CS 529 Project 2

Using Naive Bayes and Logistic Regression to Categorize Articles

Mike Adams and Jack Ringer

March 2023

GitHub Repository: https://github.com/Jack-42/cs529_project2

1 Code summary

We implemented Naive Bayes and logistic regression classifiers for the 20 newsgroups dataset. The documents were provided in a Bag of Words format such that each feature corresponded to the count of a specific word present in a document. The data was first sparsified using SciPy in Python to make loading faster. In the implementation of Naive Bayes, the $P(X|Y)$ and $P(Y)$ distributions were estimated using Dirichlet-smoothed MAP and MLE, respectively. To extract the label counts for computing $P(X|Y)$, documents with the same label were separated and their word counts were summed. $P(Y)$ was also easy to estimate with NumPy's built in counting methods. The Naive Bayes algorithm was quick enough such that loaded data was transformed into dense form to enable easier programming.

Since logistic regression is an iterative algorithm and requires matrix multiplications, data was kept in sparse format to speed up these operations. Even with the speedups achieved by using sparse operations, a thorough hyperparameter search necessitated using multiple machines. Data was normalized using the standardization procedure such that each X_i becomes $X_{i,j} = \frac{X_{i,j} - \mu_i}{\sigma_i}$, where μ_i is the mean value for feature i and σ_i is the standard deviation of feature i's values. This transforms word counts to have mean 0 and standard deviation 1. This enables logistic regression to treat all words equally instead of emphasizing extremely rare and common words. The matrix form of the weight update given in the assignment was programmed using SciPy. Both minibatch and stochastic gradient descent were implemented. All results for logistic regression were generated using multiple iterations over the entire dataset as a minibatch.

2 Questions

Question 1:

The number of parameters required by such a model would be enormous. For example, if each of our documents were 1000 words long and we had 20 labels we would need $20^{1001} - 1$ parameters for the model.

Question 2:

As can be seen by Figure 1, there is a middle-ground for attaining high validation/testing accuracy when using different values of β. The value of β may be interpreted as a smoothing parameter which weighs the likelihood probabilities toward a uniform distribution. Thus, it is not surprising that increasing the value of β degrades accuracy on the training set. However, given that the training, validation, and test sets will have non-identical distributions for $P(X|Y)$, increasing this value by a certain amount can help prevent the model from overfitting to the training set and from assigning extremely low probabilities to examples that are infrequent in training data. However, decreasing the value of β too much can degrade performance on validation/testing data as the model becomes too constrained. From our experiments we found that values of $\beta \in [0.07, 0.09]$ generally performed best, with our highest test accuracy for Naive Bayes (88.131%) being achieved with $\beta = 0.070716$.

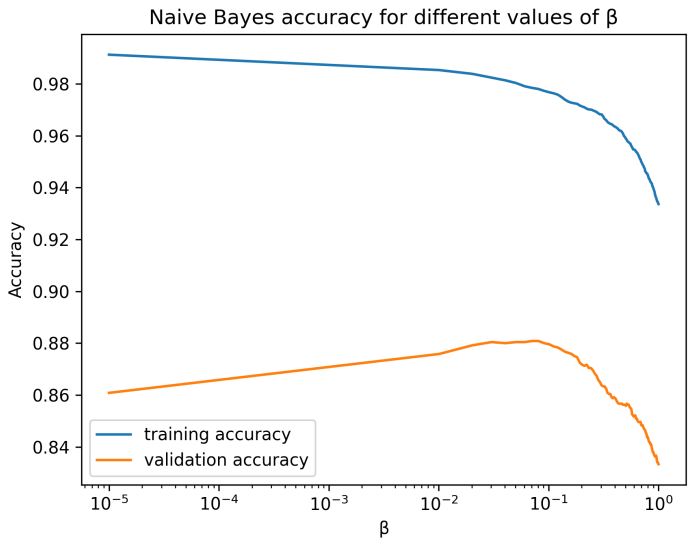

Figure 1: Training and validation accuracies for values of β between 0.00001 and 1. Results were generated using 100 evenly-spaced values of β taken from the range [0.00001, 1]

Question 3:

Our hyperparameter tuning results for logistic regression can be seen in Figure 2. In the first plot, η has significant impact on validation accuracy. Until around 2×10^{-3}, validation accuracy is maximized, suggesting that lower learning rates yield better performance. This makes sense as the gradient descent algorithm will overshoot the optimum for large values of η, but will converge closer at a slower rate when η is small. The regularization strength, λ, had no impact on validation accuracy as seen in the second plot. This likely implies that the algorithm has reached a ceiling on the complexity of the problem such that there is not room for improvement. Theoretically, there should be a nonzero amount of regularization that improves validation performance on non-trivial datasets. The third plot shows the impact of the number of training steps on validation accuracy. Performance peaks at 67 iterations and stabilizes for further training. As discussed, with a small enough η, accuracy should improve with more than 67 iterations; however, this is not seen in our smaller hyperparameter searches with η being a factor of 10 to 100 times smaller. This provides further evidence for a complexity ceiling on this algorithm and problem pairing.

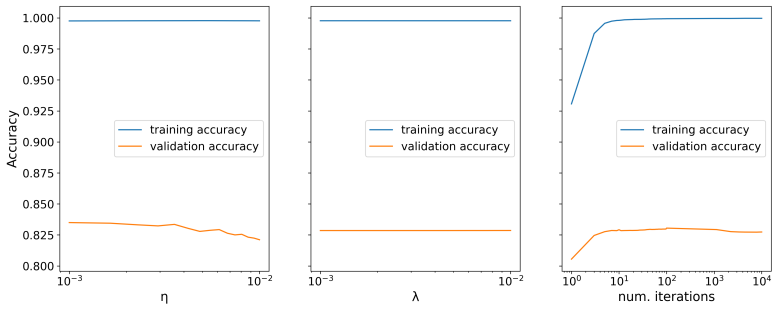

Figure 2: Training and validation accuracies for values of η between 0.001 and 0.01 (left), λ between 0.001 and 0.01 (middle), and the number of iterations between 1 and 10,000 (right). Results were generated using 15 evenly-spaced values of η and λ, and 10 evenly-spaced iteration values, with all odd counts between 1 and 99

Question 4:

In this section, we feature the results from our Naive Bayes and logistic regression models that scored highest on the Kaggle competition. The shown confusion matrices come from the validation split, which was taken from 20% of the labelled data. Naive Bayes achieved 87.625% validation accuracy and 88.544% test accuracy through Kaggle with a β of 0.011111. Logistic regression achieved 83.83% validation accuracy and 84.853% test accuracy through Kaggle with 67 iterations, an η of 0.001, and a λ of 0.001.

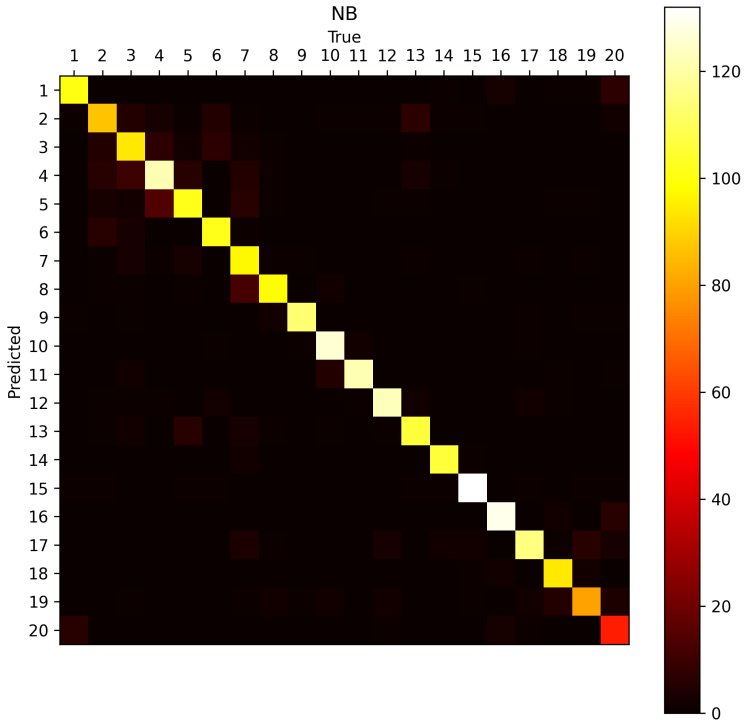

Figure 3: Confusion matrix for Naive Bayes classifier that attained the best Kaggle accuracy. The shown matrix comes from the validation split of our data.

