MTH786P Final Project - Diabetes Prediction
Thanh Trung Vu - 230849442

1 Introduction

This project utilises the Diabetes Dataset[4], originally sourced from the National Institute of Diabetes and Digestive and Kidney Diseases. The primary goal is to develop a predictive model that accurately determines the presence of diabetes in patients based on various diagnostic measurements.

This project begins with a comprehensive examination of the dataset, involving an analysis of each feature, followed by appropriate data processing. Subsequently, different classification algorithms, namely K-Nearest Neighbors (KNN), Logistic Regression, and Support Vector Machine (SVM), are discussed and implemented. The models are then trained and evaluated using specified performance metrics, including accuracy and F1 score. In the final stage, the results are analysed to compare the effectiveness of the different models and interpreted which features hold the highest predictive power in addressing the diabetes prediction problem.

2 Data Analysis and Processing

2.1 Preliminary Analysis

The dataset comprises of diagnostic data from 768 female individuals of Pima Indian heritage, all aged 21 or above. It includes the following eight features and one target variable. All data are numerical with no missing values (NaN).

- `Pregnancies`: Number of times pregnant.
- `Glucose`: Plasma glucose concentration post 2 hours in an oral glucose tolerance test.
- `BloodPressure`: Diastolic blood pressure (mm Hg).
- `SkinThickness`: Triceps skin fold thickness (mm).
- `Insulin`: 2-hour serum insulin (mu U/ml).
- `BMI`: Body mass index (weight in kg / (height in m^2)).
- `DiabetesPedigreeFunction` (`DPF`): A function indicating diabetes pedigree.
- `Age`: Age in years.
- `Outcome`: Binary variable indicating diabetes diagnosis (0 = No, 1 = Yes).

We can see that this dataset contains information that could be valuable in predicting diabetes. It seems plausible that there is a significant link between diabetes and variables such as glucose concentration and insulin levels. Additional indicators, such as age and body mass index (BMI), may also provide insights into an individual's overall health, which might aid in forecasting diabetes risk. One could, for instance, speculate that a person with a high BMI is at higher risk of obesity, which in turn may increase their chances of developing diabetes.

An initial assessment reveals certain anomalies in features such as `Glucose`, `BloodPressure`, `SkinThickness`, `Insulin`, and `BMI`. The presence of biologically implausible zero values in these features suggests missing or incorrect data. Since the number of features is reasonably low, a closer examination of each feature is carried out which can help us decide the appropriate data cleaning strategies.

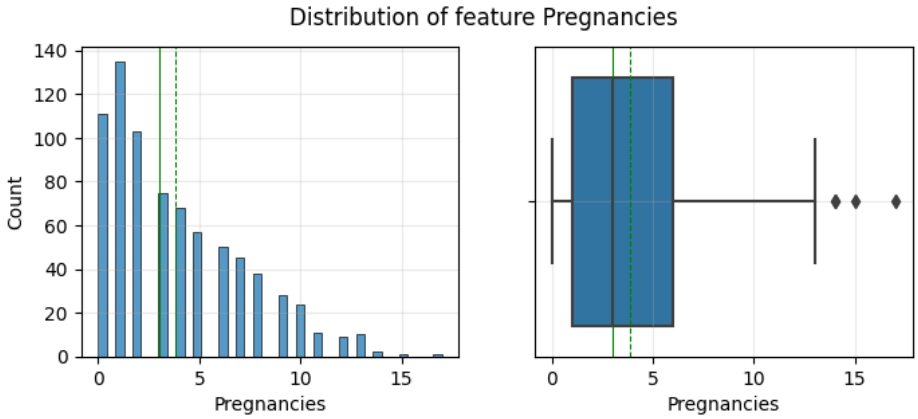
2.2 Feature-Specific Analysis

	count	mean	std	min	25%	50%	75%	max
`Pregnancies`	768.0	3.85	3.37	0.00	1.00	3.00	6.00	17.00
`Glucose`	768.0	120.89	31.97	0.00	99.00	117.00	140.25	199.00

