Seminar Report

Machine Learning - Assignment 02

Jonas Ortner (joortner, 2265527)
Marmee Pandya (mpandya, 1963521)

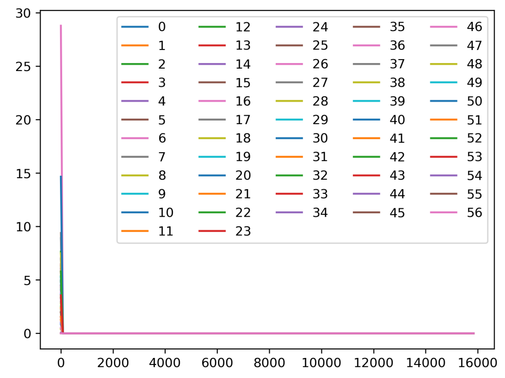
October 24, 2025

1 Dataset Statistics

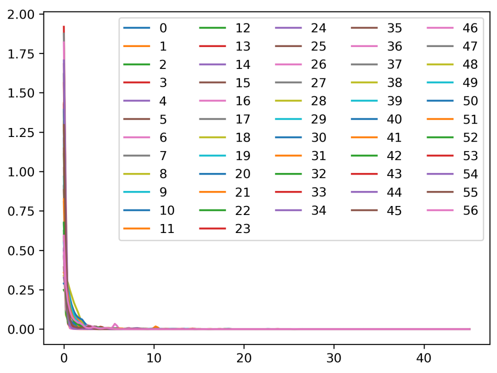
The figure 1a shows the kernel density plot for the original data, showing the estimated density of each feature per sample. There is no way to get any useful insight from this plot, as all the scale of the x-axis is way too large to identify trends in the feature densities. The reason for this is that the feature densities vary extremely: For word counts, the maximum value across all 3065 samples) is below 100, for length statistic features the maximum values get even larger than 1000 for extreme cases.
As a consequence, normalization to the data using z-scores has been applied,

$$z = \frac{x - \hat{\mu}}{\hat{\sigma}} \ ,$$

where x is an arbitrary sample, $\hat{\mu}$ is the estimated mean of the samples, $\hat{\sigma}$ is the estimated variance of the samples and z is the normalized feature. $\hat{\mu}$ and $\hat{\sigma}$ are estimated using only the training data, but then the normalizing routine with the estimated values has been applied to the testing data as well.
One can clearly see that normalizing helps to gain useful insight from the kernel density plot. Most features show a fastly dropping curve (which is what we would expect from normal mails) but some peaks are identifiable, most striking a peak from the feature providing the sum of lengths of upper case letters. Presumably, this peaks belongs to spam mails.



(a) Original data



(b) Normalized data

Figure 1: Comparison of kernel plots for original and normalized data.

2 Maximum Likelihood Estimation

2.1 Analysis of bias terms and scaling

In this part of the asssignment, we should show that scaling features by a constan $a > 0$ as well as shifting them by another constant $\vec{c}$ leads to estimates with the same Likelihood.

As for the iid. assumption the likelihood decomposes into the individual likelihoods, which are given by

$$L = \sigma(\vec{w} \cdot \vec{x} + b)^y \cdot (1 - \sigma(\vec{w} \cdot \vec{x} + b))^{1-y}.$$

Here, $\cdot$ denotes the scalar product, $\vec{w}$ the weight vector and b the bias feature. If one can show that the argument of the sigmoid function stays the same, it immediately follows that the likelihood stays the same as well. Therefore, we need to show that a weight vector $\vec{w}'$ and a bias vector b' exist which fulfill the following criteria:

$$\vec{w} \cdot \vec{x} + b = \vec{w'} \cdot (A\vec{x} + \vec{c}) + b' \ .$$

A denotes a diagnoal matrix with entries a_1, ..., a_n that is used to scale the n features of each vector x. Especially, we know that an inverse A^{-1} exists because A is diagonal and thus invertible. We set $\vec{w}' = \vec{w}A^{-1}$ and $b' = b - \vec{w}' \cdot \vec{c}$. Then we can simplify:

$$\begin{aligned} &\vec{w'} \cdot (A\vec{x} + \vec{c}) + b' \\ &= (\vec{w}A^{-1}) \cdot A\vec{x} + \vec{w}' \cdot \vec{c} + (b - \vec{w'} \cdot \vec{c}) \\ &= \vec{w} \cdot \vec{x} + b \ , \end{aligned}$$

which is what we wanted to show. This proofes that for modified samples $\vec{x}' = A\vec{x} + \vec{c}$ we can find a new weight vector and bias term such that the likelihood doesn't change.

2.2 Comparison

In this task gradient descent and stochastic gradient descent should be compared. The first differences is the runtime of the code. Whilst 500 gradient descent epochs finish within 0.1 seconds, stochastic gradient descent need 7 seconds. This is because one epoch in gradient descent can be fully vectorized, hence computation is much faster. In contrast, for our stochastic gradient descent implementation, the program needs to loop over all the samples, which takes much more time.
However, the rate of convergence per epoch is much better for SGD, as figure 2 shows. After 500 epochs, both trained models yield a training accuracy of 92.14% and a testing accuracy of 91.86%. However, one can clearly see that the model trained with SGD reached this plateau much faster than the model trained with GD, which is what we would have expected, as SGD takes 500 steps within an epoch compared to GD which takes only a single step in an epoch.

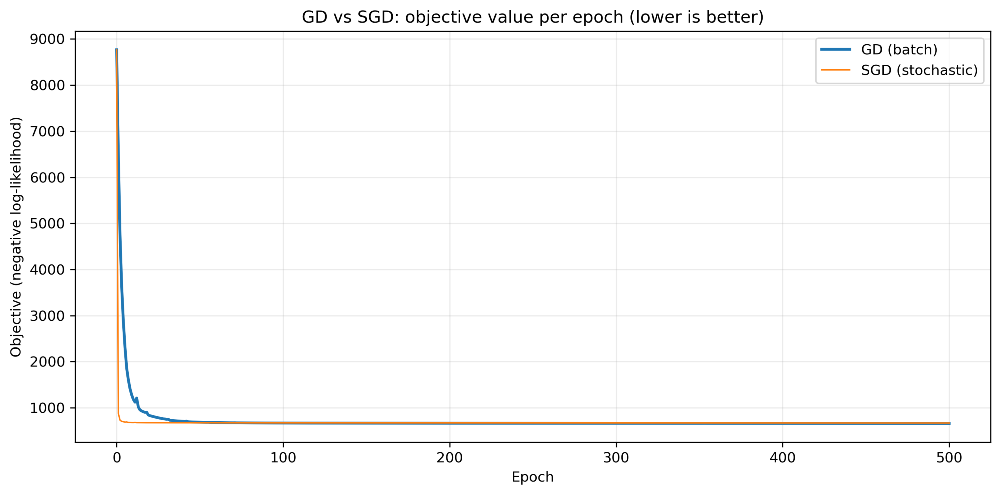


Figure 2: Convergence rate of GD and SGD classifier

Lastly, figure 3 shows the learning rate of both trained logistic regression models. One can see the zig zag trend belonging to the bold-driver heuristic. The step size increases until the model performance drops, then the step size drops as well and again starts to increase. Remarkably, the step size of the SGD model is constantly below the step size of the GD model. Probably, due to noisy gradients the SGD model performance decreases much more often than the performance of the GD model which uses a more accurate computation of the gradient. This might explain the much lower step size.

