Machine Learning
University of Mannheim
Winter 2019
Prof. Dr. Rainer Gemulla

Assignment 2 : Logistic Regression

Sudhanva Rao

Spambase Dataset

In this assignment, we are using a dataset on email spam detection, provided in the file data/spamData.mat
The dataset is derived from 4601 emails, each being labeled as no-spam (0) or spam (1). Each example has 57 features:

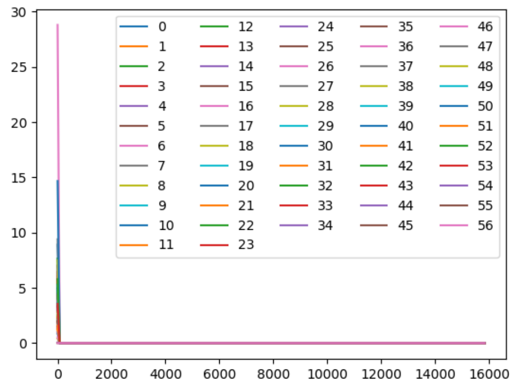
- 48 word features, each indicating the frequency percentage of a word in the email (e.g., “business”, “free”, “george”).
- 6 character features, each indicating the frequency percentage of a character in the email (for [!$#)
- 3 features on length statistics of consecutive upper case letters (e.g., “HELLO” gives 5), one for minimum length, one for maximum length, and one for average length.

The dataset is split into a training set (3065 examples) and a test set (1536 examples). We provide code to load the data and provide all feature names.

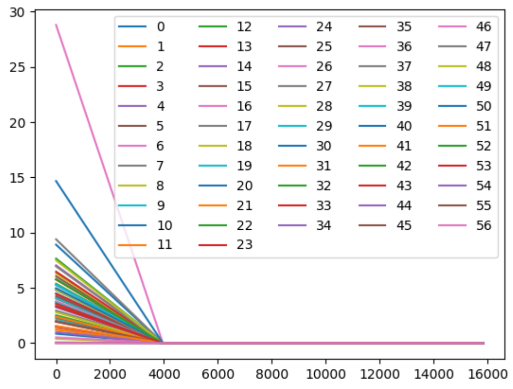
1. **Dataset Statistics**

   Explore and preprocess the dataset.

   (a) Look at the kernel density plot of all features and discuss what you see (or don't see).
   The kernel density plot is as shown below :



(a) KDE



(b) KDE to illustrate the jump

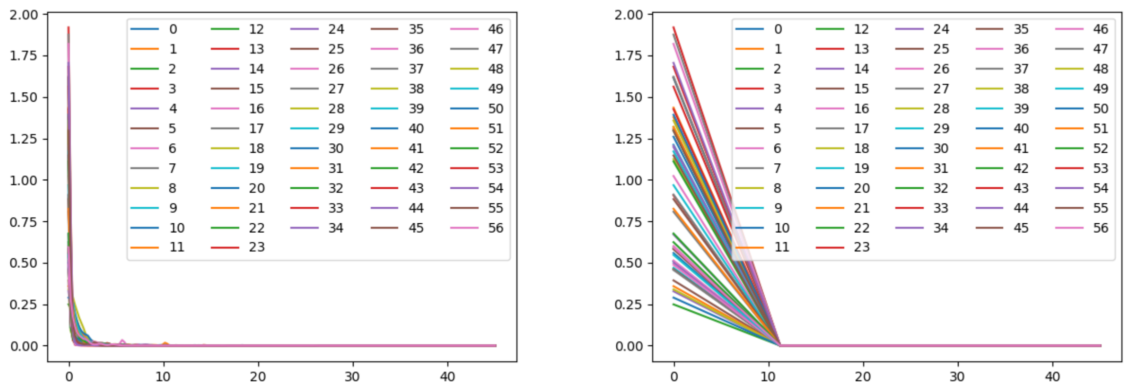
Figure 1: Kernel density plot of features.

