized confusion matrix which was obtained as a result of Naive Bayes implementation. The diagonal elements denotes correct prediction while other elements indicate miss-predictions. The matrix also shows that a couple of authors are frequently miss predicted in lieu of other authors which can be easily inferred by looking at the vertical lines of elements.

Support Vector Machine

Kernels Options

Sigmoid and Polynomial kernels are available, however it's stated [1] that their performances are not promising.

Optimization Options

- -w choice of structural learning algorithm
- -e: (epsilon) allow that tolerance for termination criterion

	1_tp	2_fp	3_fn	4_prediction_count	5_accuracy	6_precision	7_recall	8_f_measure
abbasGuclu	6.0	0.0	0.0	6.0	1.000000	1.000000	1.000000	1.000000
abdullahAymaz	4.0	0.0	0.0	4.0	1.000000	1.000000	1.000000	1.000000
ahmetAltan	4.0	1.0	0.0	4.0	1.000000	0.800000	1.000000	0.888889
ahmetHakan	6.0	0.0	0.0	6.0	1.000000	1.000000	1.000000	1.000000
aliBulac	4.0	0.0	0.0	4.0	1.000000	1.000000	1.000000	1.000000
atillaDorsay	5.0	0.0	1.0	6.0	0.833333	1.000000	0.833333	0.909091
ayseArman	4.0	3.0	2.0	6.0	0.666667	0.571429	0.666667	0.615385
balcicekPamir	12.0	12.0	0.0	12.0	1.000000	0.500000	1.000000	0.666667
bekirCoskun	4.0	1.0	0.0	4.0	1.000000	0.800000	1.000000	0.888889
bulentKorucu	4.0	0.0	0.0	4.0	1.000000	1.000000	1.000000	1.000000
canDundar	2.0	0.0	2.0	4.0	0.500000	1.000000	0.500000	0.666667

Figure 4: Author based success metrics

- -h: number of svm-light iterations a variable needs to be optimal before considered for shrinking
- -k: number of new constraints to accumulate before recomputing the QP solution
- -f: number of constraints to cache for each example
- -b: percentage of training set for which to refresh cache when no epsilon violated constraint can be constructed from current cache
- -n: number of new variables entering the working set in each svm-light iteration (default n = q). Set n $<$ q to prevent zig-zagging.
- -#: terminate svm-light QP subproblem optimization, if no progress after this number of iterations.
- -m: size of svm-light cache for kernel evaluations in MB

