Machine Learning

Project 1 Report

By Vighanesh Sharma (vxs240002)

1. Results

The results/scores are compiled in the following tables for each dataset.

1. Dataset 1 (enron 1)

Model	Accuracy	Precision	Recall	F-1 Score
Multinomial Naïve Bayes (BOW)	90.57	87.85	82.55	85.12
Discrete Naive Bayes (Bernoulli)	76.97	84.37	36.24	50.7
Logistic Regression (BOW)	92	85.43	90.20	87.75
Logistic Regression (Bernoulli)	94.44	89.86	93.00	91.40
SGD Classifier With GridSearchCV (BOW)	89.33	81.45	86.01	83.67
SGD Classifier With GridSearchCV (Bernoulli)	95.5	90.72	95.80	93.19

2. Dataset 2 (enron 2)

Model	**Accuracy**	**Precision**	**Recall**	**F-1 Score**
Multinomial Naïve Bayes (BOW)	92.05	82.85	89.23	85.92
Discrete Naive Bayes (Bernoulli)	80.12	79.66	36.15	49.73
Logistic Regression (BOW)	92.00	84.21	83.47	83.84
Logistic Regression (Bernoulli)	95.68	90.598	92.17	91.37
SGD Classifier With GridSearchCV (BOW)	90.71	78.12	86.95	82.30
SGD Classifier With GridSearchCV (Bernoulli)	95.46	89.83	92.17	90.98

3. Dataset 3 (enron 4)

Model	**Accuracy**	**Precision**	**Recall**	**F-1 Score**
Multinomial Naïve Bayes (BOW)	95.02	94.60	98.72	96.62
Discrete Naive Bayes (Bernoulli)	89.5	97.98	87.21	92.28

Logistic Regression (BOW)	95.5	96.86	96.86	96.86
Logistic Regression (Bernoulli)	97.94	97.20	100	98.58
SGD Classifier With GridSearchCV (BOW)	95.70	96.87	97.12	97.00
SGD Classifier With GridSearchCV (Bernoulli)	95.88	96.64	97.65	97.14

2. <u>Tuning the Hyperparameters</u>

1. **<u>Logistic Regression</u>**
   1. **Learning Rate**
      a. High learning rate can cause the model to overshoot or oscillate the convergence point.
      b. Low learning rate can make training very slow and will require higher number of iterations to converge.
      c. So, I made sure to choose the learning rate as such.
      d. I tested my assumption by dividing train data randomly into 70/30 ratio and trained on 70% of data and validated on 30% data, by using a range of learning rates. I found out that 0.1 was optimal for all the sets.
   2. **Lambda Value**
      a. Lambda value is used to prevent overfitting, it does so by reducing the weights. Big weights can make model more complex and thus lead to overfitting.
      b. But the higher value of lambda can lead to underfitting, since then our model won't be complex enough to classify properly.
      c. I set the range of lambda values from 0.0001 to 10 in the increments of 10.
      d. Just like it says in the project description, I divided the training data into 70/30 ratio. I iterated over the values of lambda in the range and trained on 70% data. Then I validated on 30% of the data. I found out that lambda value of 0.1 was optimal for all the datasets.
   3. **Hard limit on the iterations**

   a. Having large number of iterations require a lot more time and resources, which might not be worth it, since the improvement in the model could be very less.
   b. Stopping before convergence can also help preventing data from overfitting.
   c. I decided to go with 250 iterations, since I could train data on one dataset in around 20 minutes and still gain a good accuracy, while giving me leverage to experiment more, I set that the hard limit.

2. **<u>SGDClassifier with GridSearchCV</u>**
   1. **GridSearchCV** takes care of the Hyperparameter tuning by finding the set of parameters which gives the best score based on scoring chosen.
   2. **Penalty**
      a. I chose l1 and l2 regularisations, since those were the ones I was most familiar with in the class.
   3. **Alpha**
      a. It was the regularisation constant (lambda), I chose the ones I got best scores in the experiments conducted before
   4. **Learning Rate**
      a. I chose 'constant' to match with the one I chose in logistic regression and 'optimal' and 'adaptive' because I wanted to learn if these LRs improve the performance.
   5. **Max Iterations**
      a. I chose 100 and 250, since these were the values I experimented in logistic regression, and I wanted to keep the maximum same so that comparison can be consistent.
   6. **Loss**
      a. I chose 'log_loss' this is for the logistic regression with SGD, so that I can compare better. 'squared_hinge' is for Linear SVM with SGD, to experiment with a different model and 'perceptron' to take simple model.

