Universiteit Antwerpen

Assignment 2 Data Mining

Classification/Calibration

By
Lore Hilgert

2020-2021

Contents

1 Introduction

This project handles the issues when one wants to detect spam e-mails. The company desires a most flexible system to determine whether an e-mail is spam or not. For example, the CEO absolutely does not want to deal with spam, so the spam filter should be set such that any incoming e-mail has a low probability of being spam. However, the inquiry and sales department does not want to miss important e-mails, for this reason they are far more tolerant to spam.

If a classifier is chosen, it should give a probability of being spam for each incoming mail. The company can then decide to set a threshold. This threshold determines whether a mail directly goes to trash or not, f.e. the threshold (probability of being spam before throwing away) will be lower for the CEO and higher for the sales department. Therefore, a Naive Bayes classifier will be chosen for this project. Moreover in the first section.

A second task will be to calibrate the given outcome such that these probabilities are reliable. To solve this spam-ham issue, `Python` will be used, accompanied with `scikit-learn`.

2 Classification

The dataset that was given for this assignment contains a train and a test set. The train set will be used to train the chosen classifier. To work on this train set properly, the .txt files were added to a .csv file (namely `train.csv`). This .csv file gives a document with two columns: the label (spam or ham) at the left and the e-mail text at the right. The code for this conversion is found in https://github.com/LorHil/Classification-Calibration. If one wants to try out different classifiers, the training dataset should be subdivided into two parts: train and validate. Another reason for splitting the dataset is to evaluate the learned model and calculate its accuracy.

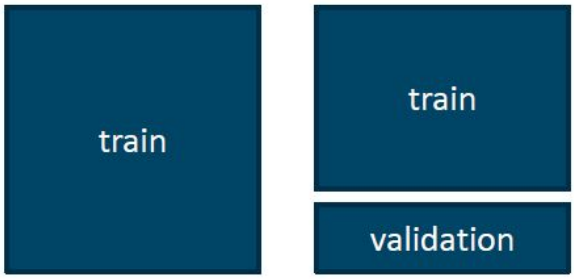

Figure 2.1: Dividing the training dataset, from [3]

Given that the outcome of the classifier should provide the user with a probability (of being spam), a Naïve Bayes Classifier will be used. At this moment in time, we will thus train a Naïve Bayes classifier using the train dataset. There is a need of subdividing the training dataset as in figure 2.1, providing us with a validation set that is used to evaluate the learned classifier.

2.1 Methodology Naïve Bayes Classifier

The Naïve Bayes is based on the following formula:

$$P(C|A_1A_2A_3...A_n) = \frac{P(A_1A_2A_3...A_n|C)P(C)}{P(A_1A_2A_3...A_n)}$$

With A_i the given attributes and C the class. In this example, the class C can only be ham or spam. Learning the model (classifier) is a process that begins with estimating $P(C)$. When looking at the validate dataset (see further on), we estimate $P(Spam) = 0.6$. Then for every attribute A_i, the probability $P(A_i = i|C)$ should be learned. This is actually just counting how many times that attribute occurs in that class.

Recall that we assume class independency such that $P(A_1 = a, A_2 = b|C) = P(A_1 = a|C) \cdot P(A_2 = b|C)$.

There are different types of Naïve Bayes classifiers:

- Categorical
- Multinomial
- Bernoulli
- Gaussian

For this project, the **Multinomial Naïve Bayes classifier** can be used because it counts for example the occurences of words in a text. Another option is to utilise the Bernoulli Naïve Bayes, but this will only detect an absence or presence of words. Given the great amount of words that can be used in a spam e-mail, the Bernoulli classifier will not give optimal outcomes.

2.2 Implementation and analysis

2.2.1 Preprocessing data

Before one can start working with a dataset, the structure of this dataset needs to be as desired. Here, the goal is to provide the classifier with a .csv file where The data should be preprocessed. Important for text files is that all words that appear are in lower case. Also, stemming of the words is a nice to have when one does not want to take into account all derivatives of the roots of words. Therefore, this simplification of the messages (see `DatasetAdjustment.py`) is done.