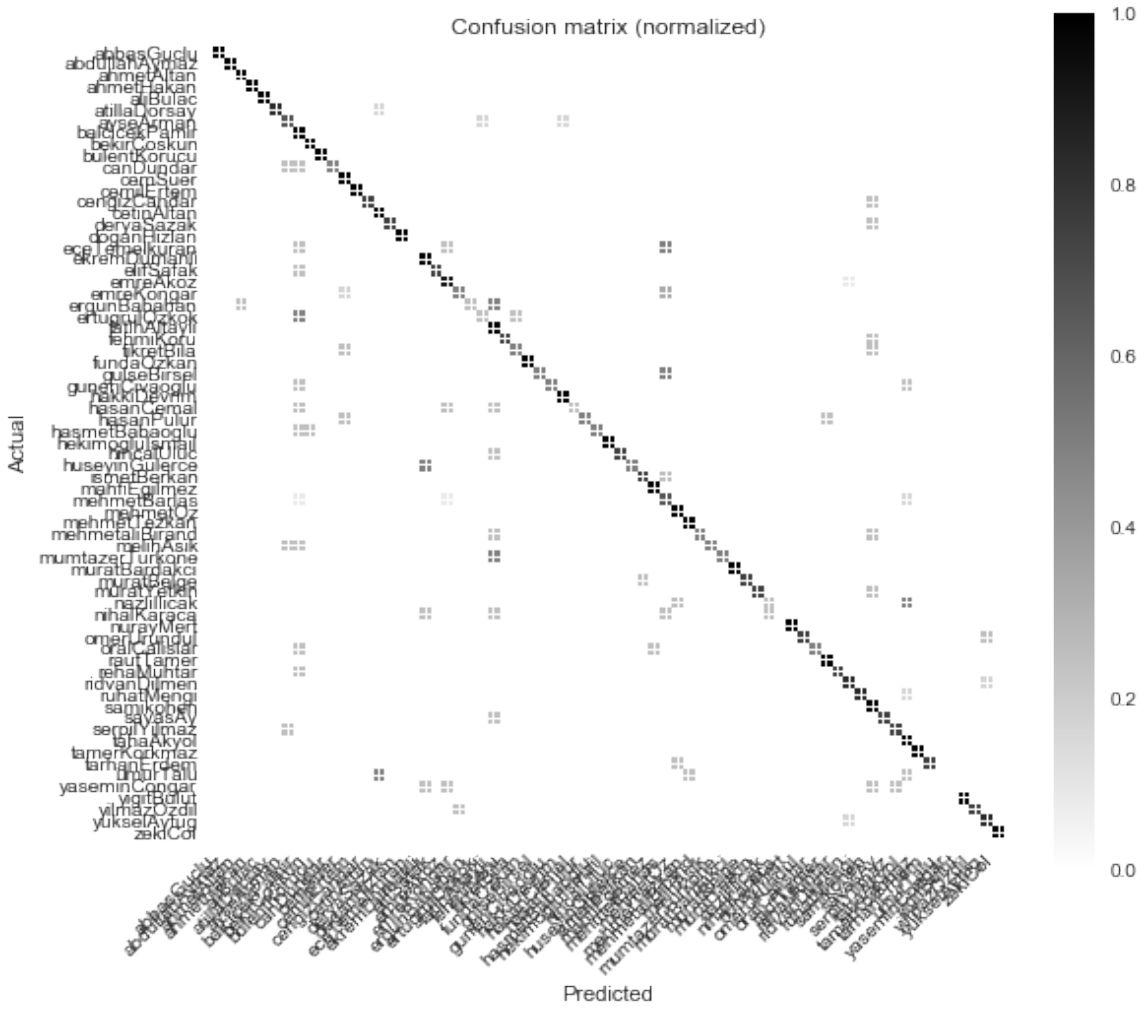

Figure 5: Normalized confusion matrix which was obtained as a result of SVM implementation. The diagonal elements denotes correct prediction while other elements indicate miss-predictions. The matrix is an overview that shows the overall performance of SVM.

Difficulties Encountered

Naive Bayes

Problem: Sparse Data

In Naive Bayes implementation, the words that didn't appear in test file but training files had zero posterior probabilities.

Applied Solution: In order to assign a value for these words, we smoothed the posterior probabilities of the words that were seen in training data by using add-k smoothing.

For c:10.0, t:0 o:1 results of trained model with additional features	
Final epsilon on KKT-Conditions	0.09593
Upper bound on duality gap	0.95921
Dual objective value	$p_val = 1.67227$
Total number of constraints in final working set	629 of 969
Number of iterations	970
Number of calls to find most violated constraint	81900
Number of SV	595
Norm of weight vector	$\|\|w\|\| = 1.19415$
Value of slack variable on working set	$\xi = 0.02542$
Value of slack variable on global set	$\xi = 0.09593$
Norm of longest difference vector	$\|\|\Psi(x,y) - \Psi(x,y_{bar})\|\| = 35855.61647$
Runtime in cpu-seconds	1176,80
Final number of constraint in cache	2730

For c:10.0, t:0 o:1 results of tested model with additional features	
Runtime without IO in cpu-seconds	0.09
Average loss on test set	21.4286
Zero/one-error on test set	21.43 %
Number of correct predictions	286
Number of incorrect predictions	78
Total number of targets	364

Table 2: Additional results regarding runtime performance of SVM

Problem: Imbalanced feature of dataset

The author-based vocabulary size differ significantly from author to author. And there are some words which were not seen in training data but in test data. We called these word as *missing words*. We had an option to smooth the probability of the *missing words*. However, the add-k smoothed probability of the same word used by different authors had significantly different smoothed values since the vocabulary sizes of each author differ from each other significantly. When we smoothed the probability of missing words, we obtain a very bad results which was on the order of 10^{-1}.

Overall success metrics	
accuracy	**0.786**
micro averaged precision	0.786
micro averaged recall	0.786
micro averaged f measure	0.786
macro averaged precision	**0.830**
macro averaged recall	0.740
macro averaged f measure	0.755

Table 3: The overall success metrics for SVM implementation

Applied Solution: We ignored the missing words and used only the add-k smoothed posterior probability of the words that were seen in training dataset.

