COMP30027 Report

Anonymous

0. Introduction

During the years there is an increasing amount of publicly accessible blogs, it offers us opportunity to harvest information and answer interesting questions like: can we recognize just from reading the text the author? If not the author so maybe the gender of an author or maybe his age?. How content style, length of words and more features change between different ages and more. In this project we focus on predict the range of age of an author according to the text and some information about popular words that we were given in csv files.

In the following sections I will describe the task we were given, what we have done - mainly about feature engineering, different direction of thoughts to attack this problem, configurations and tuning of the different learners we used and the evaluation part of the learners. In addition, I will include some short summary and explanations of related literature we read. Finally, we will explain about the learners and the different error rates.

1. The Task, our corpus and related literature

The task is to predict for each author in the test file the correct class, which is one of the following: “14-16”, “24-26”, “34-36”, “44-46”, by using one or more learners which can be trained on one or more of the following files:
Train_raw.csv - here we have user ID, age, text.
Train_top10.csv - here we have popular words and count of times the author mentioned this word with the age of the author.
In the training - all instances are fallen to one of the classes, but in test we have 24% of instances with author in an age which is not in the ranges we trained the learner on.

1.1. Related literature

In this document I suggested some feature engineering and explained about the features we chose for this problem, there are a lot of literature with thorough research about feature engineering for text classification problem such as T. Raghunadha Reddy, B. Vishnu Vardhan, P. Vijayapal Reddy(2016). There are also a lot of researchers who published different accuracies for TF-IDF for text classification problems such as Rini Wongso(2017). Moreover, Eckert (1996) gives an overview of the literature on age as a sociolinguistic variable. Linguistic variation can occur as an individual moves through life, or as a result of changes in the community itself as it moves through time. As an added complexity, Argamon et al. (2007) found connections between language variation and age and gender. Finally, Schler et al. (2006) shows some differences in content and style over age.

1.2. Training corpus

Here we will show in the following tables the distribution of the authors to the classes in the training data. In addition, we will include the distribution after pre-processing of the data which is to concatenate text for the same user ID(because we know it’s the same author, we prefer to save it as one row with a lot of concatenated text for this author which is better for prediction because we have more text to train on and we don’t want to “challenge” our predictor for vain on instances we have already know the accuracy).

Table 1: Authors distribution over age
Without Pre-Processing

Age	Counts	%age
14-16	4109	0.46
24-26	3820	0.42
34-36	807	0.09
44-46	214	0.023
Total	8950	

Table 2: Authors distribution over age
After Pre-Processing

Age	Counts	%age
14-16	98454	0.356
24-26	141104	0.51
34-36	30347	0.11
44-46	6510	0.023
Total	8950	

2. Implementation- Feature Engineering, learners we have used

2.1 Classifiers Used:

- **Logistic Regression**
- **Naive Bayes**
- **SVM**

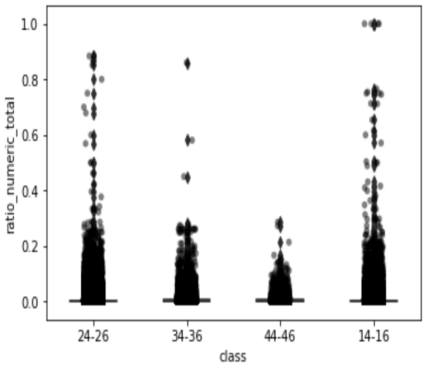
2.2 Features Engineered

2.2.1 Engineered features - not included in our solution

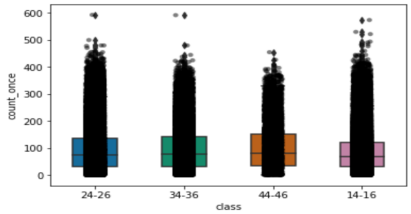
The features we finally include in our solution are the last two - TF, TF-IDF, before that we used a lot of different NLP features which are worse for the models to predict with, but can be important for future work on this task, like combining some of them together as extra to tf-idf.
We read through article[1] which suggests Character, Lexical, Syntactic, Structural and readability features we can extract from the text in order to predict the range of an age of author.
We wrote a python program that reads the csv file and creates a new one with the new numeric features we extracted from the text, then we also normalized the values between -1 to 1 because we have different size of values in the features. Here are some of the features that we engineered and include in the new csv:

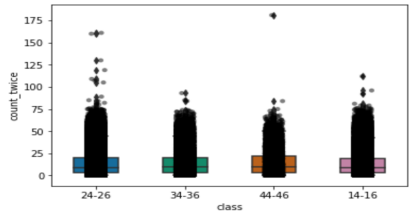
Character based features:

E.g ratio of numeric data to total characters in the text

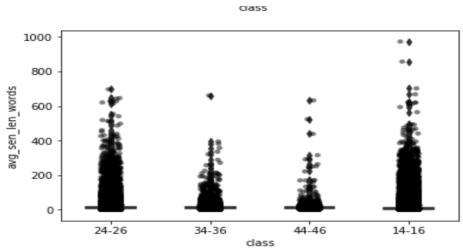


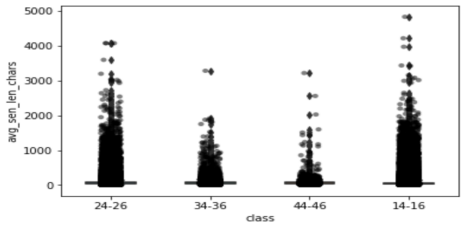
Lexical based features: E.g number of words that appearing once and twice.





Structural features: E.g Average sentence length in terms of characters, Average sentence length in terms of words.





Readability features: we chose to include Flesch-Kincaid which gives each text a score in range based on readability of the text which consider number of syllables, number of sentences and number of words, then according to this range we can map to range of ages. The problem with this feature is that it gives a score for ages between 5-grade old to graduate-old so it's not cover all of our classes, and also we found it not accurate enough.
Score for Readability is given by :

$$206.835 - 1.015\left(\frac{\text{total words}}{\text{total sentences}}\right) - 84.6\left(\frac{\text{total syllables}}{\text{total words}}\right)$$

We used chi2 for feature selection, we recursively reduce each feature at a time to find the best combination of features. We trained Linear Regression, SVM, Logistic Regression and Decision Tree learners. If we predict on data which includes only ages in our trained ranges(we removed instances in dev of age which is not in one of the classes) we got accuracy between 50-55% for all of the learners which means 40-45% if we should predict on all of the data(including instances with age which is out of ranges), it's not a good accuracy it's close to Z-R, so we looked into new features. We check using other readability tests and POS-tagger(count for each text how many verbs, nouns and so on) - but still, hard to differentiate between classes.

Next thing we have done is feature engineering on top10 csv file- this file contains in each row count of popular words for the author, the problem is that almost every cell in a row is 0, so features almost always share the same value - zero which is not good for the learners. We decided we should pre-processing the data before give it to our learners. We have done the following: group by each class and sum the counters, then calculate probability of class given a popular word as we can see in this dictionary:

```
anyways {'14-16': 0.67, '24-26': 0.31, '34-36': 0.02, '44-46': 0.0}
cuz {'14-16': 0.77, '24-26': 0.19, '34-36': 0.04, '44-46': 0.0}
digest {'14-16': 0.03, '24-26': 0.08, '34-36': 0.89, '44-46': 0.0}
diva {'14-16': 0.04, '24-26': 0.21, '34-36': 0.75, '44-46': 0.0}
evermean {'14-16': 0.0, '24-26': 0.0, '34-36': 1.0, '44-46': 0.0}
fox {'14-16': 0.16, '24-26': 0.35, '34-36': 0.09, '44-46': 0.4}
gonna {'14-16': 0.68, '24-26': 0.28, '34-36': 0.03, '44-46': 0.01}
greg {'14-16': 0.15, '24-26': 0.43, '34-36': 0.1, '44-46': 0.33}
haha {'14-16': 0.87, '24-26': 0.12, '34-36': 0.01, '44-46': 0.0}
jayel {'14-16': 0.0, '24-26': 0.0, '34-36': 0.0, '44-46': 1.0}
kinda {'14-16': 0.65, '24-26': 0.32, '34-36': 0.03, '44-46': 0.01}
levengal {'14-16': 0.0, '24-26': 0.0, '34-36': 0.0, '44-46': 1.0}
literacy {'14-16': 0.02, '24-26': 0.17, '34-36': 0.8, '44-46': 0.01}
lol {'14-16': 0.83, '24-26': 0.1, '34-36': 0.03, '44-46': 0.04}
melissa {'14-16': 0.36, '24-26': 0.35, '34-36': 0.02, '44-46': 0.27}
nan {'14-16': 0.28, '24-26': 0.29, '34-36': 0.03, '44-46': 0.41}
nat {'14-16': 0.45, '24-26': 0.16, '34-36': 0.0, '44-46': 0.38}
postcount {'14-16': 0.0, '24-26': 0.48, '34-36': 0.52, '44-46': 0.0}
ppl {'14-16': 0.86, '24-26': 0.13, '34-36': 0.01, '44-46': 0.0}
rick {'14-16': 0.09, '24-26': 0.28, '34-36': 0.12, '44-46': 0.51}
school {'14-16': 0.56, '24-26': 0.35, '34-36': 0.08, '44-46': 0.02}
```