3 Prediction

In task three, I have completed a predict and classify method to get predictions for the labels after obtaining a good approximation for the weight vector $\vec{w}^*$ using gradient descent. The macro average of the F_1-score is 0.91 which is quite decent. Next, I have

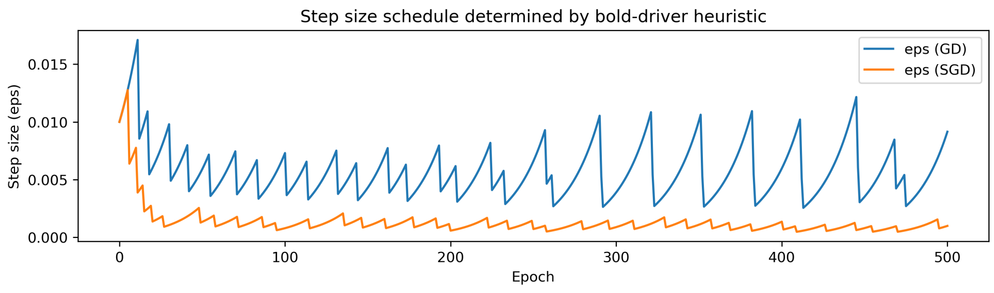


Figure 3: Learning rate of GD and SGD classifier

sorted the weight vector by magnitude of the 57 dimensions to learn which features have been the most important ones for prediction.
For predicting negative samples (non-spam), the most important features where word counts for hp, george, edu, re, meeting. This makes sense as all of this words seem to primarily occur in non-spam e-mail. Obviously, there are many more words important for the model predictions, the given ones are just a subsample of the 5 most important ones.
For the prediction of a positive sample (spam), the most important features have been the word counts for 3d, $, 000, remove and the longest captial run length. This seems to make sense as well, many spam mails contain description about money which explains the 000 and %. The also often create pressure which might give record of remove. Its nothing new that spam mails often contain long capital words. Maybe the most interesting finding is, that the word 3d seemingly also belongs to spam mails.
However, we need to keep in mind that a normalization has been applied to the features, therefore a direct interpretation of the weight vector and comparison of the individual magnitudes is difficult. Nevertheless, the findings show that the model is capable to learn useful aspects of spam mails.

4 Maximum Aposteriori Estimation

4.1 Effect of Prior Strength

We systematically varied $\lambda \in \{0, 5, 10, 50, 100, 200, 500, 1000\}$ and evaluated training/test log-likelihood and accuracy. Table 1 summarizes the results.

Observations: The training log-likelihood decreases monotonically with increasing λ, as expected since the penalty term constrains model fit. Test log-likelihood also decreases continuously, suggesting the dataset has strong signal-to-noise ratio where even minimal regularization reduces capacity to capture genuine patterns.

Test accuracy shows the most interesting pattern: it peaks at $\lambda = 5$ or $\lambda = 10$ (91.99%), slightly outperforming the unregularized MLE (91.73%). This represents the optimal bias-variance tradeoff. For $\lambda \geq 200$, accuracy drops noticeably as the model underfits.

Table 1: Performance metrics across different regularization strengths

λ	Train LL	Test LL	Train Acc	Test Acc
0	-661.52	-430.37	0.9214	0.9173
5	-684.16	-435.01	0.9204	0.9199
10	-698.42	-436.32	0.9214	0.9199
50	-772.85	-455.45	0.9194	0.9180
100	-833.01	-476.74	0.9155	0.9167
200	-917.14	-509.56	0.9155	0.9115
500	-1072.21	-573.82	0.9126	0.9076
1000	-1224.72	-639.64	0.9106	0.9004

Surprising aspects: Unlike typical regularization scenarios where test likelihood improves before degrading, our test likelihood decreases even at small λ values. This indicates the spam dataset features are highly informative with minimal noise, and the MLE solution already generalizes well. The modest accuracy improvement (0.26 percentage points) confirms that overfitting is naturally limited by dataset quality and size.