Support Vector Machine

Problem: Regularization Parameter (c)

The C parameter trades off misclassification of training examples against simplicity of the decision surface. A low C makes the decision surface smooth, while a high C aims at classifying all training examples correctly by giving the model freedom to select more samples as support vectors.

As C increasing accuracy increases until some point and program gets slower.

Problem: Kernel (t)

We tried linear(t:0),and radial(t:2) kernels. Our observations shows that accuracy is higher for linear kernel, lower for radial kernel.

Problem: Rescaling (o)

There are two methods to use for loss.We observed that slack rescaling increases the accuracy.

- slack rescaling (o:1)
- margin rescaling (o:2)

Some results of our experiments						
c	kernel	o	new features	Accuray	Iterations	CPU- Seconds (training time)
1.0	linear	2	No	62	40	7.10
10.0	linear	2	No	**66,21**	124	39.56
1.0	radial	2	No	1.92	3	9.55
1.0	linear	1	S	**75,82**	389	186,63
100.0	linear	2	S	**69,23**	359	256,51
1.0	linear	1	S, '...' ,Avg(sent), len(art)	**78.57**	970	1105.54
10.0	linear	1	S, '...' ,Avg(sent), len(art)	**78,57**	970	1118,70
10.0	linear	2	S, '...' ,Avg(sent), len(art)	66,21	663	625,11
100.0	linear	1	S, '...' ,Avg(sent), len(art)	**78.57**	970	1119.34

Table 4: The results of SVM experiments

Table 4 shows the results of our experiments where:

- **S** : Semicolon count
- **'...'** : Three Dots Count
- **Avg(sent)** : Avg sentence length
- **len(art)**: length of article

Conclusion

In our Naive Bayes implementation, we obtained a baseline accuracy of 0.151%. We also implemented add-k smoothing method to solve the sparse data problem. Then, we improved our accuracy result up to 0.253% by fine tuning the value of k. We improved the SVM test score from 62 % to 78 % using additional features, linear kernel and slack rescaling together with the regularizaton parameter value 1.0.

A Source Code

Divide Corpus into Train file and Test file

```
1  #split dataset into train/test files
2  import zipfile
3  import random
4  random.seed(9001)
5
6  def split_files(input_filename):
7      archive = zipfile.ZipFile(input_filename, 'r')
8
9      author_info = {}
10     for filename in archive.namelist():
11         author, paper = filename.split('/')
12         author_info.setdefault(author, {})
13         author_info[author].setdefault('papers', [])
14         author_info[author]['papers'].append(paper.decode('utf8'))
15
16     author_info_train = {}
17     author_info_test = {}
18     for author in author_info:
19         author_info_train.setdefault(author, {})
20         author_info_test.setdefault(author, {})
21
22         #we disabled shuffling to observe the relative improvements in our
               controlled experiments.
23         #random.shuffle(author_info[author]['papers'])
24         num_of_papers = len(author_info[author]['papers'])
25         slice_index = int(num_of_papers*0.60)
26
27         papers_train = author_info[author]['papers'][0:slice_index]
28         papers_test = author_info[author]['papers'][slice_index:]
29
```

```
30        papers_train = [archive.read(author+'/'+filename).decode('ISO-8859-9
              ').lower() for filename in papers_train]
31        papers_test = [archive.read(author+'/'+filename).decode('ISO-8859-9'
              ).lower() for filename in papers_test]
32
33        papers_vocab_train = [set(tokenize(paper)) for paper in papers_train
              ]
34        papers_vocab_test = [set(tokenize(paper)) for paper in papers_test]
35
36        papers_train = [{'raw': paper_raw, 'vocab': paper_vocab} for
              paper_raw, paper_vocab in zip(papers_train, papers_vocab_train)]
37        papers_test = [{'raw': paper_raw, 'vocab': paper_vocab} for
              paper_raw, paper_vocab in zip(papers_test, papers_vocab_test)]
38
39        author_info_train[author]['papers'] = papers_train
40        author_info_test[author]['papers'] = papers_test
41
42    return author_info_train, author_info_test
```

Tokenize

```
1  def tokenize(text):
2      import re
3      return re.split(u'␣|#|,|/|\'|-|"|;|\.|\?|\r|\n|\t|<|>',text)
```