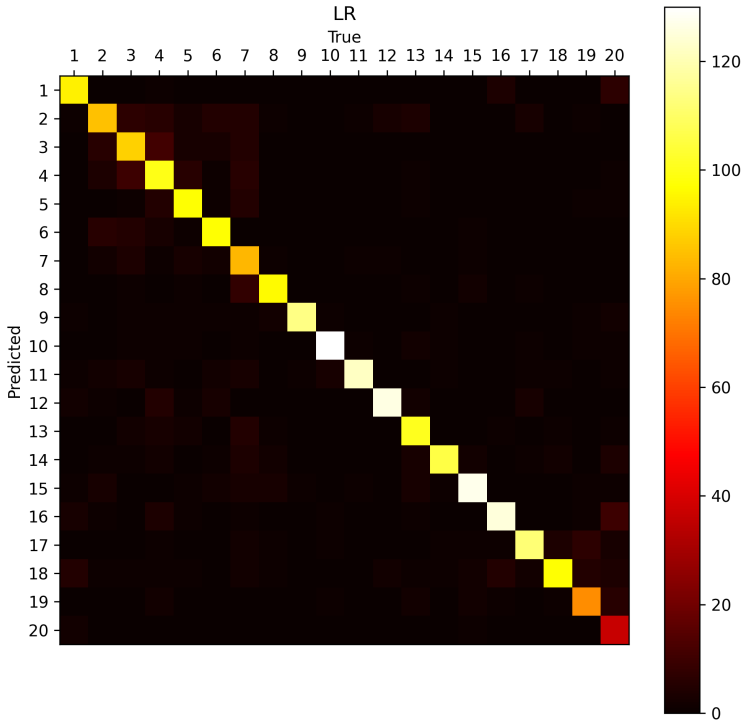

Figure 4: Confusion matrix for logistic regression classifier that attained the best Kaggle accuracy. The shown matrix comes from the validation split of our data.

Question 5:

As seen in Figure 3 and 4, both algorithms confuse groups 2, 3, 7, and 20 the most. These are computer graphics, MS Windows, classified ads, and miscellaneous religion topics. It is seen that the first three topics get mislabeled as each other as well as two recreational vehicle categories for classifieds. Computer terminology is commonly shared between systems, so the computer related documents can be similar. Secondly, vehicles are commonly sold as ads in news media, so we predict that causes these misclassifications. Lastly, the miscellaneous religion topics are confused with atheist and Christian documents. We argue that much of philosophy and religion use similar terms for imagery, so this could cause misclassification.

The confusion matrices show significantly more confusion in these sane categories when using logistic regression over Naive Bayes, particularly the classified ads group with computer and graphics documents. It is likely focusing on the words outside of graphics-related terms.

Question 6:

One method of ranking the importance of words would be to use the calculations of $P(X|Y)$ and $P(Y)$ in conjunction with the Kullback-Leibler (KL) Divergence [1]. KL Divergence, often denoted as $D_{KL}(P \parallel Q)$, is a measure of dissimilarity between two probability distributions P and Q. It is defined as follows for discrete distributions:

$$D_{KL}(P \parallel Q) = \sum_{x \in X} P(x) \cdot log\left(\frac{P(x)}{Q(x)}\right) \tag{1}$$

Note from equation 1 that the KL divergence is 0 if P and Q are identical and becomes increasingly large as the two distributions become more dissimilar. This property is useful for our task of ranking words because we want a measure that can pick out words where $P(X_k|Y_i)$ is relatively larger for one or two values of i compared to all other values of i.

Since there are 20 labels in our dataset, we can use the following formulation to measure the relative importance of a word k for predicting class i:

$$g(k,i) = \sum_{j=1}^{20} D_{KL}(P(X_k|Y_i) \parallel P(X_k|Y_j)) \tag{2}$$

Note that we don't have to explicitly mention $i \neq j$ in equation 2 since D_{KL} will be 0 for this case. Since we want to get an overall ranking of words we should also weight these values by $P(Y)$ as follows:

$$f(k,i) = P(Y_i) \cdot g(k,i) \tag{3}$$

Our formulation of finding the top n words is finding the values of k that give the n largest values of $f(k,i)$. These k values will indicate the top words our classifier is using.