Figure 1 shows the kernel density plot for all features. One can see that the value range for the different features is not uniform. Some of the features have values that are as high as 16000, while most of the features have values that are between 0 and 50. Due to this imbalance, the distributions of the features can't be properly seen. This can be clearly seen in second image. Second image is created by having very few points for density calculation.

(b) Normalize the data using z-scores, i.e., normalize each feature to mean 0 and variance 1. Normalize both training and test data. In particular, think about how test data should be normalized.Make sure to stick to the variable names provided in the code fragments. From now one, we will exclusively work with the normalized data.

The training data is normalized by subtracting the mean and dividing by the standard deviation. Our test data has to be normalized with the mean and standard deviation of the training data, so that both data sets are rescaled with the same parameters. If one uses different parameters for each set, the models derived from the training set can not be correctly evaluated on the test set

(c) Redo the kernel density plot on the normalized data. What changed? Is there anything that "sticks out"?



(a) KDE (b) KDE to illustrate the jump

Figure 2: Kernel density plot of features after normalization.

Figure 2 shows the kernel density plot for all normalized features. Since the features have now similar value ranges, it is easier to study their distributions. One can see that the distributions of the features are in general right-skewed. Two features stick out have a bump in their density at around 5 and 10, respectively. These bumps are out of the ordinary and therefore these features might have some important information for detecting spam emails.

2. Maximum Likelihood Estimation

(a) Show analytically that if we use a bias term, rescaling (multiply by constant) and shifting (add a constant) features leads to ML estimates with the same likelihood. Why do you think we computed z-scores then?

Let us take weight vector w, input vector X and log odds η(w,x). Let us take two constant vector a and b.
We have

$$\eta(w, X) = w^T X = w_0 + \sum_j w_j X_j \tag{1}$$

By scaling and offsetting the features by vector a and b respectively, we get

$$\begin{aligned}\eta(\tilde{w}, \tilde{X}) &= \tilde{w}^T(aX - b) \\ &= \tilde{w_0} - \sum_j \tilde{w_j} b_j + \sum_j \tilde{w_j} a_j X_j\end{aligned} \tag{2}$$

In MLE, weight vectors are determined such that likelihood or log odds are maximum. As a and b are constant vectors, it implies that the weight vector before and after scaling the features should be related via vectors a and b. From equation 1 and 2 we can imply that

$$\tilde{w_o} = w_0 + \sum_j \frac{b_j}{a_j} * w_j \tag{3}$$

and

$$\tilde{w_j} = \frac{w_j}{a_j} \tag{4}$$

which implies, equation 2 becomes

$$\begin{aligned}\eta(\tilde{w}, \tilde{X}) &= w_0 + \sum_j \frac{b_j}{a_j} * w_j - \sum \frac{b_j}{a_j} * w_j + \sum_j \frac{a_j}{a_j} w_j X_j \\ &= w_0 + \sum_j w_j X_j \\ &= \eta(w, X)\end{aligned} \tag{5}$$

Since the log odds are the same for the rescaled and non-rescaled case, not only the Likelihood is the same, but also the predictions are the same. However, rescaling the data by using z-scores is still useful, as it allows us to easily compare the importance of the resulting weights. Additionally in case of gradient descent we will converge to the weight vector faster.

(b) Complete the methods for computing the likelihood,log-likelihood, and gradient of the log-likelihood for logistic regression. We do not use a bias term throughout.
We have

$$\eta = w^T X$$

We have predicted class probabilities as

$$p(Y_i|x_i, \theta) = \log \frac{1}{1 + e^{-\eta}}$$

and log likelihood given by

$$l(y|X, \theta) = \sum_N y_i log p_i + (1 - y_i) log p_i$$

where y_i is the true label.