We found that only half of the instances in the dev_raw contains in the text column some of the popular words, for those instances we compare the average of probabilities for class given word and take the class with the maximum, then we got 63% accuracy if we predict only on these instances. The problem with this approach is that we can predict only on half of the data(only texts which include some popular words)

Finally, we decided to return back to the train_raw csv with the text column, after reading to the following articles, we found the TF-IDF method for text classification, which in the end was our solution, here is a summary of this method:

2.2.2 Engineered features - included in our solution

TF and IDF features:

The TF (term frequency) of a word is the frequency of a word (i.e. number of times it appears) in a document
The IDF (inverse document frequency) of a word is the measure of how significant that term is in the whole corpus.

TF-IDF is an information retrieval technique weighs a term's frequency (TF) and its inverse document frequency (IDF). Each word or term has its respective TF and IDF score. The product of the TF and IDF scores of a term is called the TF*IDF weight of that term.
First step is to get a matrix of count vector - each row has counter for each word in the text, cells which are zeros means this word no appear in this text. Then, get a matrix of inverse frequency, each row contains how much each word is rare comparing to other documents, then by multiplication we have the final matrix of TF-IDF. More formally, For a term **t** in a document **d**, the weight **Wt,d** of term t in document d is given by:

$$Wt,d = TFt,d*\log (N/DFt)$$

Where:

- TFt,d is the number of occurrences of t in document d.
- DFt is the number of documents containing the term t.
- N is the total number of documents in the corpus.

Before we apply the TF-IDF we wanted to pre-process the text, we consider the following:
Normalize the text - remove spaces, remove stop words which are not useful for text classification like: are, is, the and so on.
Stemming - we used snowball stemming to replace words with similarity like "ambitious" and "ambition" and "ambitiously" - we wanted to refer all of them as the same, the stemming create one word for all of them: "ambit", because we think it's probably the same author using the word and not a different one. Then, we read the file and prepare the TF-IDF matrix for our learners.
We applied the TF-IDF on the pre-process data(after concatenation of text of the same user ID). Stemming and normalization didn't change much the accuracy so we didn't include them.

3. Evaluation of classifier over the development documents

To evaluate the model at development data we looked at the precision, recall and F1 score for each of the models(Naive Bayes, Logistic Regression and SVM).
We tuned the parameters for each model and found that some models were good at predicting the age of authors with range "14-16" while some were good at predicting age "24-26" but none were so good at predicting authors with age "34-36" and "44-46".
In order to judge our performance we used a baseline of zero-R as instances of age "24-26" were 38.15% of dev data. We could easily beat this by all of our classifiers which used tfidf metrics which considers all possible distinct words with frequency in each instance of author.
We also see our classifiers were tested with .05% of data range "44-46" and 0.012% of data from range "33-36" , .28% of data is of range "14-16".
Our classifiers gave high scores for classes "24-26" and "14-16" as it was trained for 0.36% and 0.51% respectively while due to not enough of trained data for classes "34-36" and "44-46" (0.1% and 0.02%) respectively. While there is 24% of unseen data also while testing our models. Below are evaluation table for precision, recall and f1 scores of different classifiers with best parameters(approx 4 runs for different parameters):

Naive bayes (smoothing =1e-6)

Classes	Precision	Recall	F1 score	Instance
14-16	0.72	0.80	0.76	13100
24-26	0.53	0.94	0.68	17298
34-36	0.06	0.01	0.01	2584
44-46	0	0	0	551

Logistic Regression (regularisation parameter C=13)

Classes	Precision	Recall	F1 score	Instance
14-16	0.71	0.94	0.81	13100
24-26	0.60	0.95	0.73	17298
34-36	0.48	0.07	0.13	2584
44-46	0	0	0	551

SVM (learning parameter, alpha=1e-6)

Classes	Precision	Recall	F1 score	Instance
14-16	0.73	0.93	0.82	13100
24-26	0.58	0.95	0.72	17298
34-36	0.27	0.05	0.16	2584
44-46	0	0	0	551

While ensembling we tried to evaluate the final accuracy with f1 score and checked that Logistic Regression is good at predicting class 24-26 while SVM at 14-16, so while ensembling we checked if Logistic Regression predicted class 24-26, we say class 24-26 irrespective of SVM and when SVM

predicted class 14-16 we say class 14-16 irrespective of Logistic Regression.
This approach didn't increase much of our accuracy and found that it ran best for Logistic Regression.

