GENDER CLASSIFICATION ON TWITTER TEXT

Sarat Chandra Karasala
Shankar Veruva Paramatma
Debdeep Guha

June 14, 2019

Abstract

It is a group assignment where we have developed a classifier that can assign a set of twitter texts to their corresponding labels. This is a gender classification task, where we have developed a classifier that can identify the gender of the tweet's author as accurate as possible.

Table of Contents

1 Introduction

Text classification is one of the main tasks in **Natural Language Processing**. Text classification has many applications like **sentiment analysis, topic modelling, spam filtering** etc. These applications can be both supervised and unsupervised. Topic Modelling is an example of **unsupervised classification**. Here there is no pre labelled text. The model works without any context or prior knowledge of the corpus. On the other hand, **supervised learning** models take large amounts of pre labelled text learn it. They are then able to take new data classify it.

For this assignment our task is **Authorship Analysis**. We are given a corpus of tweets (Posts by people on the social media platform Twitter) from different users. There are 3600 users with each users' posts in an XML file. Among those, we are given the gender labels of 3100 entries. We need to build a model that can classify the remaining 500 users in the test dataset as accurately as possible.

Text is a rich source of information, but only to a human reading it. To a computer it is only a string of 1s and 0s. A computer cannot read and understand text like a human does. So, in order to train a system to recognise text we need to convert that text into something a computer can understand. **Numbers.**

We split up this project into 5 segments – data extraction, data wrangling, pre-processing/feature extraction, building a classifier and calculation of accuracy. Once we extract the text bodies from the input files, we have text written by users. This will have spelling mistake and personal idiosyncrasies like emoji and other fancy way or writing or saying things like different slang. The pre-processing steps used in our task are detailed below

Once pre-processing is done, we can move on to converting the data into something a machine can learn. This process is called **Feature Extraction**. One popular method of feature extraction is **TF-IDF**.

Shankar initiated the project by extracting the data from all the XML files, structured the content into a data frame with the given train labels and did basic pre-processing steps such as replacing multiple spaces with single space, substituting the emoji unicode and sites with 'emoji' and 'weblinks' respectively. After basic pre-processing, Sarat took up on the next steps of pre-processing such as removing the punctuations, stop words, tokenize, segregate male and female rows and then check the most common words used by each gender and which ones are relevant to build classifier. Debdeep performed lemmatisation, vectorized the tokens using TF-IDF vectorizer, encoded the genders into 1s and 0s and classified the different emoticons used into various categories to be used in the model. After this, all 3 of us, built 3 classifiers – SVC, logistic regression and naïve bayes and performed cross validation. Then we moved onto do the same pre-processing steps with the test data and ran it on all 3 models to predict labels, calculated respective accuracy and compared accuracies to find out which one is a better model. All of us worked on the code commenting and documentation.

2 Pre-processing and feature generation

Our main goal is to do **Gender Classification** and we are provided with a huge list of XML files which contains twitter posts from many authors and a train dataset which has document ids and gender of the author. So, we got the document id along with its internal twitter post data and gender out into a dataframe. After, that comes our main task, i.e., **Data Pre-processing**. It is a data mining technique through which we intend to transform raw human readable data into machine understandable format. The twitter data consists of a lot of weblinks, i.e. https links, emojis, tagged twitter accounts (user ids

starting with '@'), hash tagged words (e.g. '#LOL', '#Cricket'), stop words and many more. Hence, we want to transform our data into a format which will make it easy for the machine to learn patterns in the data and further increase our model accuracy.

2.1 Pre-processing

Below are the steps we have performed under pre-processing stage:

1. Replacing double spaces with single spaces from our twitter post content.
2. Removing punctuations and numeric characters.
3. We attempted to format the words in such a way that it considers individual emoji characters, then using a dataset downloaded from Kaggle, we checked for each emoji that existed there and replaced it with the subgroup it belonged to. Our idea was to see if a gender used a set of emojis more than the other, which at the end of the day would give us a better idea of the data but it resulted into low accuracy. So, we decided to drop it.
4. Replaced all weblinks (i.e. https links) into a single word 'weblink' and all emojis into a single word 'emoji'. Our idea was to see if one gender used emojis/weblinks more than the other, which at the end of the day would give us a better idea of the data.
5. Maintained the integrity of the words coming after '@' or '#' by simply removing these characters and keeping them but this wasn't fetching us high accuracy so we dropped this too.
6. Removed some weird symbols such as '…', 'ʃ ノ ಠ_ಠ ʅ ノ ︵┻━┻ 'etc. which we were not able to remove by the above Pre-processing steps.
7. Removed words that had less than 2 characters.
8. Before removing stop words, we wanted to check if there were any stop words that one gender used more than the other but based on our analysis, we found that both genders had almost equal amount of stop words that were being used, so we removed all the stop words.

Next step is Feature Extraction, which simply refers to extracting important subset of features from any data provided, that we can use for implementing our classification task. So, we convert of corpus data into vectors which is easier for the machine to cope with. To achieve this, we use two traditional Bag of Words approach, **TF-IDF and CountVectorizer**. Below, is a detailed description of what it is and what different parameters were passed to it.

2.2 Feature generation

TFIDF, known as **Term Frequency-Inverse Document Frequency**, will consider all the tokens that we pass into it. TF-IDF refers to the frequency of a token occurring in each twitter post (term frequency) and the number of documents in which that particular token exists (inverse document frequency). It makes sure that tokens which occur frequently in a document and which occur in the majority of the documents be penalized. It allows us to extract features that might be important for increasing model classifying accuracy.

Count Vectorizer is one of the most basic ways of representing data numerically. Here, we create vectors based on the frequency of occurrence of a token. TF-IDF acts as count vectorizer if we set 'use_idf' parameter to 'False'.

We worked with both of them and found that TF-IDFVectorizer yields better accuracy. Below is a list of parameters for TF-IDFVectorizer and an example of how to use it:

1. lower = True, (will lowercase all the tokens)
2. min_df = 2, (will remove tokens that occurs in less than the number of documents specified)

3. max_df = 1.0, (will remove words that occur too frequently, in the entire document, it will include all the documents if value set to 1.0)
4. ngram_range = 1,2 (this will tell the vectorizer to consider single occurring words as it is, and also combine any and all 2 words co-occurring in the corpus, also known as unigrams and bigrams, if we want to include more than 2 then we just have to specify that very number there)
5. tokenizer = LemmaTokenizerSpacy() (this is a function provided by Spacy that takes care of lemmatizing and tokenizing the tokens for it to be passed into the models)

After running these two vetorizers, we found out the following for this dataset:
1. **TF-IDF** works **better** than CountVectorizer for **SVC** and **LinearSVC** models.
2. **CountVectorizer** works **better** than TF-IDF for **Logistic Regression** model.
3. Lower values for min_df(2) and higher values for max_df(1.0) seem to give better accuracy score for **SVC** models.
4. Higher values for min_df(8), and high values for max_df(0.5) yielded the best result for **Logistic Regression**.

3. Models

Our task is **binary classification**. Classification is a machine learning process where the algorithm learns a set of data called training and then tries to identify which target class a new observation belongs to. This includes examples like spam detection, cancer detection in medicine etc. The data we have is the tweets of 3100 users. This text data has been transformed into a form that machine learning models can work with.

We now move to the classification algorithms. There are many classification algorithms that we can use. The algorithms we chose are Logistic Regression, SVC and Naïve Bayes classifier.

3.1 Logistic Regression

Logistic Regression (LR) is a special case of linear regression. The target variable here binary. That is it have only two possible values that are in most cases encoded to 0 and 1. LR computes the **probability** of the target being in one of the two classes. This is done using the **Sigmoid function** which takes any real values and converts it to between 0 and 1. The closer to 1 the probability is the higher the chances that the target will be classified 1. In order to get optimal results, there are a few **assumptions** that LR makes about the data.