The gradient of log-likelihood is given by

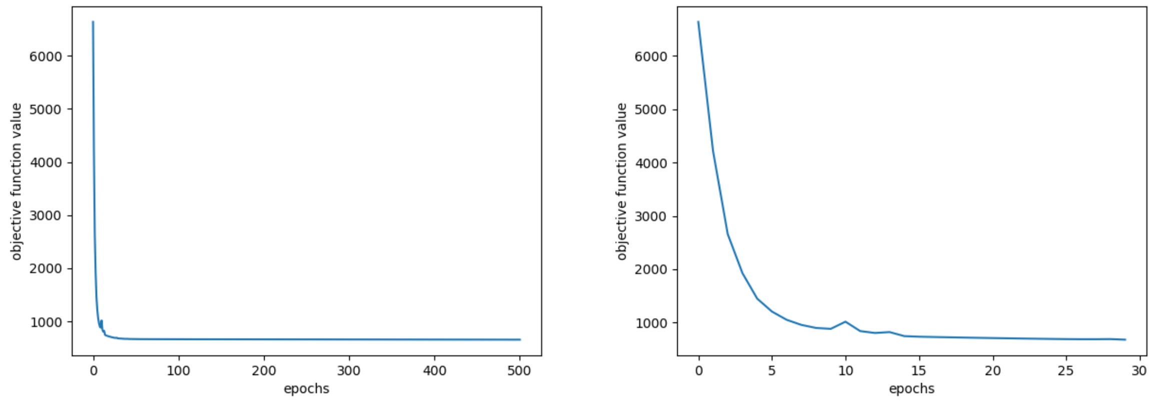
$$\frac{\partial l}{\partial w} = e^T * X$$

where e is error given by

$$e_i = y_i - p_i$$

(c) Implement gradient descent using the framework provided to you.

The objective function is given by negative log likelihood and the weights for each feature are updated at every epoch based on the product of error e_i and input feature, adapted according to learning rate ϵ.

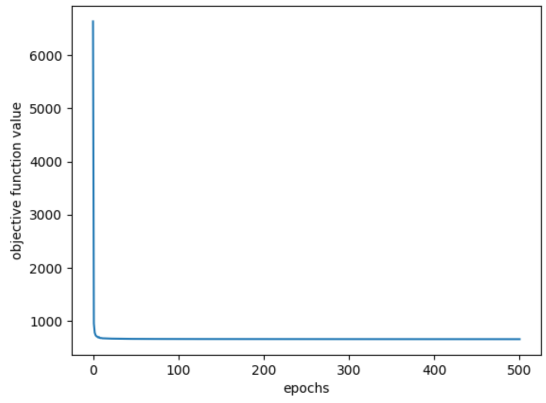


(a) GD (b) GD zoomed

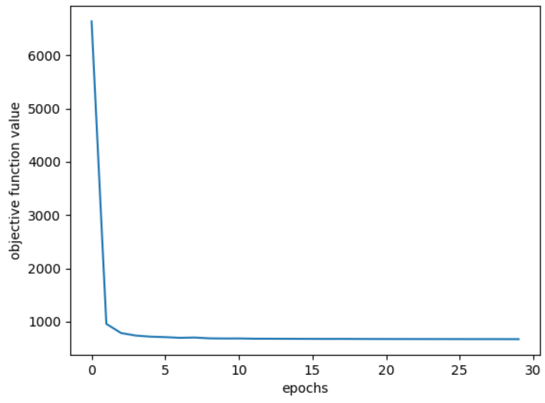
Figure 3: Gradient descent learning

(d) Implement stochastic gradient descent.

For Stochastic gradient descent, the random examples are chosen, (in our case without replacement) and the approximate gradient is calculated like GD. The parameters are then updated. This is repeated for number of times equal to number of samples, every epoch.



