DSM Project

Thijmen Brand, Alpay Demirci, Adil Yahia

January 15, 2026

1 Introduction

Olive flies are among one of the many species of flies that live around plants and stocks. They are considered as being a pest to the olives. That is why it can be beneficial to olive farmers to identify the flies on their stocks. There are however many more types of flies that exist on the olives that might not be harmful to the plants. Therefore a method of just counting every fly would not work as that would result in false data and potentially incorrect and inadequate pest control. In order to be able to better control the flies and protect the olives, a farmer must know roughly how many olive flies there actually are on the olives.

This project aims to answer the question:

¿ How can a logistic regression model be constructed that is capable of identifying olive flies?

This kind of classification model is a perfect example of multiple logistic regression and a good first project for new learners. The project involves important fundamental steps like data pre-processing, model construction and training and model evaluation. This report describes and outlines the process of developing such a machine learning model. It aims to outline the methods used to pre-process the data and construct the model itself, the results of the model and closes with an analysis and discussion about pre-processing and training alternatives and approaches.

2 Methods

2.1 Data collection and pre-processing

The dataset consists of two image classes provided as part of the assignment: images containing olive flies and images containing other insects or objects found on olive plants. All images were resized to a fixed resolution of 170×170 pixels to ensure consistent input dimensions. Pre-processing focused on isolating the insect from the background. Foreground extraction was performed using grayscale conversion followed by inverse Otsu thresholding, assuming the insect appears darker than the background. Morphological closing was applied to reduce noise, after which the largest connected component was selected as the foreground region. All non-foreground pixels were replaced with a white background. To remove poorly extracted images, the ratio of foreground pixels to total pixels was computed for each image. Images exceeding a manually chosen threshold were considered faulty and removed from the dataset. Data augmentation was applied to the olive fly class using horizontal flipping and small rotations of ± 15 degrees to increase variation. After cleaning, the dataset was rebalanced by augmenting the olive fly class and randomly down sampling the non-olive fly class so that both classes contained an equal number of images.

2.2 Feature extraction

Histogram of Oriented Gradients features were extracted to capture shape and edge information of the insects. Images were converted to grayscale and normalized before computing HOG descriptors with fixed orientation bins, cell size, and block size. Color information was captured using normalized three-dimensional RGB color histograms. Each image was converted to RGB space, and a fixed number of bins per channel was used to represent color distribution. In addition to using HOG and color features separately, a combined feature representation was created by concatenating both feature vectors. This allowed the model to leverage both structural and color-based information during classification.

2.3 Model training

2.3.1 Sigmoid function

The nature of the problem is a binary classification problem. An object in an image must be identified a "olive fly" or "not olive fly". Because of this very nature of the problem, a logistic regression model is chosen for the classification task. A logistic regression model is well suited for these types of binary classification because it outputs a value between 0 and 1. Therefore we can apply the following logic:

$$\hat{y} = \begin{cases} 1, & \text{if } f(x) \geq 0.5, \\ 0, & \text{if } f(x) < 0.5. \end{cases}$$

A linear regression model works using the sigmoid function. This function essentially squeezed the output of the model between 0 and 1 by using the logarithm. It can be noted as