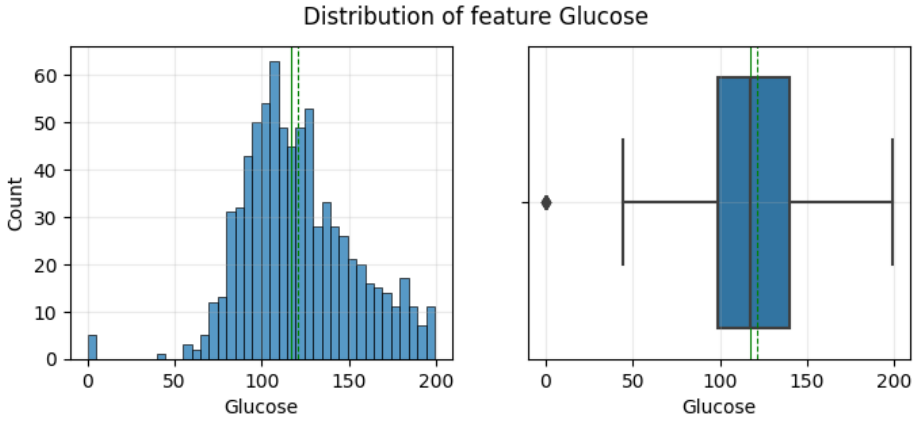
Key statistics of the features `Pregnancies` and `Glucose`

In this section, we demonstrate the analysis and processing of two features: `Pregnancies` and `Glucose`. The analysis and processing of the remaining features follow a similar methodology.

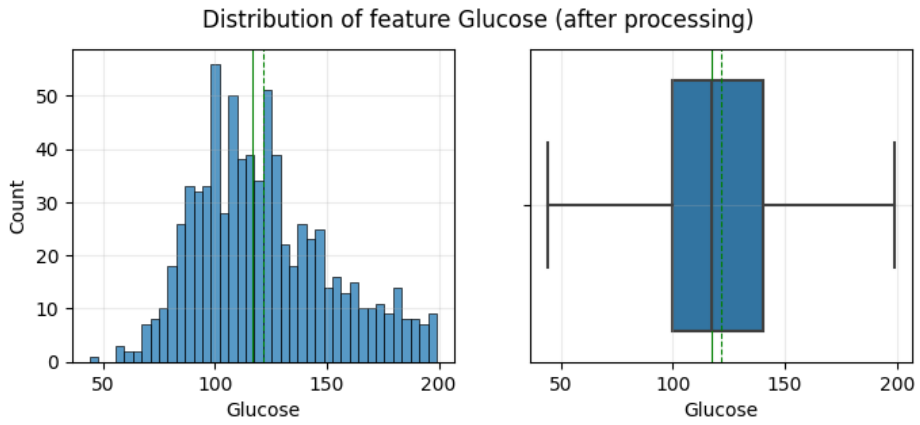
Regarding the feature `Pregnancies`, we can see that the number of pregnancies ranges from 0 to 17. With an average of approximately 3.8 pregnancies per individual, the data exhibits a reasonable range, given the demographic profile of the participants. Large values such as 17 and 15 may raise concerns, but we can disregard them for the time being.



The feature `Glucose` also contains zero values. Glucose measurements of zero are biologically implausible and likely indicate missing data.