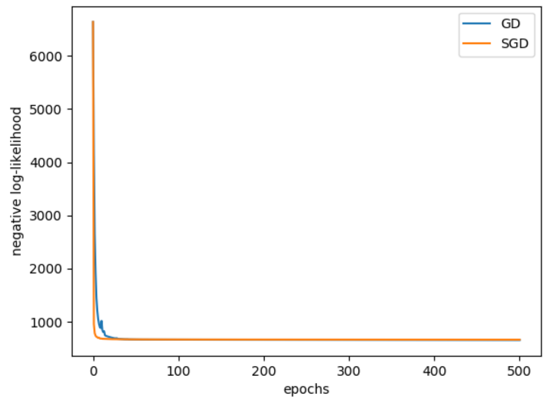
(a) SGD



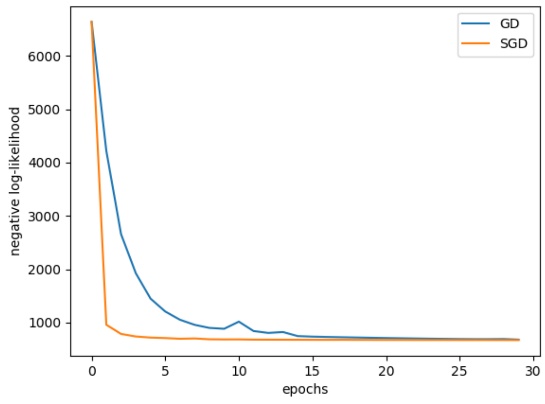
(b) SGD zoomed

Figure 4: Stochastic Gradient descent learning

(e) Explore the behavior of both methods for the parameters provided to you. Discuss!



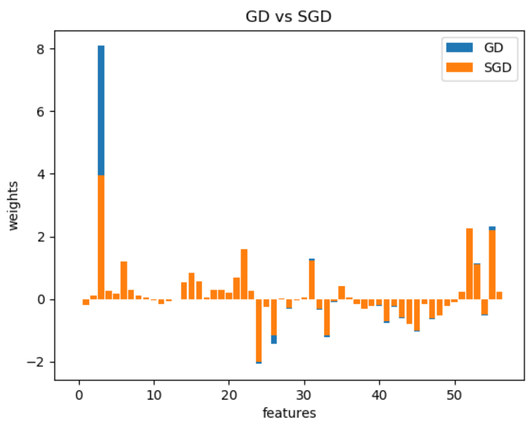
(a) learning 500 epochs



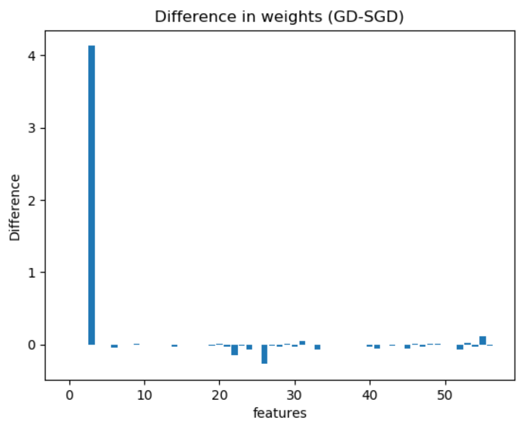
(b) learning zoomed

Figure 5: Stochastic Gradient descent vs Gradient Descent learning

Figure 5 shows the development of the negative log-likelihood during gradient descent (GD) and stochastic gradient descent (SGD). One can see that SGD converges faster than GD. However, after about 30 epochs GD has closed the gap and is even able to achieve smaller values than SGD for the rest of the epochs. After 500 epochs GD has found weights that lead to a negative log-likelihood of 655.41, while SGD found weights that only lead to a negative log-likelihood of 663.30. It is also interesting to see that SGD converges more smoothly during the rst 30 epochs compared to GD. This might be due to the fact that SGD updates the weights N times with approximate gradients during one epoch, while GD updates the weights only once with exact gradients in one epoch. The multiple updates of SGD might allow the algorithm to make small corrections during one epoch.



