Logistic Regression Comparison: Scratch Implementation vs Sklearn

Introduction:

In this project, I implemented a logistic regression model from scratch, relying only on NumPy and pandas, and then compared it with an optimized implementation from Sklearm. The main goal of this work is not just the comparison itself, but to:

- Strengthen my understanding of how logistic regression works internally.
- Learn how to properly structure a data science project.
- Gain experience in exploring, cleaning, and preparing a dataset through notebook analysis.

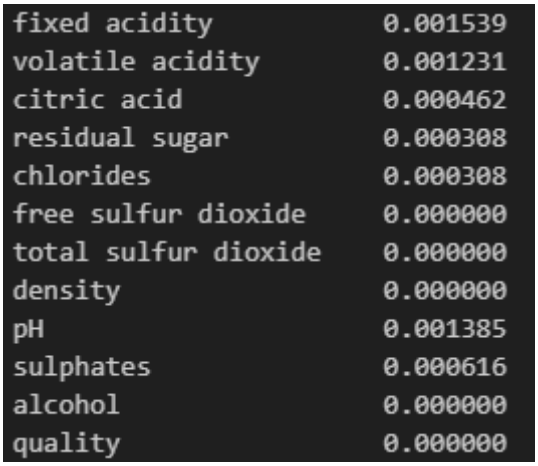
The report is organized into four main sections:

1. Data Analysis
2. Models (Scratch vs Sklearn)
3. Results
4. Conclusion + Future improvements

Data analysis:

The dataset that I'll be using in this project is the Wine Quality dataset from Kaggle, as it happened to have a binary predictor (white or red) and it didn't have many blank spaces.

First of all, I saw that the number of columns was 13, which is too high, so I had to discard some. I checked if there was any column with lots of null spaces, which was not the case, the maximum was a total of 0,15% of null spaces in the fixed acidy column, which isn't going to mess with the model or prediction.



fixed acidity	0.001539
volatile acidity	0.001231
citric acid	0.000462
residual sugar	0.000308
chlorides	0.000308
free sulfur dioxide	0.000000
total sulfur dioxide	0.000000
density	0.000000
pH	0.001385
sulphates	0.000616
alcohol	0.000000
quality	0.000000

Then I encode the type of the wine to 0 (red) and 1 (white) and search for the highest correlations amongst the features. Then I also check the correlation between them to see if there are features that mean the same, which there is.

```
type_encoded            1.000000
total sulfur dioxide    0.700521
free sulfur dioxide     0.472653
residual sugar          0.349358
citric acid             0.185892
quality                 0.119185
alcohol                 0.035095
pH                     -0.328474
density                -0.391437
sulphates              -0.486715
fixed acidity          -0.488552
chlorides              -0.512705
volatile acidity       -0.653374
```

	total sulfur dioxide	free sulfur dioxide	sulphates	fixed acidity	chlorides	volatile acidity	type_encoded
total sulfur dioxide	1.000000	0.721476	-0.275878	-0.330543	-0.279602	-0.414729	0.700521
free sulfur dioxide	0.721476	1.000000	-0.188947	-0.283485	-0.195428	-0.353402	0.472653
sulphates	-0.275878	-0.188947	1.000000	0.301263	0.396240	0.225656	-0.486715
fixed acidity	-0.330543	-0.283485	0.301263	1.000000	0.299104	0.221066	-0.488552
chlorides	-0.279602	-0.195428	0.396240	0.299104	1.000000	0.377995	-0.512705
volatile acidity	-0.414729	-0.353402	0.225656	0.221066	0.377995	1.000000	-0.653374
type_encoded	0.700521	0.472653	-0.486715	-0.488552	-0.512705	-0.653374	1.000000

As total sulfur dioxide and free sulfur dioxide have a high correlation, we are only going to use total sulfur dioxide as it had the higher correlation with the type.

