**Gender Prediction: Text Classification Analytics Using Naive Bayes Approach**

Zhipeng(Jason) Liu, Zhiyue(Jerry) Zhang, Justine Bernshine and Shivani Tyagi

The University of Manitoba, Winnipeg, MB, Canada

**Abstract**

As one of the data mining tasks, Naive Bayes algorithm provides valuable information about possibility. For instance, Naive Bayes algorithm can be applicable to analyze classify twitter topic, weather forecast, topic predictions and so on. In many real-life situations, data can be mined by classifying data step by step, then users get the result to process useful predictions. Naive Bayes conditional independence assumption actually holds, a Naive Bayes classifier will converge quickly than discriminative models like logistic regression, so we need less training data. A good bet for some kind of semi-supervised learning, or want something simple and perform very well. Hence, by using Naive Bayes algorithm to handle these situations, we propose this algorithm to form statistical models, and then to predict future gender of users. Testing results show the efficiency and prediction accuracy of our proposed Naive Bayes algorithm in user gender prediction.

**Keywords**

data mining, Naive Bayes algorithm, user gender prediction, training data, testing data, statistical models

**1. Introduction**

Data mining discovers implicit, previously unknown and potentially useful information or knowledge from data. Over past few decades, numerous algorithms have been proposed for handling different data mining tasks. As one of the important data mining tasks, Naive Bayes algorithm to produce the probability to predict the class of unknown data set. The discovered knowledge- in the form of the statistical probability-is useful in numerous real-life applications for predicting future probability. For instance, we analyze probability patterns from user gender relative their context on twitter in this paper so as to predict future web content and web design.

A user gender prediction is based on the content of user, we use Naive Bayes algorithm to process these training data set about the content of user on the twitter, then get the probability about which content is more likely write by man and which content is more likely write by women. Volkova, S. & Yarowsky, D, 2012). With the rapid growth of social media in recent years, there has been an increased interest in automatically characterizing social media users based on the informal content they generate. An important goal of this task of customer profiling or personal analytics is to label users with demographic categories, such as gender, age, ethnicity, or to determine user interests or preferences, such as political

orientation, favorite movies or product. Moreover, predicting user characteristics, preferences and opinions

from these personalized and diverse timely data can help answer important social science questions

and support many commercial applications including targeted computational advertising to

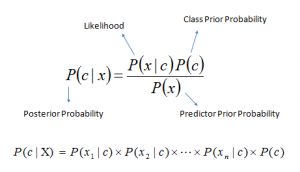
match user interest profile from Twitter or Facebook, detecting fraudulent product reviews

or branding analytics. In addition, Naive Bayes algorithm mining from content of user can also be used to predict user favor about web content and web design relative their gender, which in turn provides information on the product recommended to user based on their gender.

**2. Related works**

As a preview, we will reduce a social network analysis problem of finding the possibility of which gender write a specific sentence into a data mining problem. (Leung,c et al. 2016)Several data mining algorithms and techniques have been proposed during the past a few years, such as the detection of communities, subgraph mining and so on. Given that we will reduce our network analysis problem into a data mining problem of finding the possibility, we review some related works on possibility pattern mining in this section.

(R, SUNIL, 2015). The Naive Bayes that is a classification technique based on Bayes’ Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. We choose Naive Bayes as our algorithm because of naive models is easy to build and particularly useful for large data sets.



Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c) and P(x|c). Look at the equation above:

P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes).

P(c) is the prior probability of class.

P(x|c) is the likelihood which is the probability of predictor given class.

P(x) is the prior probability of predictor.

(Brownlee, J, 2014). The Naive Bayes Algorithm can be divided up by five steps:

1. Handle Data: Load the data from CSV file and split it into training and test datasets.
2. Summarize Data: summarize the properties in the training dataset so that we can calculate probabilities and make predictions.
3. Make a Prediction: Use the summaries of the dataset to generate a single prediction.
4. Make Predictions: Generate predictions given a test dataset and a summarized training dataset.
5. Evaluate Accuracy: Evaluate the accuracy of predictions made for a test dataset as the percentage correct out of all predictions made.

**3. Steps and Algorithms for Naive Bayes**

1. Handle Data

First we need to load csv into our python file. We need convert the attributes that were loaded as strings into numbers that we can work with them. Below is the loadCsv() function for loading the Pima Indians dataset.

Next we need to split the data into a training dataset that Naive Bayes can use to make predication and a test dataset that we use to evaluate the accuracy of the model. We need to split the data set randomly into train and datasets.

1. Summarize Data

The Naive Bayes model is comprised of a summary of the data in the training dataset.

1. Separate Data by class

We can do that by creating a map of each class value to a list of instances that belong to that class and sort the entire dataset of instances into the appropriate lists.

1. Calculate Mean

The mean is the central middle or central tendency of the data, and we will use it as the middle of our Gaussian distribution when calculating probabilities.