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Here, z is essentially our multiple linear regression function $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$
In the code, this sigmoid function is implemented using the numpy library.

```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

2.3.2 Forward propagation

Of course, when the model is not yet trained, we do not know the $\beta_{1..n}$. However, to predict an outcome we must solve the multiple linear regression problem, to afterwards feed it to the sigmoid function in order for it to output the a prediction between 0 and 1. This value can then be interpreted as $true$ or $false$ and we have our prediction.
Mathematically, this is represented as

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

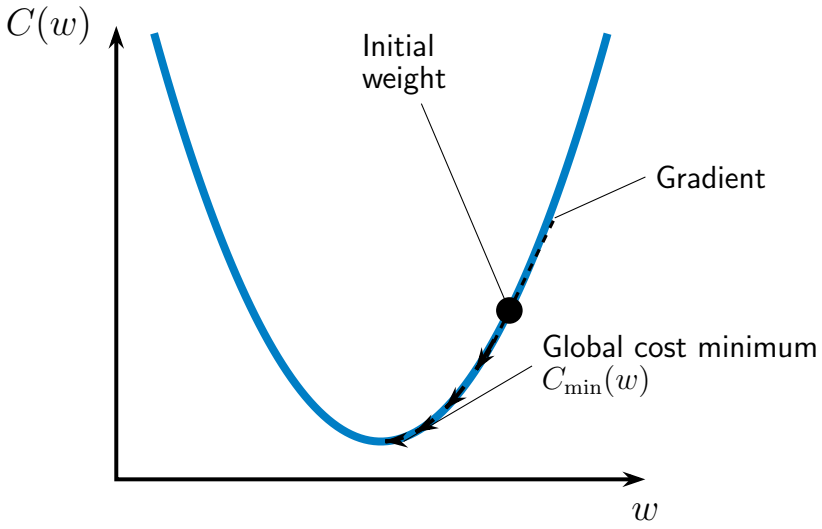
That is, actually, everything that pertains to the forward propagation from a mathematical perspective. There is, however, a big optimization technique that we can use to speed up the multiple linear regression computation. Because all dependent variables $X = \{x_1, x_2, \ldots, x_n\}$, independent variables $Y = \{y_1, y_2, \ldots, y_n\}$, and weights and bias can be represented using matrices.

$$X = \begin{bmatrix} & x_{11} & x_{12} & \cdots & x_{1m} \\ & x_{21} & x_{22} & \cdots & x_{2m} \\ & x_{31} & x_{32} & \cdots & x_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ & x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix}, \qquad Y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}, \qquad \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \vdots \\ \beta_m \end{bmatrix} \qquad \beta_0 = 0.0$$

We can then use clever matrix operations to calculate the predictions in a batch, so we can calculate all the predictions at once using these matrix operations. We can calculate $\hat{y}$ taking the dot product of $Y = X \times \beta + \beta_0$ And then we can directly feed the result into the sigmoid function. In code this is implemented as follows

```
Y_hat = sigmoid(np.dot(X, w) + b)
```

Figure 1: Cost Function in Logistic Regression in Machine Learning



2.3.3 Gradient decent and backward propagation

With weights and biases of 0 we cannot make accurate predictions. For that we need to train the model. And during the training we want to know how well our model is doing, so we can tweak some parameters if necessary. For the evaluation of the model we use a loss function. This function captures our models' predictions with how far it is off from reality. So to calculate this, we need the predictions and the actual labels that we classified the data with ourselves.

In logistic regression, we use the log loss function, or cross-entropy loss [GeeksforGeeks, nda]. This function is noted as follows

$$C(h\sigma(x), y) = -\frac{1}{n}\sum_{i=1}^{n}\Big(y_i \log\big(h_\sigma(x_i)\big) + (1-y_i)\log\big(1-h_\sigma(x_i)\big)\Big) \tag{1}$$

This cost function captures the how far our model is off from the true predictions we gave it. And in order to train our model, the goal is to minimize this cost function, such that the loss, or error, of the model becomes as small as possible.

This loss function is implemented as the following code:

```
cost =
    (-1/n) * np.sum(Y * np.log(Y_hat) + (1-Y) * np.log(1-Y_hat))
```

This training is then done using gradient decent. With gradient decent our goal is to minimize the cost function to the local minima. We can see this as a parabolic function, we are somewhere along the slope, and we want to step down towards the valley of that slope.

To do this, we have a two main parameters we can tweak. That is the weights, and the bias. Because $X = \{x_1, x_2, \dots x_n\}$ are our fixed features. We calculate the new values for the weights and biases by calculating the partial derivative of the cost function, with respect to the weights and biasses. When we use the chain rule on the cost function, we see that the partial derivatives, with respect to the weights and biasses become fairly simple [Karunakaran, 2023].

$$\frac{\partial}{\partial b}C(w,b) = \frac{\sum_{i=1}^{n} h\sigma(x_i) - y_i}{n}$$
$$\frac{\partial}{\partial w}C(w,b) = \frac{\sum_{i=1}^{n} (h\sigma(x_i) - y_i)x_i}{n}$$

Using these partial derivatives, we can then update the respective weights and biasses to make the model (hopefully) more accurate, and decrease the loss.

This process of gradient decent can also be done using matrices and matrix operations. In this formula we see $h\sigma(x_i) - y_i$, which calculates the residual. Or in other words, it calculates the amount the models' prediction was off from the actual label. In order to calculate these residuals in bulk, we can easily just subtract our label matrix from the predictions matrix, which gives us a residual matrix.

$$r = \begin{bmatrix} \hat{y}_0 - y_0 \\ \hat{y}_1 - y_1 \\ \hat{y}_2 - y_2 \\ \vdots \\ \hat{y}_n - y_n \end{bmatrix}$$

In order to calculate the partial derivative for the weights. we must multiply the residual once more with the feature $((h\sigma(x) - y)x)$. However when we want to do this in batch, using matrix operations, we must first transpose the feature matrix X such that the shapes are compatible. After we have done this, we can easily use the following formula to calculate the partial derivatives for w and b.

$$\frac{\partial}{\partial w} C(w) = X^T r/n$$
$$\frac{\partial}{\partial b} C(b) = r/n$$

In the code, this is implemented as follows

