COMPSCI 589 Homework 2 - Spring 2024

By Noshitha Padma Pratyusha Juttu

1 Notes on submission:

- The dataset for test and train folders needs to be downloaded in the same folder as the python files
- Instructions to run the classifier is :
  - python3 utils.py
  - python3 naivebayes.py
  - For Question 1 and Question 2:
    - python3 runq1q2.py
  - For Question 3
    - python3 runq3.py
  - For Question 4
    - python3 runq4.py
  - For Question 6
    - python3 runq6.py
- The output will be the metrics - accuracy, precision , recall and confusion matrix (heat map of confusion matrix)

2 The IMDB Dataset of Movie Reviews - Questions

Q.1 (18 Points) You will first run an experiment to evaluate how the performance of the Naive Bayes algorithm is affected depending on whether *(i)* you classify instances by computing the posterior probability, $\Pr(y_i \mid Doc)$, according to the standard equation (Eq. (**??**)); or *(ii)* you classify instances by performing the log-transformation trick discussed in class and compare log-probabilities, $\log(\Pr(y_i \mid Doc))$, instead of probabilities. Recall, in particular, that estimating the posterior probability using Eq. (**??**) might cause numerical issues/instability since it requires computing the product of (possibly) hundreds of small terms—which makes this probability estimate rapidly approach zero. To tackle this problem, we often compare not $\Pr(y_1 \mid Doc)$ and $\Pr(y_2 \mid Doc)$, but the corresponding log-probabilities: $\log(\Pr(y_1 \mid Doc))$ and $\log(\Pr(y_2 \mid Doc))$. Taking the logarithm of such probabilities transforms the product of hundreds of terms into the sum of hundreds of terms—which avoids numerical issues. Importantly, it does not change which class is more likely according to the trained model. When classifying a new instance, the log-probabilities that should be compared, for each class y_i, are as follows:

$$\log(\Pr(y_i \mid Doc)) = \log\left(\Pr(y_i) \prod_{k=1}^{len(Doc)} \Pr(w_k \mid y_i)\right) \quad (1)$$

$$= \log(\Pr(y_i)) + \sum_{k=1}^{len(Doc)} \log(\Pr(w_k \mid y_i)). \quad (2)$$

In this experiment, you should use 20% of the training set and 20% of the test set; *i.e.*, call the dataset-loading functions by passing 0.2 as their parameters. First, perform the classification of the instances in the test set by comparing posterior probabilities, $\Pr(y_i \mid Doc)$, according to Eq. (**??**), for both classes. Then, report *(i)* the accuracy of your model; *(ii)* its precision; *(iii)* its recall; and *(iv)* the confusion matrix resulting from this experiment. Now repeat the same experiment above but classify the instances in the test set by comparing log-probabilities, $\log(\Pr(y_i \mid Doc))$, according to Eq. (2), for both classes. Report the same quantities as before. Discuss whether classifying instances by computing log-probabilities, instead of probabilities, affects the model's performance. Assuming that this transformation does have an impact on performance, does it affect more strongly the model's accuracy, precision, or recall? Why do you think that is the case?

Answer:

Evaluation using posterior probabilties without Laplace Smoothing:
Accuracy: 0.523
Precision: 0.732
Recall: 0.055
Confusion Matrix:
'true positive': 139, 'true negative': 2531, 'false positive': 51, 'false negative': 2383

Evaluation using Log Probabilities without Laplace Smoothing:
Accuracy: 0.732
Precision: 0.744
Recall: 0.698
Confusion Matrix: 'true positive': 1761, 'true negative': 1975, 'false positive': 607, 'false negative': 761