Evaluation

```
1  #evaluation metrics
2  import pandas as pd
3
4  def calculate_accuracy(TP, prediction_count):
5      return float(TP) / float(prediction_count)
6
7  def calculate_precision(TP, FP):
8      if (TP + FP) == 0:
9          return 0.0
```

```
10
11      return float(TP) /float(TP+ FP)
12
13  def calculate_recall(TP,FN):
14      if (TP + FN) == 0:
15          return 0.0
16
17      return float(TP) / float(TP +FN)
18
19  def calculate_f_measure(TP, FP, FN):
20      pi = calculate_precision(TP, FP)
21      rho = calculate_recall(TP, FN)
22      if pi + rho == 0:
23          return 0
24      return 2 * (float(pi) * float(rho)) / (float(pi) + float(rho))
25
26  def evaluate_predictions(author_based_success_metrics):
27      overall_success_metrics = {}
28
29      for author in author_based_success_metrics:
30          overall_success_metrics.setdefault('true_positive_count', 0.0)
31          overall_success_metrics.setdefault('false_positive_count', 0.0)
32          overall_success_metrics.setdefault('false_negative_count', 0.0)
33          overall_success_metrics.setdefault('prediction_count', 0.0)
34
35      for author in author_based_success_metrics:
36          accuracy = calculate_accuracy(author_based_success_metrics[author]['
                tp'], author_based_success_metrics[author]['prediction_count'])
37          precision = calculate_precision(author_based_success_metrics[author
                ]['tp'], author_based_success_metrics[author]['fp'])
38          recall = calculate_recall(author_based_success_metrics[author]['tp'
                ], author_based_success_metrics[author]['fn'])
39          f_measure = calculate_f_measure(author_based_success_metrics[author
                ]['tp'], author_based_success_metrics[author]['fp'],
```

```
            author_based_success_metrics[author]['fn'])
40          author_based_success_metrics[author]['accuracy'] = accuracy
41        author_based_success_metrics[author]['precision'] = precision
42        author_based_success_metrics[author]['recall'] = recall
43        author_based_success_metrics[author]['f_measure'] = f_measure
44       overall_success_metrics['true_positive_count'] +=
             author_based_success_metrics[author]['tp']
45        overall_success_metrics['false_positive_count'] +=
             author_based_success_metrics[author]['fp']
46        overall_success_metrics['false_negative_count'] +=
             author_based_success_metrics[author]['fn']
47        overall_success_metrics['prediction_count'] +=
             author_based_success_metrics[author]['prediction_count']
48
49    accuracy = calculate_accuracy(overall_success_metrics['
          true_positive_count'], overall_success_metrics['prediction_count'])
50    micro_averaged_precision = calculate_precision(overall_success_metrics[
          'true_positive_count'], overall_success_metrics['
          false_positive_count'])
51    micro_averaged_recall = calculate_recall(overall_success_metrics['
          true_positive_count'], overall_success_metrics['false_negative_count
          '])
52    micro_averaged_f_measure = calculate_f_measure (overall_success_metrics
          ['true_positive_count'], overall_success_metrics['
          false_positive_count'], overall_success_metrics['
          false_negative_count'])
53     overall_success_metrics['micro_averaged_precision'] =
           micro_averaged_precision
54    overall_success_metrics['micro_averaged_recall'] =
          micro_averaged_recall
55    overall_success_metrics['micro_averaged_f_measure'] =
          micro_averaged_f_measure
56
57    print 'overall␣success␣metrics'
```

```
58      print('accuracy␣%.3f' % accuracy)
59      print('micro_averaged_precision␣%.3f' % micro_averaged_precision)
60      print('micro_averaged_recall␣%.3f' % micro_averaged_recall)
61      print('micro_averaged_f_measure␣%.3f'% micro_averaged_f_measure)
62
63      precision_sum = 0.0
64      recall_sum = 0.0
65      f_measure_sum = 0.0
66
67      for author in author_based_success_metrics:
68          precision_sum += author_based_success_metrics[author]['precision']
69          recall_sum += author_based_success_metrics[author]['recall']
70          f_measure_sum += author_based_success_metrics[author]['f_measure']
71
72      author_count = len(author_based_success_metrics)
73
74      macro_averaged_precision = precision_sum / author_count
75      macro_averaged_recall = recall_sum / author_count
76      macro_averaged_f_measure = f_measure_sum / author_count
77       overall_success_metrics['micro_averaged_precision'] =
             micro_averaged_precision
78      overall_success_metrics['micro_averaged_recall'] =
            micro_averaged_recall
79      overall_success_metrics['micro_averaged_f_measure'] =
            micro_averaged_f_measure
80
81      print('macro_averaged_precision␣%.3f' % macro_averaged_precision)
82      print('macro_averaged_recall␣%.3f' % macro_averaged_recall)
83      print('macro_averaged_f_measure␣%.3f'% macro_averaged_f_measure)
84
85      return author_based_success_metrics, overall_success_metrics
86
87  def print_success_metrics(author_based_success_metrics,
        overall_success_metrics):
```

```
88    success_metrics = {}
89
90    for author in author_based_success_metrics:
91        success_metrics[author] = {}
92        success_metrics[author]['1_tp'] = author_based_success_metrics[
              author]['tp']
93        success_metrics[author]['2_fp'] = author_based_success_metrics[
              author]['fp']
94        success_metrics[author]['3_fn'] = author_based_success_metrics[
              author]['fn']
95        success_metrics[author]['4_prediction_count'] =
              author_based_success_metrics[author]['prediction_count']
96
97        success_metrics[author]['5_accuracy'] = author_based_success_metrics
              [author]['accuracy']
98        success_metrics[author]['6_precision'] =
              author_based_success_metrics[author]['precision']
99        success_metrics[author]['7_recall'] = author_based_success_metrics[
              author]['recall']
100       success_metrics[author]['8_f_measure'] =
              author_based_success_metrics[author]['f_measure']
101     author_based_success_metrics_dataframe= pd.DataFrame.from_dict(
            success_metrics)
102   author_based_success_metrics_dataframe=
          author_based_success_metrics_dataframe.copy()
103   author_based_success_metrics_dataframe=
          author_based_success_metrics_dataframe.fillna(0.0)
104
105   return author_based_success_metrics_dataframe.transpose(),
          overall_success_metrics
```