We also need to calculate the standard deviation of each attribute for a class value. The standard deviation describes the variation of spread of the data, and we will use it to characterize the expected spread of each attribute in our Gaussian distribution when calculating probabilities.

The standard deviation is calculated as the square root of the variance. The variance is calculated as the average of the squared differences for each attribute value from the mean. Note we are using the N-1 method, which subtracts 1 from the number of attribute values when calculating the variance.

1. Summarize Dataset

For a given list of instances (for a class value) we can calculate the mean and the standard deviation for each attribute.

1. Summarize Attributes by class

We can pull it all together by first separating our training dataset into instances grouped by class. Then calculate the summaries for each attribute.

1. Make Prediction
2. Calculate Gaussian Probability Density Function

We can use a Gaussian function to estimate the probability of a given attribute value, given the known mean and standard deviation for the attribute estimated from the training data.

In the calculateProbability() function we calculate the exponent first, then calculate the main division. This lets us fit the equation nicely on two lines.

2. Calculate Class Probabilities

We combine probabilities together by multiplying them. In the calculateClassProbabilities() below, the probability of a given data instance is calculated by multiplying together the attribute probabilities for each class. the result is a map of class values to probabilities.

1. Make Predictions

Finally, we can estimate the accuracy of the model by making predictions for each data instance in our test dataset. The getPredictions() will do this and return a list of predictions for each test instance.

1. Get Accuracy

The predictions can be compared to the class values in the test dataset and a classification accuracy can be calculated as an accuracy ratio between 0& and 100%. The value which is higher means that the possibility is bigger for man or women.

Our Proposed Naive Bayes mining algorithms

In this section, we proposed Naive Bayes algorithm for the discovery of possibility patterns.

1.// Load csv file

void loadCsv(filename) {

dataSet <- csv.reader(filename)

for (each line in dataSet) {

if (gender == 'm')

genderInt = MALE

if (gender == 'f')

genderInt = FEMALE

row <- [genderInt, str]

newDataSet.append(row)

}

return newDataSet

}// End of loadCsv

2.//Training the Naive Bayes model on data set

void trainModelWith(train\_dataSet){

rows[], index[]

count <- 0

//load example data in a format we can feed to the learning algorithm

for (each data in train\_dataSet) {

rows.append({'text': data[1], 'gender': data[0]})

index.append(count)

count += 1

}

// learn the vocabulary of the corpus and extracts word count features.

// DataFrame is a sql-like data structure in Pandas library

dataFrame <- DataFrame(rows, index <- index)

beginTime <- time.clock()

counts <- countVec.fit\_transform(dataFrame['text'].values)

targets <- dataFrame['gender'].values

// train by calling fit

classifier.fit(counts, targets)

endTime <- time.clock()

print <- " Training completed in {0:.3f} ms.\n".format((endTime-beginTime))

}// End of trainModelWith

3.// Performing gender classification on testing data

void predictUsing(test\_dataSet) {

texts [], actualGender[]

for (each line of testDataSet) {

actualGender.append(test\_dataSet[i][0])

texts.append(test\_dataSet[i][1])

}

beginTime <- time.clock()

test\_counts <- countVec.transform(texts)

probability <- classifier.predict\_proba(test\_counts)

endTime <- time.clock()

print <- "> Prediction completed in {0:.3f} ms.\n".format((endTime-beginTime))

printResults(probability, texts, actualGender)

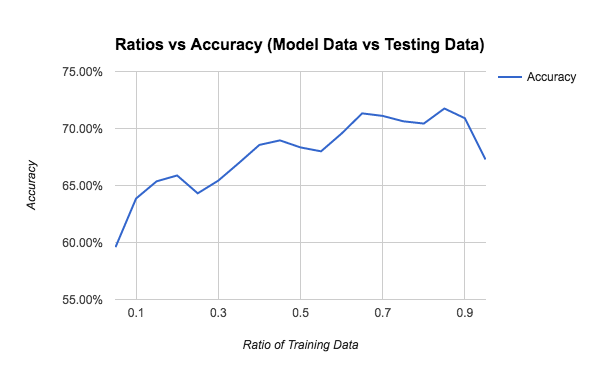
}// End of predictUsing

**4. Analytical and Empirical Evaluation**

We tested the Naive Bayes algorithm using example data from Twitter (Crowdflower, 2016). We acquired a csv file and modified it to contain the gender of the Twitter user, plus the text they wrote. Our file contains 3227 individual tweets. This file was imported into our gender predictor program. From there, the program has within a set ratio, which is a floating number representing a percentage, with a range from 0 < r <= 1.

We create a split size based off the ratio using the formula len(dataSet) \* r, when len(dataSet) is the number of entries in our example data. It is converted into an integer by truncating the decimals. The program takes the generated split point and takes the entries from our example data from 0 to split size and sends it through our training algorithm as discussed above.