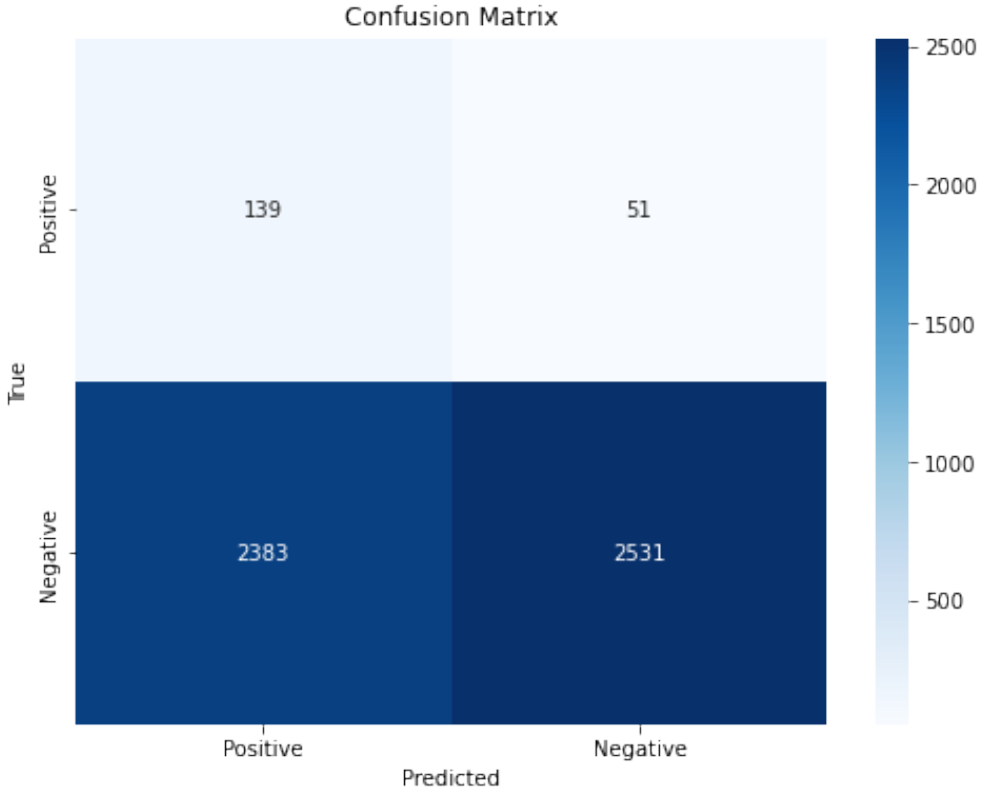

Figure 1: Confusion matrix heat map for posterior probability.

- Comparing the two approaches, we observe a substantial improvement in performance when using **log-probabilities.**
- The accuracy increases from 0.523 to 0.732, precision increases from 0.732 to 0.744, and recall increases from 0.055 to 0.698.
- This logarithmic transformation positively affects accuracy, precision, and recall, with a more noticeable impact on recall. It means we're now much better at catching instances of the positive class, reducing false negatives. This adjustment strikes a balance between precision and recall, delivering an overall performance boost.
- The log-probabilities tackle the numerical instability issues in standard Naive Bayes. By converting products of small probabilities into sums of logarithms, we avoid rapid approaches to zero, ensuring a more stable and effective model.