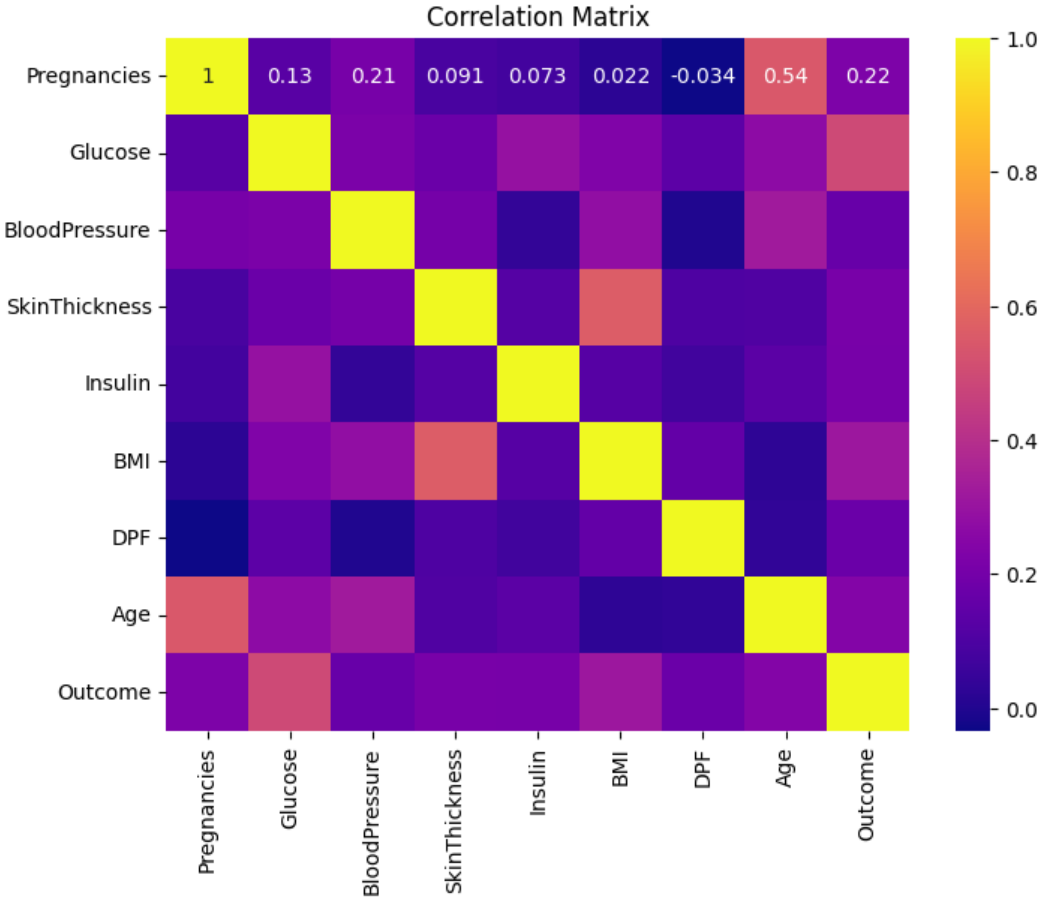
To ensure more accurate representation, these zero values are replaced with the mean of the non-zero measurements. After processing, the distribution appears more biologically plausible.



Similar examinations and data cleaning procedures are conducted for other features with implausible zero values, such as `BloodPressure`, `SkinThickness`, `Insulin`, and `BMI`. Each feature is analysed for its distribution and validity, ensuring that the dataset is ready for training a classification model.

2.3 Correlation Analysis

To visually grasp the relationships among the features, a correlation matrix is generated. Looking at the relationships between the features can help us choose the most appropriate set of features for the predictive modeling.



Correlation matrix after processing the data

We can see that some notable correlations exist between `SkinThickness` and `BMI`, as well as `Pregnancies` and `Age`. This suggests that each pair could be reduced to one feature to lower the dimensionality.

We can also see a significant correlation between `Glucose` and the `Outcome` variable, highlighting its potential as a significant predictor for diabetes. On the other hand, there is relatively weak correlation between `BloodPressure` and the `Outcome`, suggesting its limited predictive power.

These observations can help guide the selection of features for training, which can be helpful for algorithms sensitive to high dimensionality, such as the K-Nearest Neighbours (KNN) algorithm. Nonetheless, in this report, all features have been utilised. My experimentation indicates that selecting features does not significantly improve the models' performance. Moreover, this allows all features in the dataset to be assessed for their relative importance.

2.4 Target Variable Analysis

Finally, a quick look at the target variable `Outcome` reveals an imbalance in the dataset, where around 65% of the instances indicate not having diabetes. This imbalance is taken into account when selecting performance metrics.

3 Methods

3.1 Data Standardisation

Since our diabetes dataset is highly non-uniform, with some features having a much larger value range than others, we will apply data standardisation before passing the data to a learning model. Standardisation ensures that different features of objects are on almost the same scale, guarantees that each feature is equally important, and facilitates processing by learning algorithms.

3.2 Choosing Models

In this section we consider a few different learning algorithms to apply to our diabetes classification problem and describe their usage.

3.2.1 K-Nearest Neighbours (KNN)[1]

The K-Nearest Neighbours (KNN) algorithm is one of the simplest classification methods. For a given input, probabilities for the class labels are computed by analysing the k nearest neighbours. The class label corresponding to the highest probability is then chosen as the output. In our implementation, the Euclidean distance metric is employed. The probability is determined based on the proportion of the label. The label "`1`" is assigned if over 50% of the neighbours possess this label.

The number of neighbours, k, is a hyperparameter that can be determined through model selection strategies. In this project, a grid search is conducted for k varying from 1 to 20. As with all classifiers, the model is evaluated using K-fold cross-validation.

3.2.2 Logistic Regression[1]

Logistic regression for classification problems is similar to normal regression problems, except that the output is discrete. For binary classification, the prediction $f(\mathbf{x}, \mathbf{w})$ yields a probability.