Naive Bayes

Train

```
1 def naive_bayes_train(author_info_train):
```

```
    import string
    for author in author_info_train:
        total_word_count = 0.0
        author_info_train[author]['word_freqs'] = {}
        author_info_train[author]['unique_words'] = set()

        for paper in author_info_train[author]['papers']:
            author_info_train[author]['word_counts'] = {}
            words = tokenize(paper)
            for word in words:
                if len(word) == 0:
                    continue
                word = word.strip()
                author_info_train[author]['unique_words'].add(word)
                author_info_train[author]['word_freqs'].setdefault(word, 0.0)
                author_info_train[author]['word_freqs'][word] =
                    author_info_train[author]['word_freqs'][word] + 1.0
                total_word_count = total_word_count + 1.0
        author_info_train[author]['total_word_count'] = total_word_count

        author_info_train[author]['word_probs'] = {}
        vocab_size = len(author_info_train[author]['unique_words'])
        author_info_train[author]['vocab_size'] = vocab_size
        for word in author_info_train[author]['unique_words']:
            prob = (author_info_train[author]['word_freqs'][word] + add_k)/
                (total_word_count + add_k*vocab_size)
            author_info_train[author]['word_probs'][word] = prob

    return author_info_train
```

Predict

```
import math
#import pandas_ml

def naive_bayes_predict_author(author_info_train, paper):
```

```
5     import string
6     words = tokenize(paper)
7     prediction_probs = {}
8     for author in author_info_train:
9         sum_prob = 0.0
10        for word in words:
11            word = word.strip()
12            if len(word) == 0:
13                continue
14
15            #The vocab sizes of each author is imbalanced. Therefore,
                  smoothing missing word reduces the success rates
                  significantly. Hence, we ignored the probability of missing
                  words.
16            word_prob = add_k/author_info_train[author]['total_word_count']
17
18            if word in author_info_train[author]['word_probs']:
19                word_prob = author_info_train[author]['word_probs'][word]
20                sum_prob = sum_prob + word_prob
21
22        prediction_probs[author]=sum_prob
23    predicted_author = max(prediction_probs.iterkeys(), key=lambda
          author_key: prediction_probs[author_key])
24    return predicted_author, prediction_probs[predicted_author]
25
26
27 def naive_bayes_test(author_info_train, author_info_test):
28    author_based_success_metrics = {}
29    for author in author_info_test:
30        author_based_success_metrics.setdefault(author, {})
31        author_based_success_metrics[author].setdefault('tp', 0)
32        author_based_success_metrics[author].setdefault('fp', 0)
33        author_based_success_metrics[author].setdefault('fn', 0)
34
```

```
35    confusion_matrix = {}
36    true_authors = []
37    predicted_authors = []
38    for author in author_info_test:
39        confusion_matrix[author] = {}
40        author_based_success_metrics[author]['prediction_count'] = len(
          author_info_test[author]['papers'])
41        for paper in author_info_test[author]['papers']:
42            predicted_author, prob = naive_bayes_predict_author(
                  author_info_train, paper)
43            confusion_matrix[author].setdefault(predicted_author, 0)
44            confusion_matrix[author][predicted_author] = confusion_matrix[
                  author][predicted_author] + 1
45
46            true_authors.append(author)
47            predicted_authors.append(predicted_author)
48
49            if predicted_author == author:
50                author_based_success_metrics[author]['tp'] =
                      author_based_success_metrics[author]['tp'] + 1
51            else:
52                author_based_success_metrics[author]['fn'] =
                      author_based_success_metrics[author]['fn'] + 1
53                author_based_success_metrics[predicted_author]['fp'] =
                      author_based_success_metrics[predicted_author]['fp'] + 1
54
55    #confusion_matrix_for_plotting = ConfusionMatrix(true_authors,
          predicted_authors)
56    #confusion_matrix_for_plotting.plot()
57
58    confusion_matrix_dataframe= pd.DataFrame.from_dict(confusion_matrix)
59    confusion_matrix_filled=confusion_matrix_dataframe.copy()
60    confusion_matrix_filled=confusion_matrix_filled.fillna(0.0)
61
```

```
62     return confusion_matrix_filled, author_based_success_metrics
```

Main function of Naive Bayes

```
1  def naive_bayes():
2      input_filename = 'data/authors.zip'
3
4      #split dataset into train and test files
5      author_info_train, author_info_test = split_files(input_filename)
6
7      #Train Naive Bayes classifier
8      author_info_train = naive_bayes_train(author_info_train)
9
10     #Test the Naive Bayes classifier
11     confusion_matrix, author_based_success_metrics = naive_bayes_test(
           author_info_train, author_info_test)
12
13     #Evaluate predictions
14     author_based_success_metrics, overall_success_metrics =
           evaluate_predictions(author_based_success_metrics)
15
16     #Print evaluation metrics author_based_success_metrics_dataframe,
           overall_success_metrics_dataframe = print_success_metrics(
           author_based_success_metrics, overall_success_metrics)
17
18     #Return evaluation metrics
19     return author_based_success_metrics_dataframe,
           overall_success_metrics_dataframe, confusion_matrix
20
21 #Run the main method of Naive Bayes implementation with a given k value for
       add-k smoothing.
22 add_k = 5.5
23 author_based_success_metrics_dataframe, overall_success_metrics_dataframe,
       confusion_matrix = naive_bayes()
```

Support Vector Machine

New Feature Functions

Average sentence length

```
def average_sent_len(paper):
    import re
    paper=re.sub("\.\.\.", ".", paper)
    sentences = paper.split('.')
    length=0
    for x in sentences:
        length+=len(x.split())
    avg_length=length/len(sentences)
    return avg_length
```

Three dots counts

```
def count_three_dot(text):
    return text.count('...')
```

Article length

```
def article_length(paper):
    return len(paper)
```

Semicolon counts in an article

```
def num_semicolon(paper):
    return paper.count(';')
```

Create Vocabulary with tokens ids and assigns author ids

```
def authors_with_ids(corpus):
    #make list of list
    author_list_train=[[author] for author in corpus.keys()]
    author_list_train=sorted(author_list_train)
    #unique id to each author
    author_dict_train={}
    for i in range(len(author_list_train)):
        author_dict_train[author_list_train[i][0]]=i+1
    return author_dict_train
```

```
10
11 def vocabulary_with_ids(corpus):
12     #create word vocabulary for train set
13     vocabulary_train=set()
14     for author in corpus.keys(): #for each author
15         for paper in corpus[author]['papers']: #for each paper
16             paper = paper['raw']
17             for token in tokenize(paper): #for each token in tokenized paper
18                 vocabulary_train.add(token) #add token to the vocabulary
19     vocab_list_train=[[i] for i in vocabulary_train]
20     #unique id to each token
21     vocab_dict_train={}
22     for i in range(len(vocab_list_train)):
23         vocab_dict_train[vocab_list_train[i][0]]=i+1
24     return vocab_dict_train
```