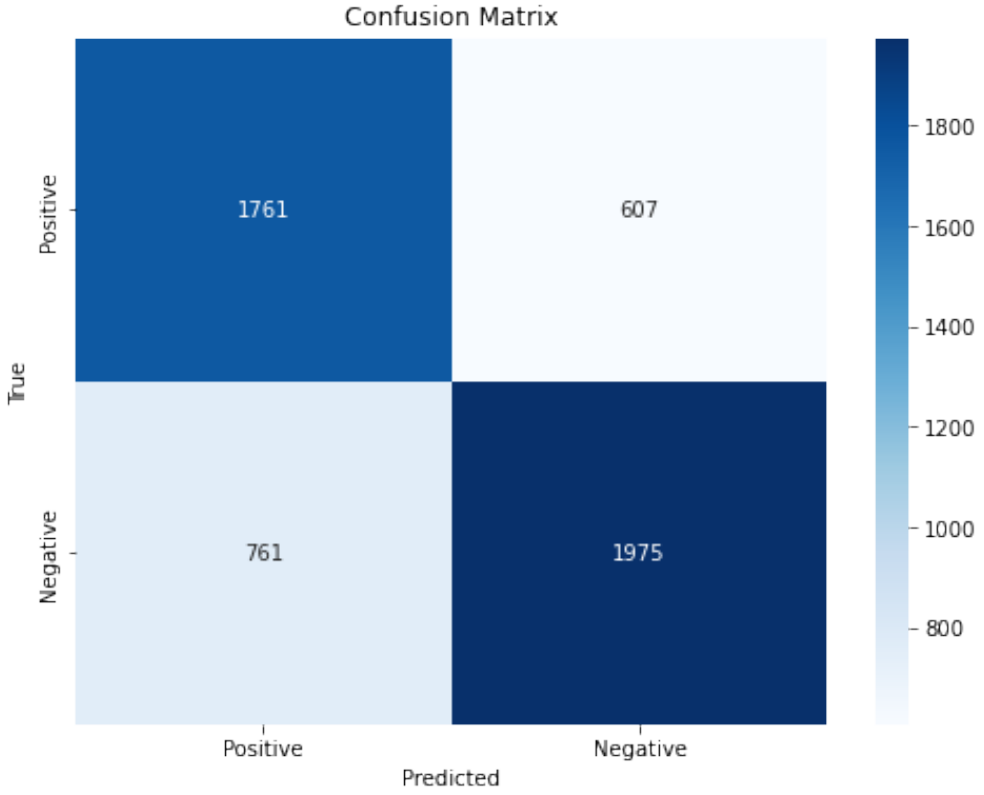

Figure 2: Confusion matrix heat map for logarithmic probability

Q.2 (18 Points) An issue with the original Naive Bayes formulation is that if a test instance contains a word that is not present in the vocabulary identified during training, then $\Pr(word|label) = 0$. To mitigate this issue, one solution is to employ Laplace Smoothing. To do so, as discussed in class, we replace the standard way of estimating the probability of a word w_k, given a class y_i, with the following equation:

$$\Pr\left(w_k \mid y_i\right) = \frac{n(w_k, y_i) + 1}{\sum_{s=1}^{|V|} n(w_s, y_i) + |V|}. \tag{3}$$

More generally, Laplace Smoothing can be performed according to a parametric equation, where instead of adding 1 to the numerator, we adjust the probability of a word belonging to a class by adding a user-defined parameter α to the numerator, as follows:

$$\Pr\left(w_k \mid y_i\right) = \frac{n(w_k, y_i) + \alpha}{\sum_{s=1}^{|V|} n(w_s, y_i) + \alpha|V|}. \tag{4}$$

Intuitively, setting $\alpha = 0$ results in the standard formulation of Naive Bayes—which does not tackle the problem of words that do not appear in the training set. Suppose, alternatively, that we set $\alpha = 4$. This is equivalent to adding 4 "fake" occurrences of that word to the training set, in order to avoid the zero-frequency problem. Using $\alpha = 1000$, on the other hand, is equivalent to pretending we have seen that word 1000 times in the training set—even though we may have seen it, say, only 8

times. Although this solves the problem of zero-frequency words, it also strongly biases the model to "believe" that that word appears much more frequently than it actually does; and this could make the predictions made by the system less accurate. For these reasons, although it is important/necessary to perform Laplace Smoothing, we have to carefully pick the value of α that works best for our dataset. Using $\alpha = 1$ is common, but other values might result in better performance, depending on the dataset being analyzed.

In this experiment, you should use 20% of the training set and 20% of the test set; *i.e.*, call the dataset-loading functions by passing 0.2 as their parameters. You should first report the confusion matrix, precision, recall, and accuracy of your classifier (when evaluated on the test set) when using $\alpha = 1$. Now, vary the value of α from 0.0001 to 1000, by multiplying α with 10 each time. That is, try values of α equal to 0.0001, 0.001, 0.01, 0.1, 1.0, 100, and 1000. For each value, record the accuracy of the resulting model when evaluated on the test set. Then, create a plot of the model's accuracy on the test set (shown on the y-axis) as a function of the value of α (shown on the x-axis). The x-axis should represent α values and use a log scale. Analyze this graph and discuss why do you think the accuracy suffers when α is too high or too low.