After this, the training dataset is split into a train and validate set. The validate set is 10 times smaller than the new training set. This step is implemented in `Project.py` (see GitHub repository) as

```
x_train, x_validate, y_train, y_validate
= train_test_split(features, Y_train, test_size=0.1, random_state=80)
```

Where `x_train` contains the messages of all mails and `y_train` respectively the labels (ham or spam). The `validate` data is used to evaluate the classifier, implemented in the following section.

2.2.2 Implementation classifier

For this project, the Multinomial Naïve Bayes classifier is used. This for the reason that this type of classifier is one of the two classic Naïve Bayes variants used in text classification, where the data are typically represented as word vector counts. This is also what happens here. The method `CountVectorizer()` is used to provide the classifier with words as vector counts. Naïve Bayes relies upon the following formula:

$$P(C|A_1A_2A_3...A_n) = \frac{P(A_1A_2A_3...A_n|C)P(C)}{P(A_1A_2A_3...A_n)}$$

For the Multinomial variant, the distribution is parametrized by vectors $\vartheta_y = (\vartheta_{y1}, ..., \vartheta_{yn})$ for each class y, where n is the number of features (here thus the size of the vocabulary) and ϑ_{yn} is the probability P(x_i|y). The parameters ϑ_y of the distribution are estimated by a smoothed version of maximum likelihood:

$$\vartheta_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

where N_{yi} is the number of times the feature i appears in a sample of class y in the training set, and N_y is the total count of all features for class y. This information was found in [2] and [3]. The default Multinomial NB classifier in the library `scikit-learn` has an $\alpha = 1$. This means that Laplace smoothing is done. If α would be zero, the multiplication of the $P(A_i|C)$ for all i would become zero if only one of the conditional probabilities is zero [3]. To avoid this issue, one utilizes Laplace smoothing.

Given this information on the classifier, this seems the most opportune classifier for distinguishing spam from non-spam.

The model is learned based on the train data:

```
model = MultinomialNB()
model.fit(x_train, y_train)
```

Then, this model can be applied on the validate dataset, where the labels are known but not given to the model:

```
prob_pos_model = model.predict_proba(x_validate)[:,1]
```

The method `predict_proba()` returns probability estimates for each class (here for the spam and ham class). Taking the second column, only the probabilities of being spam are saved. Now, after predicting the probabilities of the messages in the validation dataset, one can compare them with the true class labels in `y_validate`. The evaluation of the classifier is done in the following section. All code concerning this topic is also to be found in the file `Project.py`.

2.3 Evaluation of the classifier

Previously, the assumption of class independency was made. Even if there are dependencies between the attributes, Naïve Bayes works quite well. But be aware of the probability estimation that Naïve Bayes gives us. This estimation may be quite off. The total score that the classifier computes means f.e. higher score is higher probability, but the computed probability can be not really correct or relevant. Usually the probabilities are an overestimation of the real probabilities. Keep this in mind for when we test the validate dataset.

The model is learned on the train dataset and validated on a part of this dataset. When using the above implemented classifier, one gets the outcome (from `project.py`) that the accuracy is 0.7428. This is not disastrous, but a better accuracy would be preferable. Not only the accuracy of a classifier is important. Therefore, the confusion matrix is requested:

		Predicted class	
		Ham	Spam
Actual class	Ham	110	29
	Spam	61	150

Figure 2.2: Confusion matrix without calibration

From the matrix above, one can derive the accuracy (as given before) and the recall:

$$acc = \frac{TP + TN}{TP + FP + TN + FN} = \frac{150 + 110}{150 + 29 + 110 + 61} = 0.74$$

$$recall = \frac{TP}{TP + FN} = \frac{150}{211}$$

One can evaluate whether the classifier's probabilistic predictions are well calibrated by using the Brier score[1]. The probabilities assigned to this set of outcomes must sum to one (where each individual probability is in the range of 0 to 1). The Brier score measures the mean squared difference between the predicted probability assigned to the possible outcomes for an instance and the actual outcome. Therefore, the lower the Brier score is for a set of predictions, the better the predictions are calibrated. The Brier score can be expressed as follows:

$$BS = \frac{1}{N} \sum_{t=1}^{N} (y_t - o_t)^2$$

[1]The Brier Score is a strictly proper score function or strictly proper scoring rule that measures the accuracy of probabilistic predictions [1].