Paper-based Corpus with ids

```
1  def corpus_with_ids_paper_based(corpus_train,corpus_test,author_dict,
       vocab_dict):
2
3      import collections
4      #create modified_corpus_train in dict of dict format with author ids
           and tokens ids together with token weights
5      #then order them with ids
6      modified_corpus_train=collections.OrderedDict()
7      for author in corpus_train.keys():
8          author_id = author_dict[author]
9          modified_corpus_train.setdefault(author_id,[])
10         for paper in corpus_train[author]['papers']:
11             paper_vocab = paper['vocab']
12             paper_words = {}
13             for word in vocab_dict.keys():
14                 if word in paper_vocab:
```

```
15                  paper_words[vocab_dict[word]] = 1
16
17          #add "article length" feature length_feature_value=
                article_length(paper_vocab)
18          paper_words[len(vocab_dict)+1]=length_feature_value
19
20          #add "semicolon counts in an article" feature
21          semicolon_count=num_semicolon(paper['raw'])
22          paper_words[len(vocab_dict)+2]=semicolon_count
23
24          #add "three dots counts" feature
25          paper_words[len(vocab_dict)+3]=count_three_dot(paper['raw'])
26
27          #add "average sentence length in an article" feature
28          paper_words[len(vocab_dict)+4]= average_sent_len(paper['raw'])
29
30          #order
31          paper_words = collections.OrderedDict(sorted(paper_words.items()
                ))
32          modified_corpus_train[author_id].append(paper_words)
33
34  #create modified_corpus_test in dict of dict format with author ids and
        tokens ids together with token weights
35  #then order them with ids
36  modified_corpus_test=collections.OrderedDict()
37  for author in corpus_test.keys():
38      author_id = author_dict[author]
39      modified_corpus_test.setdefault(author_id,[])
40      for paper in corpus_test[author]['papers']:
41          paper_vocab = paper['vocab']
42          paper_words = {}
43          for word in vocab_dict.keys():
44              if word in paper_vocab:
45                  paper_words[vocab_dict[word]] = 1
```

```
46
47              #add "article length" feature length_feature_value=
                    article_length(paper_vocab)
48              paper_words[len(vocab_dict)+1]=length_feature_value
49
50              #add "semicolon counts in an article" feature
51              semicolon_count=num_semicolon(paper['raw'])
52              paper_words[len(vocab_dict)+2]=semicolon_count
53
54              #add "three dots counts" feature
55              paper_words[len(vocab_dict)+3]=count_three_dot(paper['raw'])
56
57              #add "average sentence length in an article" feature
58              paper_words[len(vocab_dict)+4]= average_sent_len(paper['raw'])
59
60              #order
61              paper_words = collections.OrderedDict(sorted(paper_words.items()
                    ))
62              modified_corpus_test[author_id].append(paper_words)
63      #order
64      modified_corpus_train = collections.OrderedDict(sorted(
            modified_corpus_train.items()))
65      modified_corpus_test = collections.OrderedDict(sorted(
            modified_corpus_test.items()))
66
67      return modified_corpus_train,modified_corpus_test
```

Train and Test Files

```
1  import sys
2  import codecs
3
4  def create_train_and_test_files(train_data,test_data):
5      #create train file
6      output_train_filename = 'data/train_file_ordered.txt'
```

```
7     output_train_file = codecs.open(output_train_filename, 'w', encoding='
          utf-8')
8
9     for author_id in train_data.keys():
10        for paper in train_data[author_id]:
11            line = str(author_id)+' '
12            for word_id, freq in paper.iteritems():
13                line =line + '%d:%d ' % (word_id ,freq)
14            line = line.strip() + '\n'
15            output_train_file.write(line)
16
17    output_train_file.close()
18    #create test file
19    output_test_filename = 'data/test_file_ordered.txt'
20    output_test_file = codecs.open(output_test_filename, 'w', encoding='utf
          -8')
21
22    for author_id in test_data.keys():
23        for paper in test_data[author_id]:
24            line = str(author_id)+' '
25            for word_id, freq in paper.iteritems():
26                line =line + '%d:%d ' % (word_id ,freq)
27            line = line.strip() + '\n'
28            output_test_file.write(line)
29
30    output_test_file.close()
31    print "train file path: ", output_train_filename ,"\n"
32    print "test file path:", output_test_filename ,"\n"
```