Answer:
Evaluation with Laplace Smoothing:
Evaluation with α ***= 1***:
Accuracy: 0.822
Precision: 0.863
Recall: 0.822
Confusion Matrix:
'true positive': 1918, 'true negative': 2278, 'false positive': 304, 'false negative': 604

Discussion:

- The curve is showing lower accuracy for small and large values of α, with peak accuracy around $\alpha = 1$.
- The choice of α is a trade-off between overly aggressive smoothing and insufficient smoothing.
- Middle values of α strike this balance, resulting in better model performance on both seen and unseen data.
- At $\alpha = 1$, this balance is achieved for this model.
- Lower and higher values of α, on the other hand, tend to lead to sub-optimal smoothing effects, impacting the accuracy of the Naive Bayes classifier.

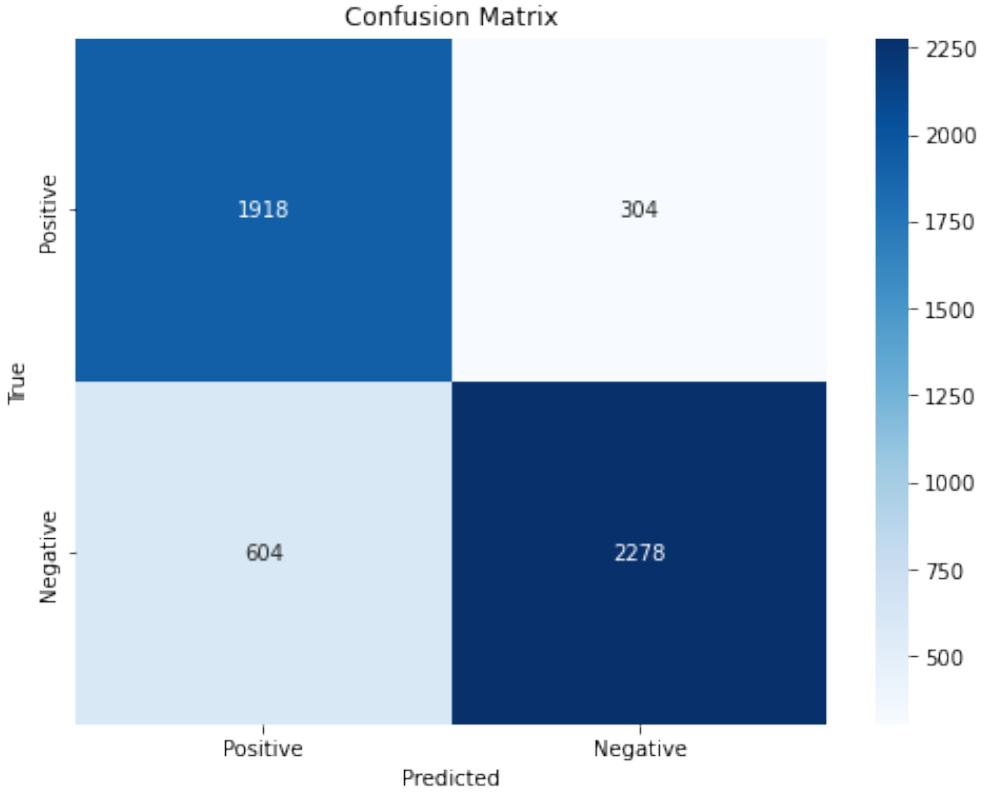

Figure 3: Confusion matrix heat map for α=1

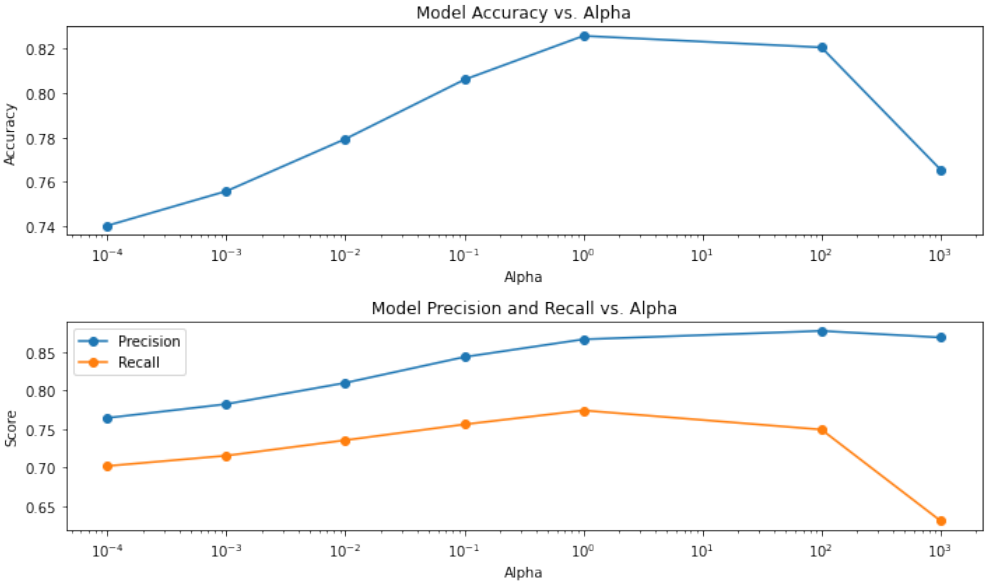

Figure 4: Model accuracy vs alpha

Q.3 (18 Points) Now you will investigate the impact of the training set size on the performance of the model. The classification of new instances, here, should be done by comparing the posterior log-probabilities, $\log(\Pr(y_i \mid Doc))$, according to Eq. (2), for both classes. You should use the value of α that resulted in the highest accuracy according to your experiments in the previous question. In this question, you should use 100% of the training set and 100% of the test set; *i.e.