(a) weights



(b) weight difference

Figure 6: Stochastic Gradient descent vs Gradient Descent learning

Additionally, it is interesting to note that the weights of all the features are almost same except third feature. This maybe due to the distribution of the third feature(maybe mostly 0s)

3. **Prediction**

Complete the predict and classify methods for the predicted spam probability and predicted class label, respectively. Explore the models that you fit in the previous task and discuss. Study the composition of the weight vector: which features are important, which are not? Is this intuitive?

The confusion matrix of the better model(Gradient Descent) is as follows in Table 1:

	precision	**recall**	**f1-score**
0	0.93	0.94	0.93
1	0.91	0.88	0.89
micro avg	0.92	0.91	0.91
macro avg	0.92	0.92	0.92

Table 1: Gradient Descent: Classification report

For the predictions a probability threshold of 0.5 is used. Overall, the results look quite good. The model achieves high precision and recall for spam, as well as for non-spam emails. However, in our application the avoidance of false positives might be especially important. It seems reasonable to assume that classifying a non-spam email as spam, is more detrimental than classifying a spam email as non-spam. In the first case a person might miss important information, while in the second case a person gets all the important information and some additional spam. In light of these assumptions, a model with higher precision for spam emails might be preferable, even if that leads to a smaller recall.

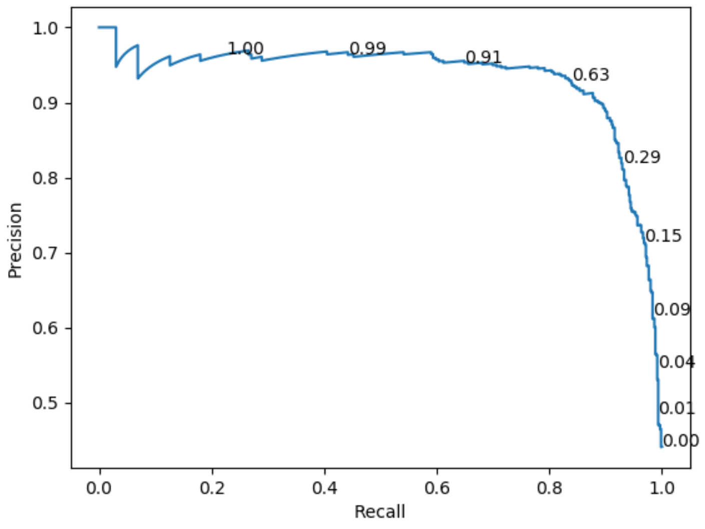


Figure 7: Precision-Recall pairs for spam emails and different probability thresholds

Figure 7 shows the precision-recall curve for the spam email class. One can see that with a probability threshold of about 0.5, a good tradeoff between precision and recall can be achieved. However, a higher probability threshold might be preferable, to achieve a higher precision. When using a threshold of about 0.9, the model is able to achieve a precision of about 0.95. The downside is that the recall drops to about 0.7. This is due to the shape of the precision-recall curve, as the curve gets quite flat for threshold values above 0.65.

important feature	**weight**	**unimportant feature**	**weight**
word_freq_3d	8.09	word_freq_650	-0.001
capital_run_length_longest	2.314	word_freq_make	-0.009
char_freq_$	2.19	word_freq_report	-0.01
word_freq_000	-2.08	word_freq_labs	-0.03
word_freq_857	1.46	word_freq_telnet	0.04

Table 2: Important/Unimportant features

Table 2 shows the features with the five highest and five lowest weights in absolute terms from the GD model. When looking at the important features (high absolute weights) one can see that a large number of dollar signs and long sequences of capital letters indicate spam emails. This seems very intuitve. Additonally, a high count of the word '3d' indicates spam emails. Since the data set is not very big, most of the spam emails might be about the same topic. In this case this could be 3d technology. When looking at the unimportant features (low absolute weights) one can see that these include the word counts of 'report' and 'labs'.It is likely that the words 'report' and 'labs' occur in almost all emails, independent of spam and non-spam.

4. **Maximum Aposteriori Estimation**

(a) Implement gradient descent for MAP estimation of logistic regression with a Gaussian prior / L2 regularization with hyperparameter λ. You can reuse the methods of your solution for MLE.