- Just like with linear regression there should be **no multicollinearity**. This is when 2 or more of the predictor variables have a strong correlation amongst themselves.
- LR requires a large sample size to work optimally
- There should be a linear relationship between the logit of the outcome and each of the predictor variable

3.2 Linear SVC

Support Vector Machines is a supervised model that can be used for both **classification and regression** tasks. Support Vector Machines work by plotting each of the data points in an n dimensional space (where n is the number of predictor variables we have) and then trying to create a **hyperplane** to classify the data into two types. The hyperplane is a plane that can clearly split the

data into two. For example, if we have just one variable that we are using to predict another then we can represent the data as a scatter plot. The classifier then tries to draw a line that splits the data with the widest margin possible. If we have two predictor variables, then the hyperplane changes from a line to a plane. If we have 3 variables the plane is now 4 dimensional and not possible to visualize.

SVC is useful when we have nonlinear data. When a linear hyper plane is not enough to split the data, SVC transforms the data into higher dimensions and fits a higher dimensional hyper plane. This is called the Kernel trick. SVC also maximises the margins while ignoring outliers. Therefore, SVC is more robust to outliers

3.3 Naïve Bayes

Naïve Bayes is a probabilistic classifier that makes a strong assumption that there is no dependency between the various variables. We know that Bayes theorem predicts the probability of an event Y occurring given the probability of event X occurring. Here Y is the event of being a male author or female author and X is a TF-IDF representation of the documents.

Naïve Bayes makes two strong **assumption** about the data

- No two features in the predictor variables are dependent
- Each feature is given the same weight. That is all the variables contribute equally to the outcome

The algorithm is called naïve because in the real word assumption of true independence is not correct. Despite that fact the model works well in practise.

3.4 Discussion of model difference(s)

Logistic Regression is a highly efficient and simple algorithm to use. **It requires very little tuning** and computational resources. Another advantage is that LR is highly interpretable. It returns a probability that tells us how confident we can be about the classification. There is a big difference between saying that author is male with a probability of 0.54 or a probability of 0.91. But the drawback of LR is that its decision plane is linear. So, it can not be used on **non-linear data.**

Support Vector Machines can be used for both **classification and regression.** Since our task is binary classification we can use a **Support Vector Classifier**. SVM works relatively well when there is a clear distinction between the classes. SVM also handles non linear data. If there is no logical way to build a hyper plane with the given dimensions, SVM can scale the data to higher dimensions and find a non linear hyperplane that does the job. This process is called **a Kernel Trick**. However choosing the right kernel function is not easy. This along with the fact that SVM is not very easy to interpret like logistic regression makes the algorithm a powerful one but tricky to use. Another disadvantage with SVM is that since it classifies points by putting them on different sides of the hyper plane there is no probabilistic explanation for the assigned class.

Naïve Bayes algorithm's advantage is also its disadvantage. The assumption of independence between the feature variables might work in certain cases but it is not universal. When it holds true NB performs well and demands fewer computational resources. But when it comes to text classification, we cannot say that the presence of one word doesn't influence the occurrence of another. Text embedding is a process of learning the similarity or words and this process works by looking at what words occur together. Another disadvantage is if a word appearing in the test data is not present in the training data then the resulting probability will be 0 and the model will be unable to make a prediction. This is called Zero Frequency.

4 Experiment setups

As a part of experimentation, we worked on the following:

1. **Cross Validation**, a resampling procedure, used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. It is used to to estimate the skill of a machine learning model on unseen data. It generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split. By setting k =10, we applied CV on all our models and compared the accuracy.
2. We changed the **Feature Parameters**, ran the code and checked how it was impacting the accuracy. For instance, we tried to run the code with different min_df, max_df and n gram range under TF-IDFVectorizer for all our models and captured their performance.
3. As mentioned earlier, we played with the emojis by setting sub-group but as it wasn't working for us, we simply set each emoji symbol to emoji. Similar experiment with the @ and # tags. As it wasn't yielding high accuracy, we dropped it.
4. We even played with the **Classifier Parameters**, ran the code and checked how it was impacting the accuracy. For instance, we experimented on SVC by passing parameters kernel='rbf' which sets on rbf kernel for SVC. We had split the data into 80-20 ratio for each of our models under classifier before fitting the data and predicting the labels.
5. As trying out individual parameters was strenuous, we employed **GridSearchCV** which does an exhaustive search over specified parameter values for a classifier. We tried this on all our models and found out a set of parameters which gives us highest accuracy via 10-fold CV. For instance, the best set of parameters for SVC we found were C=10, gamma=0.7, kernel='rbf' and for Logistic Regression since it requires **little to no tuning** we were able to get our best result using no extra parameters.