*, call the dataset-loading functions by passing 1.0 as their parameters. Then, report *(i)* the accuracy of your model; *(ii)* its precision; *(iii)* its recall; and *(iv)* the confusion matrix resulting from this experiment.

Answer:

Accuracy: 0.824
Precision: 0.863
Recall: 0.771
Confusion Matrix:
'true positive': 9632, 'true negative':10975 , 'false positive': 1525, 'false negative': 2868

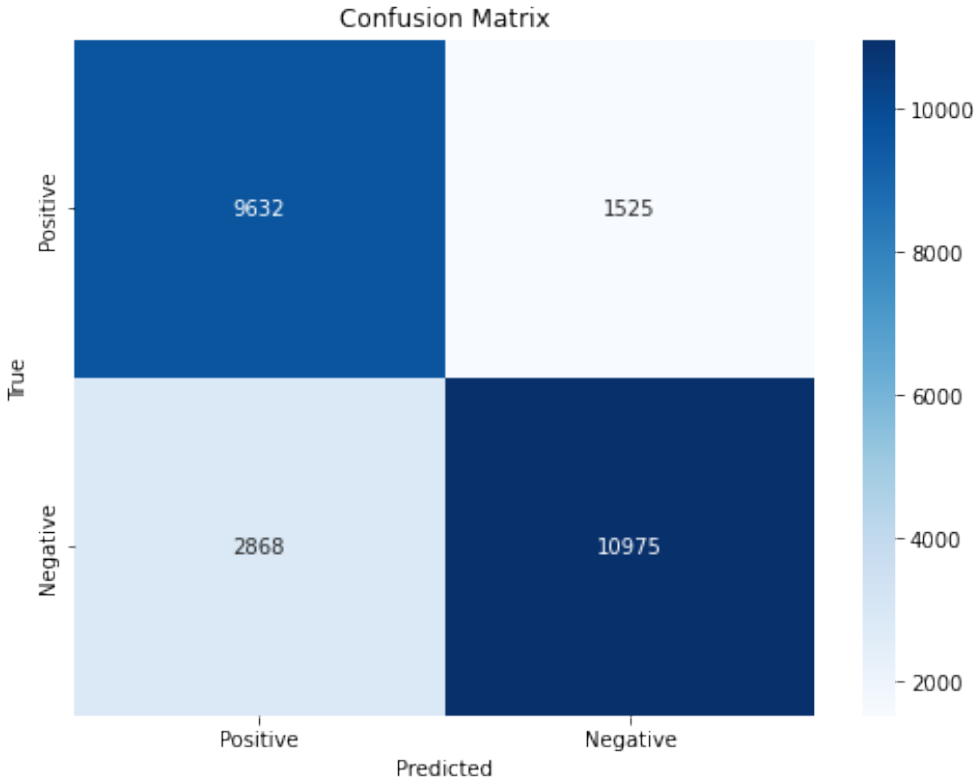

Figure 5: Confusion matrix heat map using 100 percent of training and test sets

Q.4 (18 Points) Now repeat the experiment above but use only 50% of the training instances; that is, load the training set by calling `load_training_set(0.5, 0.5)`. *The entire test set should be used.* Report the same quantities as in the previous question. Discuss whether using such a smaller training set had any impact on the performance of your learned model. Analyze the confusion matrices (of this question and the previous one) and discuss whether one particular class was more affected by changing the size of the training set.