Test SVM

```
1 import pandas as pd
2 def svm_test(author_info_train, author_info_test,ordered_corpus_test,
      reverse_author_dict_train):
3
```

```
4    author_based_success_metrics = {}
5    for author in author_info_test: author_based_success_metrics.setdefault
         (author, {})
6        author_based_success_metrics[author].setdefault('tp', 0)
7        author_based_success_metrics[author].setdefault('fp', 0)
8        author_based_success_metrics[author].setdefault('fn', 0)
9
10   confusion_matrix = {}
11    prediction_file='data/predictions_paper_based.txt'
12
13   with open(prediction_file) as f:
14       for author_id in ordered_corpus_test:
15           author = reverse_author_dict_train[author_id]
16           confusion_matrix[author] = {}
17           author_based_success_metrics[author]['prediction_count'] = len(
                 author_info_test[author]['papers'])
18           for paper in ordered_corpus_test[author_id]:
19               line = f.readline()
20               predicted_author_id=int(line.split('␣')[0])
21         predicted_author=reverse_author_dict_train[predicted_author_id]
22         confusion_matrix[author].setdefault(predicted_author, 0)
23               confusion_matrix[author][predicted_author] = confusion_matrix
                     [author][predicted_author] + 1
24
25               if predicted_author == author:
26                   author_based_success_metrics[author]['tp'] += 1
27               else:
28                   author_based_success_metrics[author]['fn'] += 1
29                   author_based_success_metrics[predicted_author]['fp'] += 1
30
31   confusion_matrix_dataframe= pd.DataFrame.from_dict(confusion_matrix)
32   confusion_matrix_filled=confusion_matrix_dataframe.copy()
33   confusion_matrix_filled=confusion_matrix_filled.fillna(0.0)
34
```

```
35    return confusion_matrix_filled, author_based_success_metrics
```

Main Function of SVM

```
1  def svm():
2      input_filename = 'data/authors.zip'
3      #split corpus into train and test files
4      author_info_train, author_info_test = split_files(input_filename)
5
6      #create vocabulary dictionary with ids
7      vocab_dict_train=vocabulary_with_ids(author_info_train)
8
9      #create author dictinary with ids
10     author_dict_train=authors_with_ids(author_info_train)
11
12     #create train corpus and test corpus in the form of: authorid f1:v1 f2:
           v2
13     ordered_corpus_train,ordered_corpus_test=corpus_with_ids_paper_based(
           author_info_train,author_info_test,author_dict_train,
           vocab_dict_train)
14
15     #create train and test files to use in SVM_Multiclass executer
16     create_train_and_test_files(ordered_corpus_train,ordered_corpus_test)
17
18     reverse_author_dict_train = dict([(value, key) for key, value in
           author_dict_train.iteritems()]) #for calls in svm_test
19     confusion_matrix, author_based_success_metrics = svm_test(
           author_info_train, author_info_test,ordered_corpus_test,
           reverse_author_dict_train)
20
21     author_based_success_metrics, overall_success_metrics =
           evaluate_predictions(author_based_success_metrics)
22     author_based_success_metrics_dataframe,
           overall_success_metrics_dataframe = print_success_metrics(
           author_based_success_metrics, overall_success_metrics)
```

```
23
24      return author_based_success_metrics_dataframe,
            overall_success_metrics_dataframe, confusion_matrix
```

References

[1] Thorsten Joachims *Multi-Class Support Vector Machine* Cornell University Department of Computer Science, 2008 https://www.cs.cornell.edu/people/tj/svm_light/svm_multiclass.html

[2] OpenCV tutorials ml module. Machine Learning http://www.swarthmore.edu/NatSci/mzucker1/opencv-2.4.10-docs/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html

[3] Naive Bayes. scikit-learn documentation http://scikit-learn.org/stable/modules/naive_bayes.html

[4] K. Crammer and Y. Singer. On the Algorithmic Implementation of Multi-class SVMs, JMLR, 2001. http://jmlr.csail.mit.edu/papers/volume2/crammer01a/crammer01a.pdf

[5] James Conigliaro *Author Identification Using Naive Bayes Classification* Marquette University,Department of Electrical and Computer Engineering