To achieve this, we consider $\sigma(f(\mathbf{x}, \mathbf{w}))$ instead of $f(\mathbf{x}, \mathbf{w})$, where $\sigma : (-\infty, \infty) \to [0, 1]$ is the logistic function or sigmoid function, defined as

$$\sigma(x) := \frac{1}{1 + \mathrm{e}^{-x}}$$

The goal is to solve a minimisation problem where we define and minimise a function $\mathrm{L}(\mathbf{w}) : \mathbb{R}^{d+1} \to \mathbb{R}$ of $d + 1$ dimensional argument $\mathbf{w} = (w_0, w_1, \ldots, w_d)$. Here $\mathrm{L}(\mathbf{w})$ is called the cost function and $w_0, w_1, \ldots, w_d$ are the weights. In this project, $\mathrm{L}(\mathbf{w})$ is linear, which simplifies finding its gradient $\nabla \mathrm{L}(\mathbf{w})$. This gradient is then used in the gradient descent method to solve this minimisation problem computationally by following the update rule:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \tau \nabla L\left(\mathbf{w}^{(k)}\right)$$

where $\tau > 0$ is the step-size and $k \geq 0$ is the iteration number. In this project, $\mathbf{w}^{(0)}$ is initially set as a zero vector. For the K-fold cross validation demonstration, gradient descent is set to terminate when k reaches $10,000$. However, for the bootstrap sampling demonstration, the maximum number of iterations is reduced to 1000 to keep the computational runtime reasonable. The hyperparameter $\tau > 0$ is also tuned using grid search.

Besides implementing logistic regression in `NumPy`[5], I also compare it with the `scikit-learn`[3] implementation. The iteration numbers for K-fold cross-validation and bootstrap sampling demonstrations are set to the same values as above.

3.2.3 Support Vector Machine (SVM)[1]

A limitation of logistic regression is that the hyperplane that spans the decision boundary is not necessarily optimal in the sense that it maximises the distance between the closest data points on each side of the decision boundary. This feature can, however, be achieved in the context of binary classification with the Support Vector Machine (SVM) algorithm. The key idea is to both maximise the distance between the closest data points to the hyper-plane and, at the same time, ensure that the each data point ends up on the correct side of the decision boundary.

For implementing SVM in this project, the I also use the `scikit-learn`[6] library.

In addition to the three previously mentioned classification algorithms, I also experimented with a simple 2-layer neural network model. Unfortunately, the outcomes were not promising, as the model consistently predicted `0` regardless of the input. Given that 65% of our data are labelled `0`, this resulted in a misleading accuracy rate of about 65%, while the F1 score metric (which is described in the next section) was actually 0. Due to these limitations, I have decided not to include this model in the results section.

4 Results and Interpretation

This section presents the metrics and methods used to evaluate the performance of the models and interpret the results.

4.1 Performance

4.1.1 F1 Score

We have found that our dataset exhibits an imbalance, with a higher occurrence of labels marked as `0` compared to those marked as `1`. To ensure a more reliable evaluation of the model,

the F1 score[2] is used alongside accuracy. The F1 score provides a more balanced metric, less impacted by label prevalence. Its calculation is as follows:

$$F_1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

where Precision $= \frac{\text{True Positives}}{\text{True Positives+False Positives}}$ and Recall $= \frac{\text{True Positives}}{\text{True Positives+False Negatives}}$.

By convention, $F_1 = 0$ when the number of true positives is 0.

4.1.2 Performance Evaluation using K-Fold Cross Validation

To reflect the real-world performance of a model, it is important to evaluate the model using data it has not seen during training. We use K-fold cross validation to achieve this. The K-fold cross validation method can be summarised as follows:

- Divide the dataset, which includes both inputs and outputs, into approximately equal parts, denoted as $D_1, D_2, \ldots, D_K$.
- For each i from 1 to K, evaluate the optimal weights $\mathbf{w}_i$ for the logistic regression evaluated over the dataset $D_1, D_2, \ldots, D_{i-1}, D_{i+1}, \ldots, D_K$ (excluding D_i), and a corresponding validation error L_i evaluated over the set D_i.
- Compute the average of the optimal weights, $\hat{\mathbf{w}} = \frac{1}{K}(\mathbf{w}_1 + \mathbf{w}_2 + \ldots + \mathbf{w}_K)$, and the average validation error, $L = \frac{1}{K}(L_1 + L_2 + \ldots + L_K)$.