5 Experimental results

After generating the predicted labels from each of our classifiers, we compared those with the true labels. We analysed our models based on Accuracy and MCC score generated by each model. **Accuracy** is a measure of how close the predicted value is to the true value whereas the Matthews Correlation Coefficient **(MCC)** allows one to gauge how well their classification model is performing. It ranges from -1 to +1 where -1 refers to completely wrong binary classifier and +1 refers to completely correct classifier. It takes into account true and false positives and negatives and is generally regarded as a balanced measure which can be used even if the classes are of very different sizes.

Below is the table for final accuracy and mcc score comparison for our models.

Model	Score	
	Accuracy	MCC
Logistic Regression	**0.816**	**0.6334**
SVC	0.8	0.604
Naïve Bayes	75	50

We can see from the above table that how each model predicts the labels accurately and how good of a model it is. Our base model had an accuracy of about 74% to 77% and the MCC score was around 0.5. As we did our experimentation with the CV and feature parameters and pre-processing, the **accuracy improved to 81.6% and the MCC score to 0.63.**

6. Conclusion

Based on our analysis, we found out that SVC and Logistic Regression perform better on such kind of textual data compared to Naïve Bayes. Logistic Regression works as a probabilistic discriminative model in this case. It generates the probability that a point belongs to a particular class and then assigns the class based on maximum probability. It is known work better for binary classification problems and large datasets whereas Naïve Bayes, on the other hand, works as a probabilistic generative model which models the underlying generation process. It is known to work better for small datasets. Finally, we have Support Vector Classifier (SVC), which is a supervised machine learning algorithm. In the SVC algorithm, we plot each data item as a point in n-dimensional space (where n is number of features) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well. It works really well when the data is clearly separable with margin. Also, it is effective in high dimensional spaces/features. However, it doesn't perform well when train set is very huge as it takes long time to get trained and when the true labels are very close by or overlapping.

References

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet allocation. *Journal of Machine Learning Research, 3*(Jan), 993-1022.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionally. *Advances in Neural Information Processing Systems 26*, 3111-3119.

Miller, G. A. (1995). WordNet: a lexical database for English. *Communications of the ACM, 38*(11), 39-41.

Pennington, J., Socher, R., & Manning, C. (2014). GloVe: Global Vectors for Word Representation. *the 2014 Conference on Empirical Methods in Natural Language Processing*, 1532-1543.

Appendix

			Final Test Data		
Number	Pre Processing Steps	Feature Vectorizer Parameters	Accuracy/ MCC Logistic Regression	Accuracy/ MCC Linear SVC	Accuracy/ MCC Naïve Bayes
1	weblink tag - YES emoji tag - YES Hashtags - Keep @ Mentions - Keep Funny Symbols - Removed	Removed all stop words Min DF 1 Max DF 1 nGrams 2	74/48	79/58	75/50
5	weblink tag - YES emoji tag - YES Hashtags - Keep @ Mentions - Keep Funny Symbols - Removed	Removed all stop words Min DF 15 Max DF 1 nGrams 2	78/57	78/56	74/48
6	weblink tag - YES emoji tag - YES Hashtags - Keep @ Mentions - Keep Funny Symbols - Removed	Removed all stop words Min DF 8 Max DF 0.9 nGrams 2	78/57	78/56	74/50
7	weblink tag - YES emoji tag - YES Hashtags - Keep @ Mentions - Keep Funny Symbols - Removed	Removed all stop words Min DF 8 Max DF 0.5 nGrams 2	79/58	77/55	74/49
9	weblink tag - Removed emoji tag - YES Hashtags - Keep @ Mentions - Keep Funny Symbols - Removed	Removed all stop words Min DF 8 Max DF 0.5 nGrams 3, 4	77.8/55.8	76.4/52.8	73.8/48.2

The above snippet shows a few of our selected experiments.