```
residuals = Y_hat - Y
XTransposed = np.transpose(X)

dw = (1/n) * (np.dot(XTransposed, residuals))
db = (1/n) * np.sum(residuals)
```

In order to update the respective weights and bias, we must take the our previous weights and bias, and subtract the learning rate (α) multiplied by the partial derivative from that.

$$w' = w - \alpha \frac{\partial}{\partial w} C(w)$$
$$b' = b - \alpha \frac{\partial}{\partial b} C(b)$$

In code this is implemented as follows

```
w = w - learning_rate * dw
b = b - learning_rate * db
```

And that concludes one iteration of training. After this iteration, you propagate forward again, calculate the loss and update the weights and bias. For the full model training, the amount of iterations and the learning rate are arbitrarily determined. Often a learning rate of 0.1 is much too large.

2.3.4 Model training

As outlines in the previous section, the model must be trainged for n amount of itterations in order to tweak the weights and biasses enought so they make accurate predictions. The model must be so to say: "Fitted" to the data. For the model described in this report, a total of 2000 iterations have been used. This is then combined with a 0.01 learning rate to alter the weights and bias with.

Testing happens with the pre-processed and feature extracted data. However, if all of this data is used for only trianing purposes, we have no good way to accurately evaluate the model for its performance. Therefore we have split the dataset in a training section, and a testing section. With a distribution of 80% and 20% respctively. The training set contained 1600 samples, and therefore does the testing set contain 400 samples.

3 results

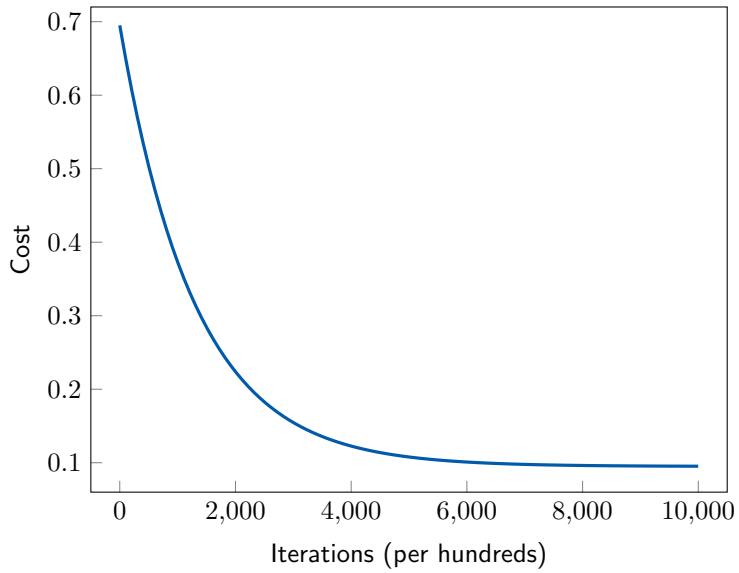
3.1 Experimental setup and training

The logistic regression model was trained using a Gradient Decent approach. Based on experimental tuning of the iterations and learning rate, the following parameters have been selected.

$$\text{Learning rate } (\alpha) : 0.01 \tag{2}$$

$$\text{Iterations} : 10000 \tag{3}$$

During the training phase, the loss function, outlined in the loss function1, showed a good and steady convergence which decrease significantly to leater stabilize. This stabilization means a local minimum has been found, and the model found optimal weights. The final training accuracy the model achieved was 99.19%. This indicates that the model has successfully learned and included the features in the training data, into the model.



In this figure, the x-axis represents the iterations (per hundreds) and the y-axis represents the loss. This figure demonstrates a successful training sequence.

3.2 Model performance evaluation

When the testing was complete, the model was evaluated on the remaining 20% of the data. Using this data, the model achieved a test accuracy of 86.75%. In order to understand the model's performance in more detail, a confusion matrix has been constructed. This matrix measures how well a classification model is performing by comparing the predictions with the actual results.

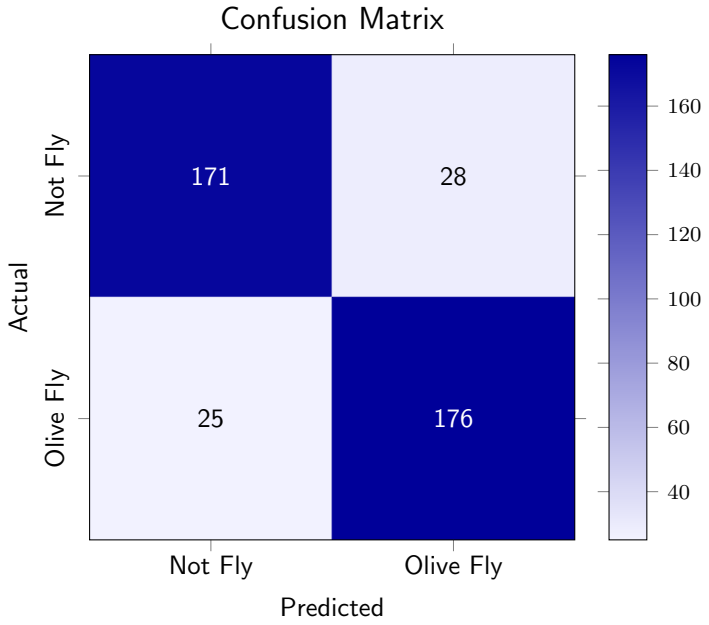
The confusion matrix is composed of four categories that measure the mistakes.

- True Positive (TP): The model correctly predicted a positive outcome. e.g. the model predicts a fruit fly, when there is a fruit fly. - True Negative (TN): The model correcty predicted a negative outcome. e.g the model does not predict a fruit fly, when there is no fruit fly. - False Positive (FP): The model incorrectly predicted a positive outcome. e.g. the model predicts a fruit fly, when there is not a fruit fly. - False Negative (FN): The model incorrectly predicted a negative outcome. e.g. the model does not predict a fruit fly, when there is a fruit fly.