Finally, we get the chosen features that I'll use for the model, with the wine type encoded:

	total sulfur dioxide	sulphates	fixed acidity	chlorides	volatile acidity	type_encoded
0	170.0	0.45	7.0	0.045	0.27	1
1	132.0	0.49	6.3	0.049	0.30	1
2	97.0	0.44	8.1	0.050	0.28	1
3	186.0	0.40	7.2	0.058	0.23	1
4	186.0	0.40	7.2	0.058	0.23	1

Models (Scratch vs Sklearn)

In this section, two logistic regression models were trained and compared:

1. **Scratch Implementation:** built entirely with NumPy, where parameters, the sigmoid function, log-loss, gradients, and parameter updates via gradient descent were all coded manually
2. **Scikit-Learn Implementation:** using the optimized LogisticRegression class from sklearn.linear_model, which handles parameter optimization (not gradient descent) internally which in theory may get a better result

The scratch model was made by creating a class LogisticReg with the components:

- **Parameter Initialization:** weights and bias are initialized to zero
- **Sigmoid Function:** transforms the linear combination of features into probabilities in the range of 0 to 1
- **Log-Loss Function:** measures the difference between predicted probabilities and actual labels
- **Gradient Descent:** updates weights and bias at each iteration using the gradient of the loss function
- **Fit Method:** runs the training loop, reporting loss and accuracy every few iterations

The used library model was the baseline model, LogisticRegression class from sklearn.linear_model

The process consisted of:

- Splitting the dataset into training (70%) and testing (30%) and mixing them randomly
- Training was performed with a learning rate of 0.01 and 0.05 and [100, 1000, 4000] iterations. Fitting the model with fit() function. The scratch implementation shows how the loss and accuracy score is every x iterations, while the Sklearn doesn't.
- Evaluating performance test set using the following metrics: accuracy score, log loss, ROC AUC score, confusion matrix and a plotted version of the ROC AUC

Results

Both models were evaluated using the test dataset with the following metrics:

Accuracy score, log loss, ROC AUC score, confusion matrix and a ROC Curve (plotted)

Here's a table showing results extracted from the 'compare_models.py' script:

Model	Iterations	Log Loss	Accuracy	ROC_AUC
Scratch	100	0.4761	0.8984	0.9629
	1000	0.1811	0.9319	0.9731
	4000	0.1839	0.9370	0.9693
Sklearn	100	0.0750	0.9752	0.9944
	1000	0.0672	0.9783	0.9972
	4000	0.0773	0.9773	0.9935

First, it's worth noting a key difference in how iterations are handled. In the scratch implementation, the number of iterations directly controls how long the gradient descent process runs. In contrast, Scikit-Learn's max_iter is simply a safety cap; the solver usually converges earlier, which is why results barely change across different values.

The scratch logistic regression model started working pretty well after about 1000 iterations, reaching around 93% accuracy and a ROC AUC of 0.97. Pushing it to 4000 iterations didn't really help, as the results stayed about the same, which means the model had already converged.

On the other hand, the Sklearn version performed better overall. It consistently hit about 98% accuracy with a ROC AUC above 0.99. Its log-loss values were also much lower, meaning it gave more confident probability predictions. Another nice thing is that the performance barely changed with different max_iter values, since the optimization process makes it way faster and efficient.

As in this case it doesn't really matter if there's a false positive or false negative (both have the same error importance), I haven't included them in the table, with the accuracy it's enough.