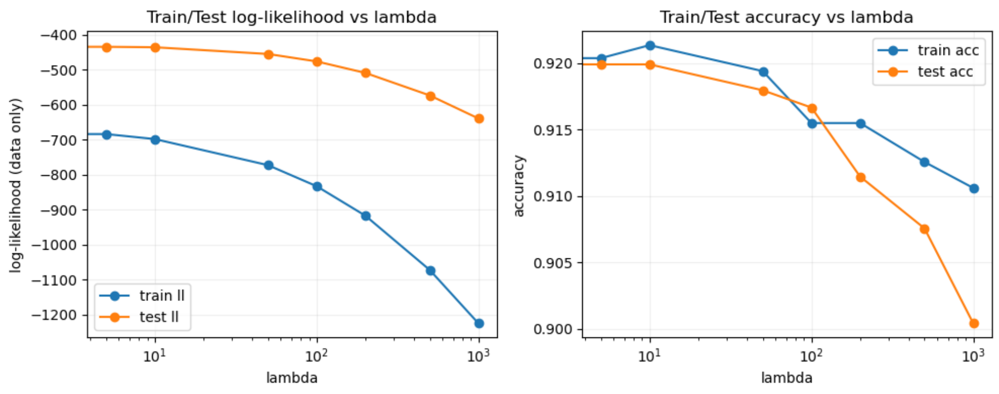


Figure 4: Training and test performance metrics across regularization strengths.

4.2 Weight Vector Composition

We analyzed how individual feature weights change across λ values by examining the top features (union of top-12 across all λ).

Weight shrinkage patterns: All weights shrink systematically with increasing λ. Features with large initial magnitudes experience dramatic reductions:

- `word_freq_3d`: 4.83 $\rightarrow$ 0.04 (99% reduction)
- `capital_run_length_longest`: 2.40 $\rightarrow$ 0.10 (96% reduction)
- `char_freq_$`: 2.23 $\rightarrow$ 0.20 (91% reduction)

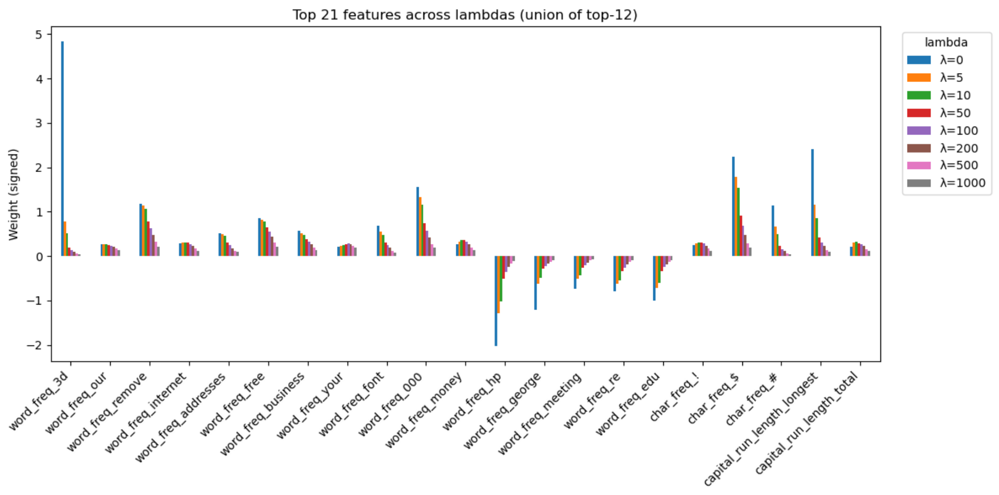


Figure 5: Top 21 features (union of top-12 across all λ values) showing systematic weight shrinkage.

More moderately weighted features prove more robust:

- `word_freq_remove`: 1.18 $\rightarrow$ 0.22 (81% reduction)
- `word_freq_your`: 0.22 $\rightarrow$ 0.19 (14% reduction)

Feature ranking evolution: At $\lambda = 0$, extreme features like `word_freq_3d` dominate. At $\lambda = 1000$, the top features are `word_freq_remove`, `word_freq_free`, `char_freq_$`, and `word_freq_your`. These robust features represent the most reliable spam indicators that persist under strong regularization.