Trial attempt to predict Unseen data

Finally, we can do a huge improvement to our learners if we will find a way to predict also instances with age which is not in one of our classes, for now we make a mistake for 24%(unseen rows) of instances in dev because we our classifier predict one of the class for them.
We used predict_proba to get information of confidence level of the classifier in his prediction.
Predict_proba - probability of classifier to predict each class [sum predictions to 1].
We investigated this information, we found that if classifier predict a class and it was indeed the actual class so the probability is usually very high, but not always, also the distance to hyperplane is usually close but not always, so we couldn't find a good threshold like if probability is more than 0.8 so this is actual class, otherwise it's unknown class("?"),. But if all probabilities close to 0.25 so it's unknown class and some more different trials. Because we didn't find a good formula for it, we can't get improve so we still predict only in our classes, that's why maximum accuracy we can get is 100-24 = 76%.
The 50% percentile are as follows for confidence of prediction of class:-

Prediction of class 14-16 if it was actually 14-16	0.9357041919112846
Prediction of class 14-16 if it was not actually 14-16	0.8945721690366383
Prediction of class 24-26 if it was actually 24-26	0.8193096554141753
Prediction of class 24-26 if it was not actually 24-26	0.7643001426827365

We can see that there is not a huge difference between prediction of class given it's the actual class and prediction a class if the actual is different one: almost 50% of the data overlaps with nearly same confidence level. Thus, it did not help us in predicting the unseen data.

4. Analysis of the behavior of our classifiers and error rate

In this part i'm going to explain how the theoretical concepts of our Logistic Regression, SVM and Naive bayes classifiers are applied in our classification problem with using TF-IDF(word level) input matrix.

One method that we want to focus on and compare it to the others is naive bayes, it's interested us why NB couldn't achieve good accuracy like SVM and Logistic regression. First, let's understand how Multinomial NB works with our TF-IDF input matrix.

First step is to calculate the probability of each class, with this formula: P(c) = fc / fd where fc is number of training documents which is labelled with.

c $\in$ {"14-16","24-26", "34-36", "44-46"}

fd = number of all training documents in all classes.

xi $\in$ distinct word of a document

n = number of features

|V| = size of vocabulary.

Then, probability of document to each class is calculated with this formula:

P(c|d) = P(c)*P(x1 ∈ d | c) * P(x2 ∈ d | c) * … * P(xn ∈ d | c)

From this formula, a class with the highest probability result will be assigned as the class of document d. We used Multinomial Naive Bayes Classifier variation which is suitable for discrete counts (as opposed to Bernoulli which is suitable for binary feature vectors) and opposed to Gaussian which assumes features (words in our case) have normal distribution.

In order to calculate P(xi ∈ d | c) naive bayes use the TF-IDF matrix which holds the scores for the words. We used naive bayes with both countvectorizer which is a matrix for TFi for each word i and tf-idf vectorizer which is a matrix for tf-idf scores.

Here we explain how naive bayes work with countvectorizer matrix:
The distribution is parameterized by vectors $\theta_y = (\theta_{y1}, \ldots, \theta_{yn})$ for each class y between ("14-16","24-26","34-36","44-46"), and θ_{yi} the probability $P(x_i \mid y)$ of word i appearing in a text belonging to class y.
The parameters θ_y is estimated by a smoothed version of maximum likelihood, i.e. relative frequency counting:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

where $N_{yi} = \sum_{x \in T} x_i$ is the number of times word i appears in a text of class y in the training set T, and $N_y = \sum_{i=1}^{|T|} N_{yi}$ is the total count of all words for class y.
The smoothing priors $\alpha \geq 0$ accounts for features not present in the learning samples and prevents zero probabilities in further computations. We found that using Lidstone smoothing ($\alpha < 1$) will give us better accuracy than Laplace smoothing ($\alpha = 1$) which is the default.
The same to tf-idf, just replace the count with tf-idf score and it's the same mechanism.

Discussion with Practical Observations

- Naive Bayes calculation is based on Bayes' theorem with assumption that there is no word related to each other (every word is independent).
- If we look at how naive bayes work, we can see that it ignores the order of words or slang. If we have authors which always tend to use slang or write some words in same order, its highly likely for naive bayes to miss it as it multiplies the probabilities of each feature for the instance of a given class. Thus, considers appearance of word to be in any order.
    - Evidence for dependencies: we found the consecutive words "NEXT GOAL", "Dear Friends" and many more n-grams which always comes together for unique authors.
- It works good because of large training instances of class 14-16 and class 24-26. Naive bayes tends to correctly predict the authors if they have used some kind of words a lot or if they has used some kind of distinct words, the probability of word appearing in the class of that author is high and has high tf-idf score too. Thus, classes with 24-26 and 14-16 contain most of the training data and testing data, the model predict them correctly with good F1 score as the count of words for those classes trained would be high and hence probability of those features given that class would also be more. As a result, those instances increases

the confidence to predict the majority classes correctly and almost tiny for least dominant classes as more instances empower the probability factor for those instances.