For this model, the confusion matrix looks a follows:

Using these values, key measures like accuracy, precision and recall can ben calculated. These give a better idea of the performance of the model. We can calculate these measures using the following formulas:

Actual \Predicted	Not Fly	Olive Fly
Not Fly	171	28
Olive Fly	25	176



$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$
$$\text{Precision} = \frac{TP}{TP_FP}$$
$$\text{Recall} = \frac{TP}{TP + FN}$$
$$\text{F1-Score} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

For this model results are as follows:

Measure	Value
Precision	0.87
Recall	0.86
F1	0.87

In this figure, the F1 score is a harmonic mean of the precision and recal, and was measured at 0.87. This metric confirms that the model is robust and does not rely solely on accuracy. Wich can be a misleading metric in imbalanced datasets. The data set used for this model however, was carefully balanced during pre-processing.

The results gathered from the model evaluation show that the logistic regression model is highly effective for this binary classification task. It does also contain a considerable error of $13,25\%$. This error is likely due to complex background noise or similarities in shape between olive flies and other insects.

4 Analysis and Discussion

4.1 General model performance

The difference between the training accuracy of $99,19\%$ and the testing accuracy of $86,75\%$ indicates that the model might be overfitting. The model has learned specific noise and patterns of the testing set very effectively, but struggles to generalize in that seem degree of accuracy to unseen data. While the F1 score of 0.87 proves that the model is robust and effective for binary classification, does the 12.44% drop in accuracy seem to show that the model fitting is too specific to the training examples. A cause for this could be the fact that the high-dimentional feature vectors used (dimention 26000) capture too much background noise.

4.2 Dimensionality reduction (PCA)

To address this high dimension the might cause the model to overfit, we can employ a technique that captures the most important information in a lower dimension, while disregarding the insignificant noise. One of the many techniques to do this is called Principal Component Analysis (PCA). This was implemented in experimentation to reduce the feature dimension, while retaining the most significant variance and thus, theoretically removing noise and speeding up the training process [GeeksforGeeks, ndb].

And while the training speed increased significantly due to the reduced number of computations required per iteration, did the model's accuracy not improve. The model even slightly worsened in accuracy after the application of PCA. Because of this downgrade in performance was the PCA removed and has the model been trained on raw, high-dimensional features.

4.3 Hyperparameter optimization

An automated search logarithm was implemented that tests for various combinations of the hyperparameter. It tried combinations of the learning rate and number of iterations such that the accuracy would be maximized. The learning rate ranged from 0.001 to 0.1 and the iterations from 1000 to 20000.