Throughout this project, the number of folds K is set to 5. This means that for each fold in the cross-validation process, the dataset is divided into a training set and a validation set with an $80:20$ ratio.

4.1.3 Hyperparameter Tuning using Grid Search

The performance of each learning algorithm depends on its hyperparameters. In the case of logistic regression, the hyperparameter is the step size τ, while for K-Nearest Neighbours (KNN), it is the number of neighbors k.

A common approach to tuning these hyperparameters is to conduct a grid search. During a grid search, the model is trained multiple times using different hyperparameter values.

The results of the grid search show that, for this diabetes binary classification problem, the optimal step size for logistic regression is $\tau = 0.01$, and the optimal number of neighbors for KNN is $k = 11$.

For the implementations of logistic regression and Support Vector Machine (SVM) provided by `scikit-learn`[3, 6], the default parameters are currently being used.

4.1.4 Metrics

Table 1 summarises the results of the experiments conducted in this project. All results were obtained using K-fold cross-validation with $K = 5$. To ensure reproducibility, the random state was set to a consistent number 123456789, once before training each model. The results may vary to some extent depending on the randomising procedure.

	Accuracy	F1 score	Precision	Recall
Logistic Regression, step size = 0.01	0.768254	0.632143	0.571292	0.710268
Logistic Regression using `scikit-learn`	0.768254	0.632143	0.571292	0.710268
SVM using `scikit-learn`	**0.773466**	**0.63687**	0.57163	**0.723763**
KNN, k = 3	0.705747	0.570164	0.55851	0.58545
KNN, k = 11	0.744835	0.611983	**0.5763**	0.655297

Table 1: Average metrics of different learning algorithms

The highlighted numbers in the table represent the maximum values within their respective metric categories.

It is interesting to see that the implementation of Logistic Regression using only `NumPy`[5] produced the same results as the `scikit-learn`[3] implementation, even though the optimal weight values, which will be detailed below, are different. Assuming that the `scikit-learn` implementation is reliable and well-established, this adds confidence in the correctness of the more "primitive" implementation, although perfectly identical results may also be a cause for concern. It is also worth noting that the relatively small size of the dataset may increase the likelihood of getting the same result.

Overall, the SVM model yielded the highest results in accuracy and F1 score as well as recall, although both logistic regression implementations were also very closely matched in performance. KNN with $k = 11$ also exhibited strong performance and achieved the highest precision, with other metrics not too far off compared to the other models. On the other hand, the worst-performing implementation was KNN with $k = 3$, which is not surprising given that 3 is not the optimal hyperparameter for k in this diabetes binary classification problem.

4.2 Feature Ranking with Bootstrap Sampling

A natural question arises: which features hold the highest predictive power in predicting the risk of diabetes? To determine a ranking for the features, we can utilise the relative order of the optimal weights. Instead of relying on a single set of weights, bootstrap sampling helps us obtain a range of values for each weight which can paint a more accurate picture of the relative ordering.

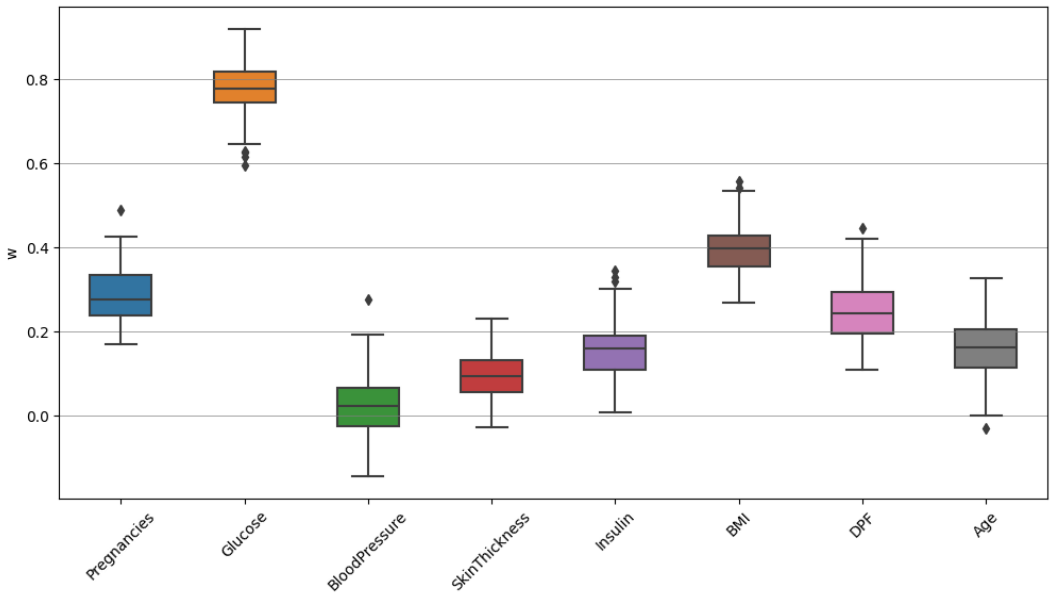
In each iteration of bootstrap sampling, new samples are created by randomly selecting data points from the original dataset with replacement. In this project, these samples are of size 691 each, equivalent to 90% of the original dataset's size. The number of samples generated is limited to 100 to keep computational runtime manageable. Bootstrap sampling is not applied to the K-Nearest Neighbours (KNN) model since it does not generate feature weights.