- Thus we can see in our evaluation also that it miserably fails to predict class 34-36, 44-46 while other models predicted a bit because the training model was not enough to increase the count of words appearing for that documents.
- The biggest difference between the models you're building from a "features" point of view is that Naive Bayes treats them as independent, whereas SVM looks at the interactions between them to a certain degree So if you have interactions, and, given your problem, you most likely do, an SVM will be better at capturing those, hence better at the classification task you want.

- Naive Bayes comes under the class of generative models for classification. It models the posterior probability from the class conditional densities. So the output is a probability of belonging to a class. SVM on the other hand is based on a discriminant function given by y = w.x+b. Here the weights w and bias parameter b are estimated from the training data. It tries to find a hyperplane that maximises the margin and there is optimization function in this regard.

- MNB is stronger for snippets than for longer documents. While (Ng and Jordan, 2002) showed that NB is better than SVM with few training cases, MNB is also better with short documents. SVM usually beats NB when it has more than 30–50 training cases, we show that MNB is still better on snippets even with relatively large training sets.
    - Evidence NB is better with short text: we trained our NB and SVM on rows with text with length less than 25 characters and didn't use the conctination of texts for the same author (we would like to keep the text short). The accuracy of NB: 0.33 while SVM 0.28, so if previous NB was with lower accuracy, now it's with higher accuracy. One reason for it is that with shorter text we can find less dependent words.

5 Error rate analysis

5.1 Multinomial NB error rate analysis

As we explained before, NB has relatively bad accuracy because of the strong assumption of independence between features (words) which is not satisfied. We find that we got lower error rates with lower smoothing and higher ones with higher smoothing.

Smoothing effect on error rate:

smooth	accuracy
0.000001	59.194
0.0001	58.181
0.001	57.38

Analysis error rate by confusion matrix:

	PRED			
TRUE	14-16	24-26	34-36	44-46
14-46	10516	2584	0	0
24-26	897	16278	123	0
34-36	18	2550	16	0
44-46	19	516	16	0

5.2 Logistic regression error rate analysis

C parameter- For small values of C, we increase the regularization strength which will create simple models which underfit the data. For big values of C, we low the power of regularization which implies the model is allowed to increase its complexity, and therefore, overfit the data. We found it's better to use higher C then the default, in our final solution we are not predicting unseen instances so more overfit the data is better.

C parameter effect on error rate:

alpha	accuracy
1	0.609
3	0.63
5	0.632
10	0.6368
13	0.639
20	0.6404
100	0.635

Analysis error rate by confusion matrix:

	PRED			
TRUE	14-16	24-26	34-36	44-46
14-46	12424	676	93	0
24-26	412	1952	220	0
34-36	412	1952	220	0
44-46	12	494	45	0

5.3 SVM error rate

Alpha- learning rate - low alpha means slow learning - better to find the optimal solution but it takes more time and the algorithm can be stuck, high alpha means we can miss the optimal solution (by gradient descent implementation means - may skip the maxima or minima of the function because we take "big steps"). We found it's better to use lower alpha than the default.

learner parameter effect on error rate:

learning	accuracy
1.00E-05	0.628
1.00E-03	0.584
1.00E-01	0.415
1.00E+00	0.416

Analysis error rate by confusion matrix:

	PRED			
TRUE	14-16	24-26	34-36	44-46
14-46	12336	726	8	0
24-26	1124	15991	169	14
34-36	651	1814	119	0
44-46	26	448	77	0

References

[1] T. Raghunadha Reddy, B. Vishnu Vardhan, P. Vijayapal Reddy(2016). A Survey on Authorship Profiling Techniques, International Journal of Applied Engineering Research ISSN 0973-4562 Volume 11, Number 5 (2016).

[2] 2nd International Conference on Computer Science and Computational Intelligence 2017, ICCSCI 2017, 13-14 October 2017, Bali, Indonesia

[3] Dong Nguyen Noah A. Smith Carolyn P. Rose´ Language Technologies Institute Carnegie Mellon University Pittsburgh, PA 15213, USA

[4] Schler J, Koppel M, Argamon S, Pennebaker J (2006) Effects of age and gender on blogging. In: Proceedings of 2006 AAAI Spring Symposium on Computational Approaches for Analyzing Weblogs.