The analysis of these tests however showed that the model is rather insensitive to hyperparameter changes once the stable convergence threshold is reached. While a low learning rate slowed down convergence and a high one cause it, did a lot of "reasonable" paramteres between these thresholds produced insignificant differences in the final accuracy of the model. This can mean that the model has hit a performance cieling that is imposed by the linearity of the logistic regression, and the quality of the extracted features.

4.4 Feature extraction and alternatives

The current approach of feature extraction relies on HOG and Color histograms, as described in feature extraction2.2. And while these appear to be effective, do the features determine what the model "sees". Which is edges and color distributions in our case. This error rate of 12.44% is likely due to the fact that insects that are not olive flies, are similar in color and structural shapes. And therefore do these two feature extraction methods do not differentiate well between olive flies and other insects.

A potential improvement for future research would be the implementation of Deep Learning, and more specifically a Convolutional Neural Network (CNN) [Esan et al., 2024]. Unlike logistic regression, which requires this manual feature extraction, do CNN's learn optimal feature representations directly from the raw pixel data. And thus, would it likely generalize better on the data, and capture more abstract patterns.

5 Conclusion

This project has been done to determine how a logistic regression model could be implemented in order to effectively identify olive flies. The model has been implemented by first pre-processing the data, doing feature extraction to finally train the model and perform the binary classification.

The main research question asked for this project was: **How can a logistic regression model be constructed that is capable of identifying olive flies?**

This questions can be answered: a logistic regression model that can identify olive flies can be constructed by combining features like a histogram of oriented gradients (HOG) and a color histogram. This combination allows

the model to differentiate the shape and color patterns of an olive fly over other insects. It does this with a final test accuracy of $87,75\%$ and F1 score of $0,87$, providing enough proof that this a robust way for this classification task.

However, the difference between the training- and test accuracy of 12.44% is an indicator that the model is likely overfitting. Further analysis showed that techniques like PCA do not improve overall performance and that tuning the hyperparameter showed negligible performance gains. However, despite the overfitting, the project shows to be a success and this machine learning model can be effectively applied to solve the problem.

References

[Esan et al., 2024] Esan, A., Okomba, N., Adebiyi, T., Adio, M., Olajide, D., and Adebisi, K. (2024). Development of a Clean Water Detection System Using Convolutional Neural Network-Based Model. *Journal Name Missing*, 3:13–18.

[GeeksforGeeks, nda] GeeksforGeeks (n.d.a). Cost function in Logistic Regression in Machine Learning. Accessed: 2024.

[GeeksforGeeks, ndb] GeeksforGeeks (n.d.b). Principal Component Analysis (PCA). Accessed: 2024.

[GeeksforGeeks, ndc] GeeksforGeeks (n.d.c). Understanding the Confusion Matrix in Machine Learning. Accessed: 2024.

[Karunakaran, 2023] Karunakaran, D. (2023). Logistic regression using gradient descent.

[Manolache, 2024] Manolache, G. (2024). Ds_Reader_June_2024.pdf. Unpublished manuscript. In Canvas.