Where y_t is the predicted outcome and o_t are the real labels.

On the graph in figure 2.3, one can find the probability predictions for the validate dataset (composed out of 350 e-mails to be labelled). The score given in the left uppercorner is the Brier score. One should thus opt for an as low as possible Brier score.

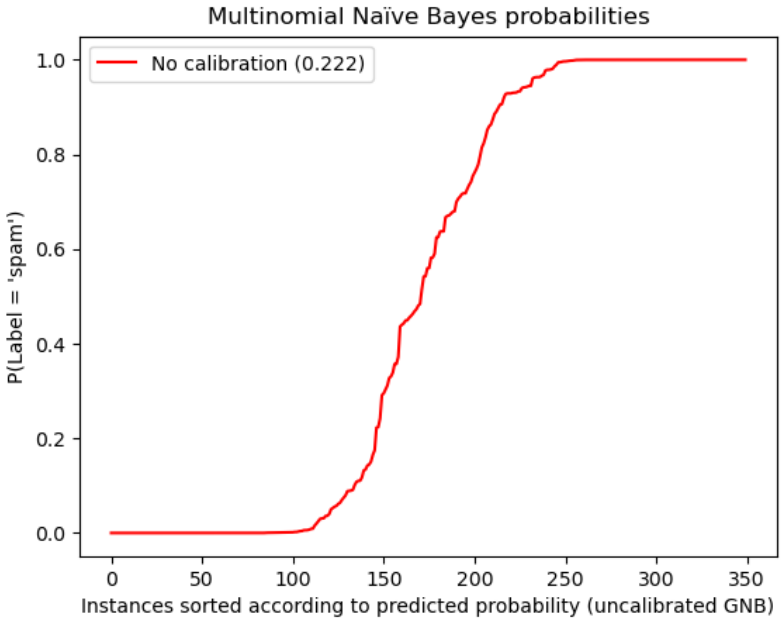

Figure 2.3: Without calibration

As mentioned before, the $P(\text{Spam})$ is estimated as 0.6. In the graph above, the classifier does clearly NOT return probabilities close to this estimated 0.6. This implies that a proper calibration should be done, in order to provide the company we are working for with some relevant data.

3 Calibration

Calibration is a type of postprocessing. Previous sections already mentioned why this calibration is needed for this application. The following section will explain how these better results as in the confusion matrix in figure 3.1 are obtained.

		Predicted class	
		Ham	Spam
Actual class	Ham	84	55
	Spam	25	186

Figure 3.1: Confusion matrix with calibration

3.1 Postprocessing

The calibration of the classifier is done using Isotonic regression, a binary approach [4]. This is one of the two most popular calibration method for probabilities. With the ROC-convex hull method, isotonic regression can give good calibration performance with automatic binning and interpolation [4]. The second one is called Sigmoid Scaling.

If a function is monotonic increasing, the Isotonic regression is not constrained by any functional form. The following quadratic equation explains the Isotonic regression problem:

$$y_i = m(x_i) + \epsilon_i$$

where m is an Isotonic function. Now given the training set $[x_i, y_i]$, the goal is to find the the Isotonic function m by minimizing the following equation:

$$m = argmin_z \sum (y_i - z(x_i))^2$$

Keep in mind that Isotonic regression quite prone is to overfitting. Using an sufficiently elaborated dataset, this issue will not or will only slightly come to light.