Answer: Accuracy: 0.820
Precision: 0.863
Recall: 0.761
Confusion Matrix:
'true positive': 9731, 'true negative': 10933, 'false positive': 1567, 'false negative': 2769

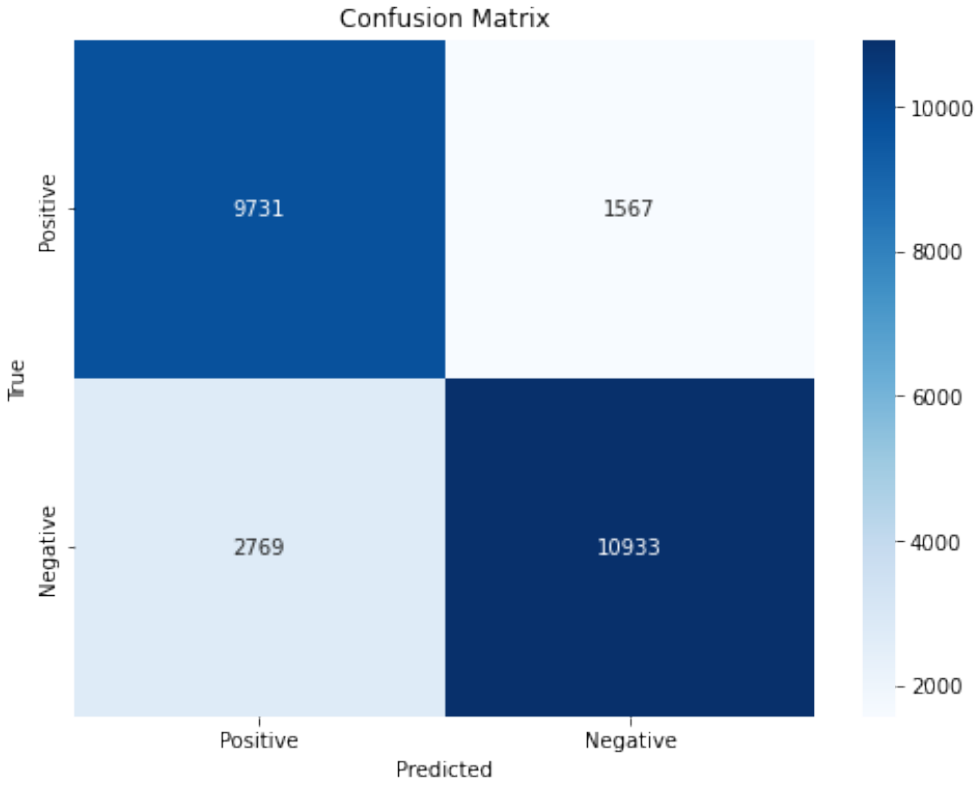

Figure 6: Confusion matrix heat map using 50 percent of training set

Comparison with Q.3: Accuracy Change: -0.004 Precision Change: -0.001 Recall Change: -0.009

- The reduction in performance with a smaller training set is primarily due to the model's limited exposure to diverse examples, resulting in a less effective learning process.
- The impact is particularly noticeable in instances where the model needs to generalize from insufficient information, leading to an increase in classifications, especially false negatives.

Q.5 (10 Points) In this application (i.e., accurately classifying movie reviews), would you say that it is more important to have high accuracy, high precision, or high recall? Justify your opinion. **Answer:**

- In my opinion, high precision makes more impact.
- Most of them people take time to see movie and they check for the positive reviews.
- Therefore the priority of the model would be to avoid incorrectly labelling a negative review as positive which needs higher precision.
- Therefore, the goal to minimize the false positives would take a priority.