The results are summarised in Table 2. It should be noted that these values only represent the mean weights. We also generate some box plots for the `NumPy`[5] logistic regression implementation to get a better idea of the distribution around the median. Besides slight variations in the scales of the weight values, both the logistic regression and SVM implementations within the `scikit-learn`[3, 6] library produce very similar rankings of feature importance.

Based on the results obtained after employing bootstrap sampling, it is clear that the most important feature by far is `Glucose`, which makes sense given the context of predicting diabetes. `BMI` comes second in the ranking. Conversely, the features `BloodPressure` and `SkinThickness` are among the least important, as indicated by their consistently low or negligible weight values.

	Logistic Regression step size = 0.01	Logistic Regression using `scikit-learn`	SVM
`Pregnancies`	0.28580	0.44800	0.34882
`Glucose`	**0.77665**	**1.07446**	**0.87862**
`BloodPressure`	0.02127	-0.09453	-0.06719
`SkinThickness`	0.09370	0.01352	-0.05411
`Insulin`	0.15653	0.18429	0.06856
`BMI`	0.39362	0.63153	0.47631
`DiabetesPedigree Function`	0.24669	0.28085	0.22302
`Age`	0.15539	0.12877	0.04258

Table 2: Average weights of all features after bootstrapping 100 times



Box plots comparing the feature weights from Logistic Regression using `NumPy`

5 Conclusion

In this project, we examined three different learning algorithms and assessed the performance of five different implementations in solving the diabetes binary classification problem. The performances of these algorithms are remarkably similar when optimal hyperparameters are used.

A potential extension to this project is the exploration of additional classification algorithms, such as decision trees and random forests. Additionally, within the domain of logistic regression, the introduction of regularisation parameters could be considered.

Expanding upon this project could also involve identifying the optimal stopping point for gradient descent. By plotting the result of the cost function after each iteration and observing when the curve begins to level off, we can determine the most effective number of iterations to balance accuracy and computational efficiency.

6 Libraries

The following list contain the primary libraries used in the coding of this project.

- `NumPy` : for numerical representations and operations, including vector and matrix multiplication.
- `matplotlib` and `seaborn` : for data visualisation, including the creation of histograms and box plots.
- `pandas` : for data analysis and manipulation tasks, such as value replacement.
- `scikit-learn` : for comparisons with other learning algorithms.
- `random` : for generating random numbers, particularly in K-fold cross-validation and bootstrap sampling.

References

[1] Martin Benning and Nicola Perra. *Lecture Notes*. Originally developed by Martin Benning. Adapted and modified by Nicola Perra. Sept. 2023.

[2] *F1 Score*. https://en.wikipedia.org/wiki/F1_score. Accessed on January 24, 2024.

[3] *Logistic Regression — scikit-learn 1.4.0 documentation*. https://scikit-learn.org/stable/modules/linear_model.html. Accessed on January 24, 2024.

[4] National Institute of Diabetes and Digestive and Kidney Diseases. *Diabetes Data Set*. Donor of database: Vincent Sigillito (vgs@aplcen.apl.jhu.edu). Date received: 9 May 1990. Accessed on January 24, 2024. URL: https://www.kaggle.com/datasets/mathchi/diabetes-data-set/data.

[5] *NumPy*. https://numpy.org/. Accessed on March 30, 2024.

[6] *Support Vector Machines — scikit-learn 1.4.0 documentation*. https://scikit-learn.org/stable/modules/svm.html. Accessed on January 24, 2024.