This formulation has several nice properties. For one, it is unlikely general English words will be selected as most important. Although $P(X|Y)$ may be large for general words such as "the", the KL Divergence for these general words will be low because $P(X|Y_i)$ for these words is nearly the same for all values of i. As well, the method will penalize words that are very uncommon due to multiplication by the value of $P(X_k|Y_i)$ when calculating $D_{KL}(P(X_k|Y_i) \parallel P(X_j|Y_j))$.

Question 7:

Implementing the method outlined above and setting $\beta = 1/|V|$, the top 100 words used by the classifier (from the training data) were found to be the following:

'windows', 'bike', 'scsi', 'hockey', 'dod', 'graphics', 'encryption', 'dos', 'wolverine', 'ide', 'car', 'armenian', 'nhl', 'jpeg', 'team', 'controller', 'israel', 'sale', 'shipping', 'turkish', 'clipper', 'armenians', 'mb', 'god', 'bios', 'window', 'image', 'mac', 'space', 'motif', 'gun', 'season', 'drive', 'baseball', 'cars', 'jesus', 'launch', 'ei', 'card', 'israeli', 'church', 'fbi', 'christ', 'stephanopoulos', 'det', 'game', 'jews', 'bus', 'disk', 'xterm', 'sabretooth', 'hulk', 'christians', 'widget', 'bos', 'liefeld', 'he', 'dx', 'ride', 'orbit', 'players', 'key', 'shuttle', 'nsa', 'images', 'bikes', 'pitcher', 'riding', 'bmw', 'firearms', 'teams', 'motorcycle', 'comics', 'armenia', 'server', 'leafs', 'flyers', 'privacy', 'engine', 'espn', 'pit', 'msg', 'batf', 'monitor', 'sin', 'tor', 'escrow', 'mouse', 'apple', 'format', 'hitter', 'irq', 'of', 'hobgoblin', 'os', 'bible', 'patients', 'cancer', 'the', 'guns'

Question 8:

One indicator for if there was heavy bias in the training dataset is to look at the distributions of $P(X)$ (the probability of observing the word X at least once in a given document) for the top 100 words listed above. If distributions of $P(X)$ differ significantly between training, validation, and testing data then it is extremely likely that the training data was biased.

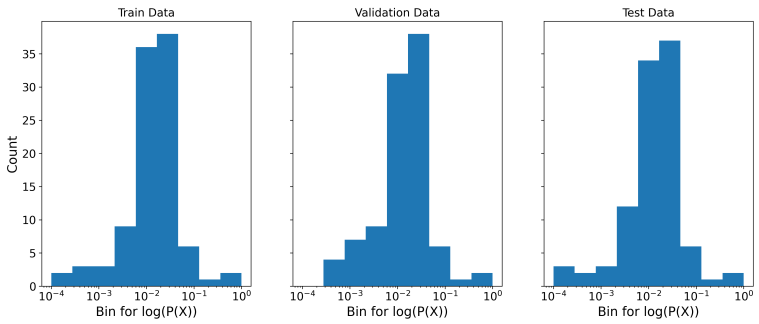

Figure 5: Histogram of top-100 word frequencies for training (left), validation (middle), and test (right) data. Note that frequencies were binned logarithmically to better display each distribution.

As can be seen from Figure 5, the frequencies of the top 100 words did not differ significantly between the training, validation, and testing datasets. In other words, the words our classifier most heavily relies on appear roughly as often in the validation and test data as they did in the training data. Thus the distributions of $P(X)$ do not (on their own) indicate any serious bias.

However, this analysis is limited in that it does not take into account how the values of $P(Y|X)$ may change from our training data to other data distributions. Although generally words in the top 100 pretty clearly track to one or two labels (e.g., 'hockey' goes with the label 'rec.sport.hockey'), there are some words that are oddly placed. For example, from a human-perspective it is not clear how 'wolverine' helps to distinguish our different newsgroups.

Additionally, the trained classifier would have limited application for real-world use because it is only looking at 20 newsgroups. Although words such as 'graphics' may be good for distinguishing one label with our given data (comp.graphics), if we were to add newsgroups discussing videogames and graphic design then that term would become less useful.

References

[1] Kullback, S., & Leibler, R.A. (1951). On Information and Sufficiency. Annals of Mathematical Statistics, 22, 79-86.