The log density of the posterior of logistic regression with L2 regularization is given by

$$l(y|X, w) - \frac{\lambda}{2}||w||^2$$

where l(y|X,w) is the log likelihood.

and the gradient is given by

$$e^T * X - \lambda * w$$

(b) Study the effect of the prior on the result by varying the value of λ. Consider at least the training data log-likelihood, the test data log-likelihood, and the prediction accuracy. Are these results surprising to you?

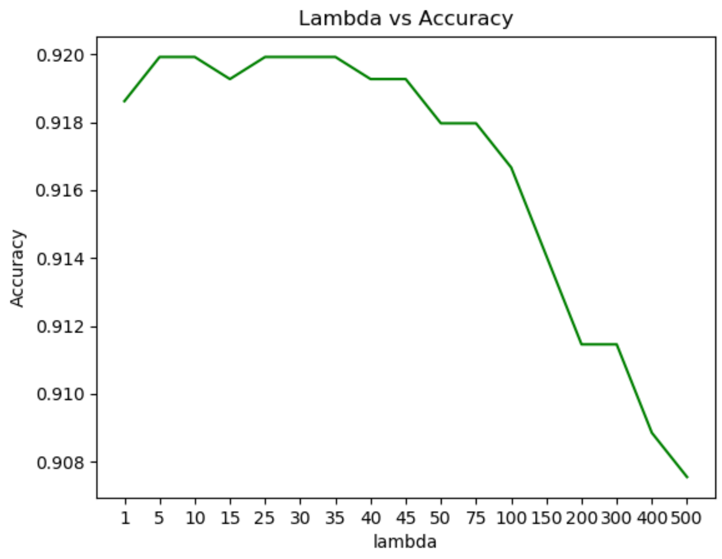


Figure 8: Accuracy for different lambdas

Figure 8 shows the accuracy of the test set for different values of lambda. One can see that the highest accuracy of about 0.92 is achieved for a lambda between 25 and 35. It seems that for a lambda below 5, the model overfits to the data, while for a lambda above 40, the model underfits to the data.

(c) Study the composition of the weight vector for varying choices of λ (try very large values). Try to explain what you saw in the task above.

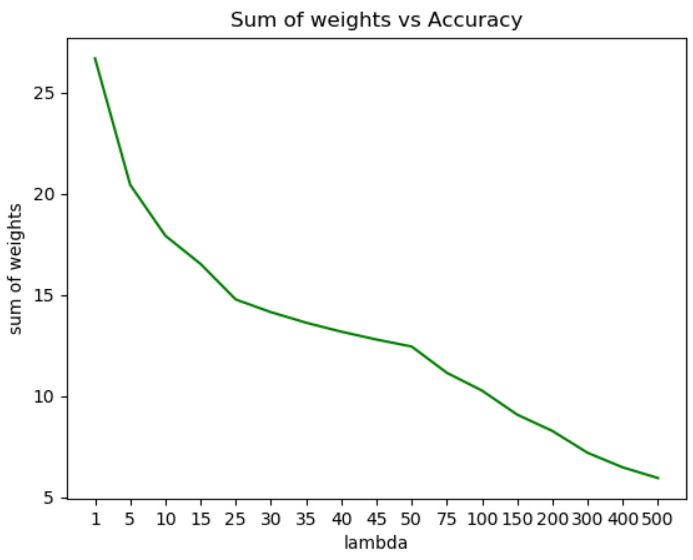


Figure 9: sum of weights for different lambdas

Figure 9 shows the sum of all weights for different values of lambda. As one can see, the sum of weights decreases as lambda increases.A higher lambda puts a higher penalty on large weights. This also explains the previous observation, that the log likelihood decreases as lambda increases. Since lambda only effects the size of the weights, an increase in lambda leads to smaller weights, which in turn leads to smaller log odds, which finally results in a smaller likelihood.