3. 3rd Party Tools used

a. **NLP toolkit:** It was used to tokenize the words in the email. Source: https://www.nltk.org/
b. **NumPy:** It was used for efficient array computations, like initialising array of 0s, random array for weights and efficient dot product. Source: https://numpy.org/
c. **Scikit-learn:** It was used to train SGDClassifier using GridSearchCV. Source: https://scikit-learn.org/stable/
d. **Scipy:** It was used for its Sigmoid function. Source: https://scipy.org/

4. **Project Questions**

1. **Which data representation and algorithm combination yields the best performance (measured in terms of the accuracy, precision, recall and F1 score) and why?**

Answer:

- **Overall, Logistic Regression** with **Bernoulli Data Representation,** gives the best scores.
- It was expected to perform better than Naïve Bayes, since unlike Naïve Bayes, Logistic Regression doesn't make any assumptions about the independence of the features for a give class. In a text, there is always dependence between the words which are features.
- Logistic Regression with Bernoulli representation worked better than with BOW representation because: -
    - Bernoulli model has much simpler feature space than BOW model, since it takes values between only 0 or 1. This reduces noise and the unnecessary complexity, since sometimes number of times a word appear won't matter, for example the words like 'is', 'a', 'and' , 'the' etc. may appear a lot of times in an email. Since, their scoring is higher, the model will give more attention to them and lesser attention to the words like 'free', 'million' which may appear only a single time but determine the whole result.
    - Similarly, Bernoulli handles rare words better which are less frequent.
- Rest of the reasons will be explained in detail in the next answers.

2. **Does Multinomial Naive Bayes perform better (again performance is measured in terms of the accuracy, precision, recall and F1 score) than LR and SGDClassifier on the Bag of words representation? Explain your yes/no answer.**

Answer:

- Multinomial Naïve Bayes did not perform better than Logistic Regression on the Bag of words representation. However, it performed better than SGD classifier on 1st and 2nd Datasets.
- Logistic Regression on BOW model performs better than Naïve Bayes on Logistic Regression because: -
  - Naïve Bayes assumes features are independent given the class. However, this isn't a correct assumption in this case. As, some word combinations can be more indicative of a spam or a ham message. For example, combinations like 'king of', 'won lottery' are strong indicator of spam messages. Whereas Logistic Regression doesn't assume independence and take into account the interactions of word (when training weights).
  - Naïve Bayes also give a lot more attention to frequency of words than Logistic regression with regularization. Highly frequent words like 'and', 'the' etc add noise.
  - Naïve Bayes doesn't have l2 regularisation like Logistic Regression, which may cause overfitting of the data.
- Naïve Bayes showed slightly better scores than SGD Classifier with GridSearchCV in Dataset 1 and 2 because, SGD can underperform on smaller datasets, as SGD use mini batches to train and generalise results based on those batches. Mini batches on small dataset can produce sub-optimal generalisation. This could be the reason it gives better results in Dataset 3 which is a bigger dataset. (around 25% bigger than dataset 1 and 2).

3. **Does Discrete Naive Bayes perform better (again performance is measured in terms of the accuracy, precision, recall and F1 score) than LR and SGDClassifier on the Bernoulli representation? Explain your yes/no answer.**

Answer:

- No, it does not perform better thank LR and SGDClassifier on the Bernoulli Representation. Moreover, it gives very bad scores for Dataset 1 and 2.
- The reason could be that in dataset 1 and 2, the number of spam emails are less than half when compared to ham emails. As a result,

the prior is biased more towards ham and classify spams as hams. Similar could be the case with dataset 3 also why its performance is bad, but not as bad as dataset 1 and 2, because spam emails usually have some suspicious words/phrasing besides having normal words, now in dataset 1 and 2, these words could not be learned but in dataset 3 it was learned, normal words will appear in both spam and ham emails. So, doesn't make it as biased as in dataset 1 and 2.

- The reason why Logistic Reasoning and SGDClassifier performs better than Naïve Bayes are given in the previous answer, except that added noise from frequency of common words is not that much in Bernoulli Model, but this frequency while inducing noise can help Naïve Bayes perform better, it is like a necessary evil. Since, for example, rare words which appear only in spam emails can contribute more towards the prediction of the model and decreasing the prior.

4. **Does your LR implementation outperform the SGDClassifier (again performance is measured in terms of the accuracy, precision, recall and F1 score) or is the difference in performance minor? Explain your yes/no answer.**

Answer:

- Yes, my LR implementation outperforms, though difference in scores is only 1 to 3 %, this increase in scores can be significant when scores are already very high (90%+).
- My implementation of Logistic Regression uses Gradient Ascent which updates parameters using the whole dataset in each iteration, but it was slow.
- SGDClassifier on the other hand uses mini batches to train the dataset, it was very fast, around 20 times faster than my implementation.
- But because of this reason SGDClassifier can underperform on smaller datasets, since mini batches can result in poor approximations for the overall model. The results can also be unstable.

- Usage of mini batches to train the model also make it more sensitive to hyperparameters, it could be that the hyperparameters options given by me to the GridSearchCV were all not optimal.
- Using mini batches would be more beneficial for higher number of iterations, it could be that SGDClassifier didn’t even come close to convergence in the hard limit on the number of iterations I used, whereas Gradient Ascent uses calculations on whole of the data to converge, so it converges faster.