Q.6 (18 Points) Finally, you will study how the performance of the learned model is affected by training it using an unbalanced dataset (*i.e.*, a dataset with significantly more examples of one of the classes). The classification of new instances, here, should be done by comparing the posterior log-probabilities, $\log(\Pr(y_i \mid Doc))$, according to Eq. (2), for both classes. You should use the value of α that resulted in the highest accuracy according to your experiments in the previous questions. You will now conduct an experiment where you use only 10% of the available *positive* training instances and that uses 50% of the available *negative* training instances. That is, use `load_training_set(0.1, 0.5)`. *The entire test set should be used.* Show the confusion matrix of your trained model, as well as its accuracy, precision, and recall. Compare this model's performance to the performance (according to these same metrics) of the model trained in question Q.4—that is, a model that was trained under a *balanced* dataset. Discuss how training under an unbalanced dataset affected each of these performance metrics.

Answer:
Accuracy: 0.509
Precision: 0.954
Recall: 0.018
Confusion Matrix:
'true positive': 310, 'true negative': 12190, 'false positive': 4, 'false negative': 12496

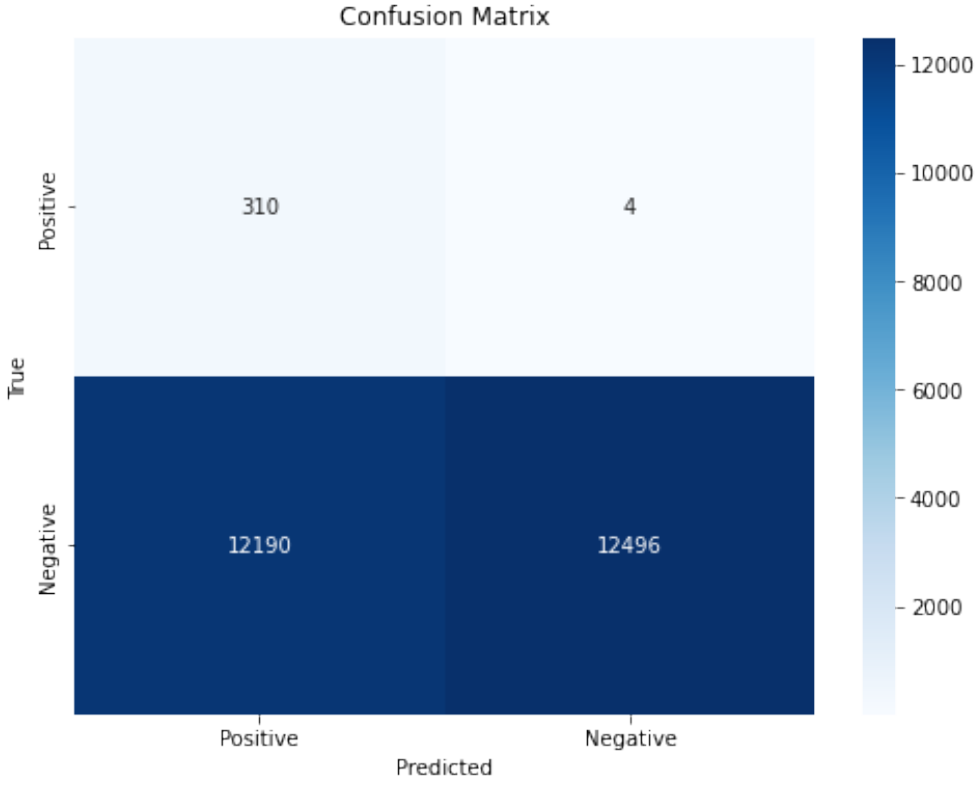

Figure 7: Confusion matrix heat map for the unbalanced dataset

Comparison with Q.4: Accuracy Change: -0.311 Precision Change: 0.092 Recall Change: -0.743

- Training with an unbalanced dataset creates a model that is skewed toward the more prevalent class, impacting its ability to generalize and predict instances of the minority class accurately.

- The biases introduced during training affect accuracy, precision, and recall, with a notable reduction in recall due to the model's difficulty in identifying positive instances.