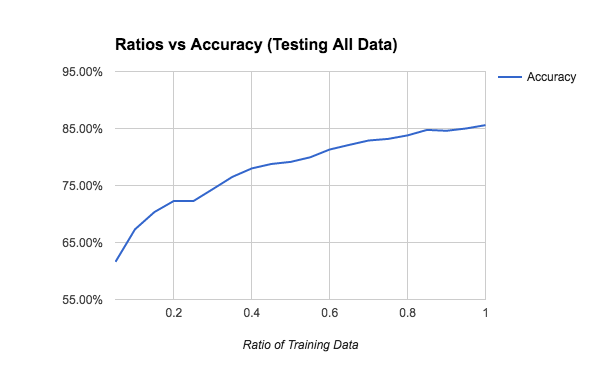
We performed two different tests with our example data. The first test has a strict separation between our model data and our testing data, that is, we don’t test the data we used to generate the model. In the other test, we still only model the first r% of our example data, but tested all of the data, including the data we used to model the Bayes model.

For the first test, we tested different ratios from 0.05 <= r <= 0.95 at 0.05 increments. As r gets bigger, the model data size increases, and the testing data size decreases. A graph of accurate predictions as a percentage with respect to the ratio of training data is below. In this test, the accuracy of our predictions peaked just above 70% from 0.65 <= r <= 0.9. The drop of accuracy at r = 0.95 can be attributed to a lack of testing data. Even at r = 0.05, it was able to accurately predict just under 60 % of the test data, while getting to a maximum accuracy of just above 70% at r = 0.65 and sustaining that for the higher ratios.

In our second test, we tested ratios from 0.05 <= r <= 1.0, also at 0.05 increments. We are able to test the 1.0 ratio because we are testing all the data regardless. Again, a graph showing the percentage of accurate predictions versus the ratio of model data is below. The rise of accuracy is more stable in this test, seeing fewer and lesser decreases in some ratio increments that are barely noticeable from the graph. Accuracies are also higher for every ratio compared to the previous test. This is expected as we also tested our model size, that size of which increases for higher ratios.

Another point to note is that the difference in accuracy for each ratio between the two tests increases as the ratio increases. For example, the accuracy at r = 0.05 is about 61% in the second test versus 59.5% in the first. The difference increases at r=0.9, where the second test shows 85% accuracy vs around 70% for the first test.

One more thing of note. Even at a ratio of r = 1.0, the model was only accurate 85% of the time, which seems counter intuitive when all of our testing data is also our model data.



In general, the second test had more accurate predictions than the first. Obviously using model data as part of the test data increased accuracy. If we had bigger data sets and used the Naive Bayes model, both similar to the one we tested with, the pattern that we have seen in both tests would probably be more pronounced.

**5. Conclusion**

Just by using the Naive Bayes algorithm and our example data, we were able to accurately predict the gender of the commenter at least 70% of the time when we use most of the data as the model. It got more accurate when we used all the example data as our model, getting up to 85% accuracy.

In real life, this can be used for knowing the amount of male and female users while not knowing their gender in advance. The owner of the website can do something such as: Give out something that is more attractive for female users while the amount of female users is much larger than that of male users that month.

There are two things we can do in the future to confirm our observations. The first is to train using a bigger example data set. This would provide more data to test with at the end of our data set to better understand if our general observations hold for r= 0.95 in the first test.

Another idea we have is if we have an independent model based on words most likely to be used by each gender. We might gain this insight from psychological research into gender differences. We could take the data from such research, generate a model, and run it through our example data, comparing the results to our tests. This would provide an independent assessment of our tests.

**6. Citations/References**

1. Bernhard,S., Leung,C., Reimer,V., Westlake,J. Clickstream Prediction Using Sequential Stream Mining Techniques with Markov Chains. Retrieved from University of Manitoba, Winnipeg, MB, Canada
2. Leung,C., Jang,F., Pazdor,A., Peddle,A. Parallel social network mining for interesting ‘following’ patterns. Retrieved from University of Manitoba, Winnipeg Canada
3. R, SUNIL. (2015, SEPTEMBER 13). 6 Easy Steps to Learn Naive Bayes Algorithm (with code in Python). Retrieved from https://www.analyticsvidhya.com/blog/2015/09/naive-bayes-explained/
4. Brownlee, J. (2014, December 8). How To Implement Naive Bayes From Scratch in Python. Retrieved from http://machinelearningmastery.com/naive-bayes-classifier-scratch-python/
5. Volkova, S. & Yarowsky, D. (2012). Improving Gender Prediction of Social Media Users via Weighted Annotator Rationales. Retrieved from <http://www.cs.jhu.edu/~svitlana/papers/VY-NIPSPersonalization14.pdf>
6. Crowdflower (2016). Twitter User Gender Classification. Retrieved from <https://www.kaggle.com/crowdflower/twitter-user-gender-classification>