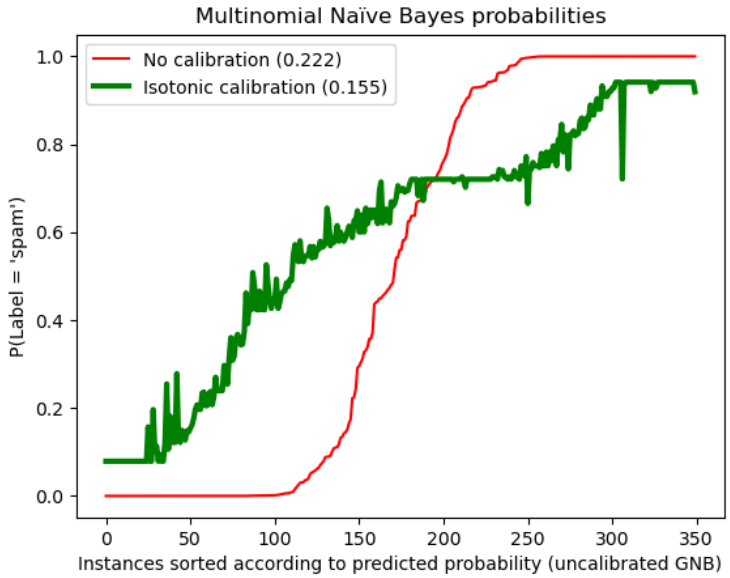

Figure 3.2: With non-parametric Isotonic calibration

This calibration method indeed is the best out of the two. Sigmoid scaling was not used here because of its higher Brier score (namely, 0.175) and the form of the graph in figure 3.3.

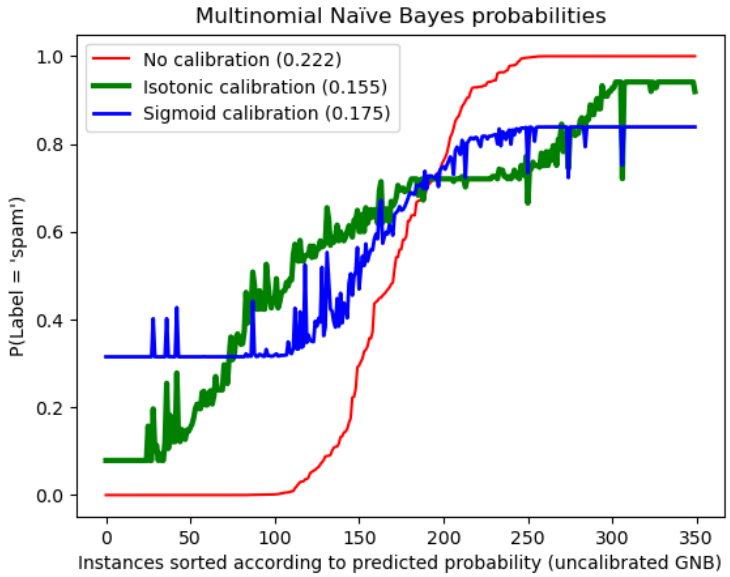

Figure 3.3: With different calibrations

Another way of looking at the calibration is through the following graph:

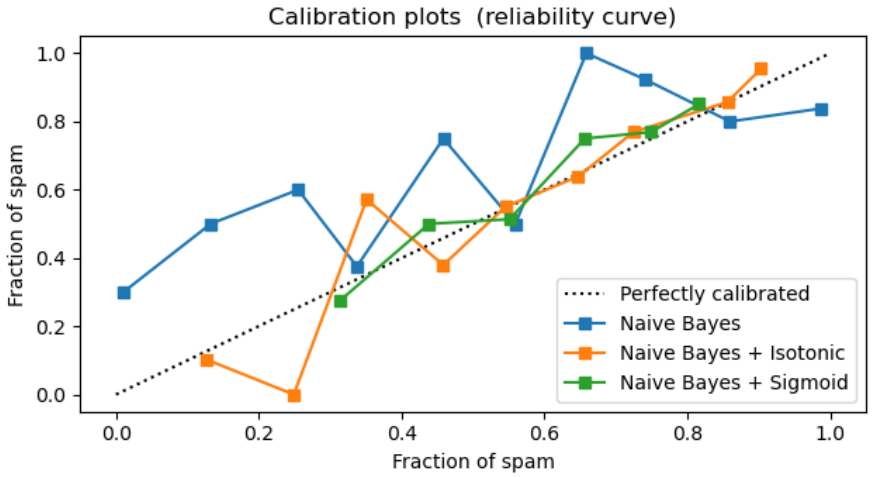

Figure 3.4: Reliability plot, code from [7]

Here, it is proven that Isotonic regression indeed lies closer to the "perfectly calibrated"-line. Nevertheless, one can see that for low fractions of spam, none of the above solutions really acts well. The most important goal in this project is that e-mails with a relative high probability of being spam are filtered. This threshold on the probability depends on the services an employee must deliver. For the CEO, this threshold will be lower than for the sales department.

4 End result and conclusion

Now, when the model is learned and calibrated, one can apply the classifier on unknown data (here to be found in the `test.csv` file in the GitHub repository). Our system will provide the company with the probabilities that incoming e-mails are spam. The file `scores.txt` are the respectively placed probabilities of being spam. The first line corresponds with the first e-mail in the test dataset.

For validation of the model, some manual verification on the outcome of the system was done. For example, the first e-mail has a rather high probability of being spam (0.76), and this is indeed a quite weird-looking e-mail. I repeated this step for more than thirty e-mails. The final file `scores.txt` can be analysed by the company as follows: if the company reads an e-mail, they have a cost of x, but reading a non-spam e-mail, the company earns y. If the chance of being spam is given by p and the threshold chosen by the company is t, then pouring this into a cost formula, this gives:

`for p > t`:

$$cost = (1 - p) \cdot y$$

`for p` $\leq$ `t`:

$$cost = x - (1 - p) \cdot y$$

Indeed, if the probability of being spam p is larger than the threshold, the mail is NOT read and the company throws away (1-p) probability that it was a non-spam mail. This means that the company looses y with a probability of (1-p). If the mail is read (meaning $p \leq t$), the company looses x because of the reading, but earns y if and only if the mail was indeed not-spam.

By minimizing this cost function, the company can find an optimal threshold t, which corresponds with the probability of spam wherefore the company thinks it wants to throw away the e-mail. In my opinion, it is better to keep the threshold quite low and to seek for yourself if the e-mail was a spam e-mail or not. Either way, a CEO that receives thousands of mails per day, would like to receive as little as possible unnecessary e-mails.

References

[1] Wikipedia Contributors. Brier Score. Retrieved from *https://en.wikipedia.org/wiki/Brier_score* on 21/04/2021

[2] Scikit-learn developers. sklearn.naive_bayes.MultinomialNB. Retrieved from *https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB* on 22/04/2021

[3] T. Calders, B. Goethals. Slide deck Data Mining: Classification. March 2021.

[4] Peter Flach, Miquel Perello-Nieto, Hao Song, Meelis Kull, Telmo Silva Filho. A tutorial at ECML-PKDD 2020. Retrieved from *https://classifier-calibration.github.io/* on 15/04/2021. September 2020.

[5] Jason Wills Chui. Spam Filtering with Machine Learning using the Naive Bayes Algorithm, retrieved from *https://jasonwillschiu.github.io/blog/2017-12-11-NaiveBayesSpam/* on 13/04/2021

[6] Daniyal Shahrokhian. The Naive Bayes Algorithm in Python with Scikit-Learn, retrieved from *https://stackabuse.com/the-naive-bayes-algorithm-in-python-with-scikit-learn/* on 23/04/2021

[7] Scikit-learn. Probability Calibration curves. Retrieved from *https://scikit-learn.org/stable/auto_examples/calibration/plot_calibration_curve.htmlsphx-glr-auto-examples-calibration-plot-calibration-curve-py* on 19/04/2021