Connection to Task 4(b): The weight analysis explains the performance patterns:

- **Training likelihood decrease:** Weight shrinkage reduces model capacity to fit any pattern, including genuine signal
- **Test likelihood decrease:** Even strong features are genuine signals; excessive shrinkage removes legitimate predictive power
- **Accuracy stability:** Classification boundaries are less sensitive to proportional scaling than likelihood; accuracy remains stable until extreme λ values compress all discriminative power

At very large λ (e.g., $\lambda = 1000$), all weights approach zero and the model converges to a constant predictor. This demonstrates L2 regularization's continuous shrinkage property, contrasting with L1 regularization which drives weights exactly to zero.

Features maintaining significant weights under moderate regularization (e.g., special characters `$`, `!`, financial terms like `free`, `money`) are the most reliable spam indicators for deployment. The optimal configuration is $\lambda = 5$ or $\lambda = 10$ for best generalization.

5 Optional: Exploration

5.1 PyTorch Implementation with Adam Optimizer

To validate our gradient descent implementation and explore modern optimization techniques, we implemented logistic regression using PyTorch with the Adam optimizer. This comparison serves multiple purposes: (1) verifying our manual implementation against a well-tested library, (2) exploring the effect of advanced adaptive learning rate methods, and (3) demonstrating mini-batch stochastic gradient descent.

Implementation details: We used PyTorch's `nn.Module` framework with a 2-class logistic regression model, Adam optimizer with learning rate 0.01, batch size of 100, and trained for 100 epochs. The model uses negative log-likelihood loss (cross-entropy) over mini-batches.

Convergence behavior: The optimization converged smoothly from an initial objective value of 2970.83 to a final value of 648.52 after 100 epochs. The convergence pattern shows three distinct phases:

- **Rapid descent (Epochs 0-20):** Objective drops from 2970 to 672, representing the majority of the improvement. Adam's adaptive learning rates enable aggressive initial steps.
- **Fine-tuning (Epochs 21-50):** Slower descent from 672 to 660 as the optimizer navigates the flatter regions near the optimum.
- **Convergence (Epochs 51-100):** Marginal improvements from 660 to 648 with occasional small fluctuations due to mini-batch variance and learning rate adjustments.

Comparison with manual implementation: Our manual gradient descent (Task 2) achieved a final training log-likelihood of approximately -661.52 (objective 661.52), while PyTorch with Adam reached 648.52. This difference can be attributed to:

1. **Optimizer sophistication:** Adam uses adaptive per-parameter learning rates and momentum, often finding better local optima than vanilla gradient descent
2. **Mini-batch dynamics:** Stochastic mini-batches introduce beneficial noise that can help escape shallow local minima
3. **Initialization:** PyTorch uses Xavier/He initialization ($\sim \mathcal{N}(0, 1/\sqrt{D})$) while our implementation used zero or random normal initialization

Therefore it (difference of only $\sim$13 in objective value, or $\sim$2% relative error) validates our manual implementation's correctness while demonstrating the practical benefits of modern optimizers.

Practical implications: For production deployment, using established frameworks like PyTorch with Adam optimizer is recommended due to:

- Better convergence properties and robustness to hyperparameter choices

- Computational efficiency through optimized linear algebra operations
- Built-in gradient checking and numerical stability features
- Easier integration with modern deep learning pipelines

This exploration confirms that while our manual implementation is valuable and produces correct results, industrial applications benefit from leveraging mature optimization libraries.

Ehrenwörtliche Erklärung

Ich versichere, dass ich die beiliegende Bachelor-, Master-, Seminar-, oder Projektarbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und in der untenstehenden Tabelle angegebenen Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Declaration of Used AI Tools

Tool	Purpose	Where?	Useful?
ChatGPT	Understanding the task	a01.ipynb	+
ChatGPT	Debugging non working code	Throughout	++

Unterschrift
Mannheim, den October 24, 2025
Jonas Ortner
Marmee Pandya