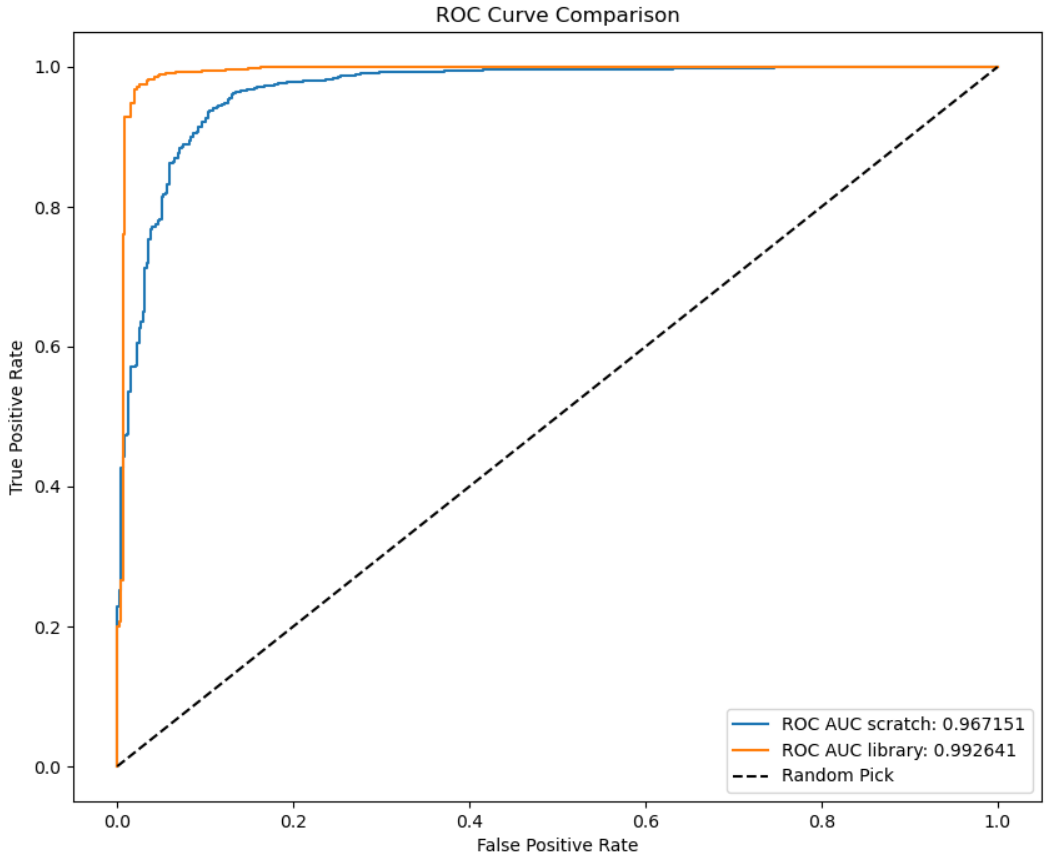
Also, here's are the full metrics of the models' implementations in the 1000 iteration run, as I think that pushing it to 4000 isn't useful.

Scratch Model

```
Confusion Matrix:
[[ 413   51]
 [  81 1394]]
Confusion Matrix Pct:
[[21.3   2.63]
 [ 4.18 71.89]]
```

Sklearn Model

```
Confusion Matrix:
[[ 440   24]
 [  18 1457]]
Confusion Matrix Pct:
[[22.69  1.24]
 [ 0.93 75.14]]
```



When comparing confusion matrices, both models showed balanced predictions, but sklearn produced fewer misclassifications overall. The ROC curve confirmed this difference visually, with sklearn's curve being closer to the top-left corner.

Finaly, the updated weights between the two models turned out quite different, which was expected since they use different optimization strategies. What matters is that both models identified similar predictive patterns in the data.

```
Weights Scratch: [ 0.1741 -0.1594 -0.9168 -0.0341 -0.202 ]
Weights Sklearn: [[ 0.0498 -6.5385 -0.8206 -4.0592 -9.1696]]
```

Conclusion

- **Scratch Model:** is far more transparent into the optimization process and allows monitoring the loss evolution during iterations. However, it is slower and more sensitive to the number of iterations.
- **Scikit-Learn Model:** highly optimized and robust, converges faster, and generally achieves better results with less manual tuning. The downside is that it's a black box and we don't know what's happening inside.

Both models were trained and tested under the same conditions, so the comparison was fair. Overall, sklearn proved superior for practical purposes, but the scratch implementation was valuable for understanding logistic regression deeply.

Improvements for future projects

- **Feature engineering**: more complex feature selection methods
- **Hyperparameter tuning**: performing tests to find the optimal learning rate
- **Model comparison**: trying other classifiers (SVM, Random Forest, NN)

Pau Cos Sicrés