INTRODUCTION

This report covers creating a machine learning model that can predict credit card approvals based on a machine learning methodology applied to a real dataset. The report is separated into three main parts, which are Data Processing, Training and Evaluation.

PART 1: DATA PROCESSING

At first, the provided dataset was loaded by using the Pandas read_csv() function. Then the closer inspection of the dataset was done. Dataset consists of 16 features with a mix of nominal and continuous values.

(a) Cleaning the data

The dataset contained a few missing values; after inspection, it was found the missing values are represented by '?'. Samples with missing values were deleted to clean the dataset from incomplete data. The question mark was replaced by NaN, and samples containing NaN were removed by using panda's dropna() function. The missing data could be substituted with, for example, mean or median values, but in this case, it was preferred not to create any artificial values since only 37 out of 690 samples were erased.

(b) Splitting the data into test/train set

Before splitting, it was distinguished between input and output data. The output data labelled as y are column A16 (returns whether the credit was approved), and the rest are input data labelled as X. Consequently, it splits the data into a training set used for training the data (creating the model for predictions) and a test set used solely for testing. The function train_test_split() from scikit-learn was used with the parameter stratify = output values, and the test/train ratio was set to 20/80. This parameter splits data so that the proportion of output values in the dataset will be the same in the training and testing set. This prevents from having only one output value in the training set and the other in the test set (extreme case).

(c) Encoding, conversion, and scaling

Values in some columns are nominal; therefore, they are converted to numeric values because logistic regression does not work with non-numerical inputs. One hot encoding was used, which assigns 1 to one of the values and 0 to others. Where are true/false values k-1 dummy variables were encoded, but where are more than two possible values, it had to be encoded into k dummy variables to keep equality between them.

As mentioned, logistic regression works with numbers; thus, an inspection of data types was done, and columns A2 and A14 were converted into appropriate data types.

Machine learning algorithms do not perform well when the numerical input attributes have very different scales; therefore, the standardisation was used for scaling. Data is transformed to take properties of normal distribution ($\mu = 0$, $\sigma = 1$) by the formula below.

$$x' = \frac{x - \mu}{\sigma}$$

Figure 1: Standardisation formula

Unlike min-max scaling, it does not have specific bound values, but standardisation is much less affected by outliers, which may be an important feature in this case because outliers have not been explicitly excluded. The argument against standardisation might be when looking at histograms of the data; they are not normally distributed. Therefore, min-max scaling was also tested in the evaluation part, but it did not perform differently.

(d) Data leakage

To prevent data leakage, the order of action is very important. Data were split into the training and testing set right after cleaning the data. There was not done any statistical analysis on training data, and fitting the scaler was done after splitting, solely with training data. Also, for better data representation, parameter stratify was used while splitting the data.

PART 2: TRAINING

For this example, logistic regression is used as a linear model for binary classification. It uses a logistic function that returns a number between 0 and 1, which models the probabilities describing the possible outcomes. LogisticRegresion algorithm from the scikit-learn library is used to train the model. This is done by fitting training data (output and scaled input data). It takes multiple parameters that manipulate with the type of logistic regression, e.g., using different solvers.

(a) Best and worst classification accuracy

In this scenario is used unregularised (penalty='none') logistic regression, where all the classes have weight one.

For the worst classification accuracy, it is possible to create the model without training the input data and use only output data for training. The model would take $x^{(i)} = 1$ for all cases and only consider the relative frequency of outputs which calculates how often something happens divided by all outcomes. This model would always achieve accuracy higher or equal to 50%.

To fit the best logistic regression model, which means finding the best regression parameters, it calculates the likelihood function (that assumes Bernoulli distribution for binary classification), which is then maximised by using optimisation technique (maximising likelihood is the same as minimising loss function). The solver parameter determines the optimisation technique in scikit-learn. There is no closed-form solution, but its loss function is convex, so optimisation is guaranteed to find the global minimum (if the learning rate is not too large, and it waits long enough).

Logistic regression can use higher-order polynomials just as linear regression. This allows for creating a more complex decision boundary. If the polynomial has enough degrees, it can achieve 100% accuracy on the training set. The problem is that such a model does not generalise enough and thus performs poorly on the testing set. This is caused by overfitting the model. A small training error does not mean a small testing error.

(b) Explain parameters

penalty – This parameter identifies which regularisation technique should be used to reduce overfitting. This is done by constraining the weights of the model, e.g., ‘l2’ represents l_2 norm of the weight vector. Where norms are various distance measures.
tol – This is the number that says to optimisation technique (solver) to stop searching for a minimum when this value is achieved.
max_iter – This number determines the maximum number of iterations of the solver to find the minimum.

(c) class_weight = ‘balanced’

When one of the target class’s values has a higher occurrence then the other, it means it is imbalanced, which can cause bias or skewness towards the majority value. When there is no parameter passed, all classes have weight 1. In this case, it has the parameter ‘balanced’; its aim is to penalise the misclassification made by the minority class by increasing its class weight and decreasing the majority class’s weight. So higher class-weight means it puts more emphasis on a class.

In this scenario, the output class is relatively evenly distributed, so this parameter will not have much effect on the final predictions.

(d) Explain predict(), decision_function(), and predict_proba()

predict_proba() – The trained model estimates the probability that an instance belongs to a particular class. This function returns that value for a particular sample.
predict() – This function predicts a particular class label for a particular sample. Therefore, when the estimation from predict_proba() is greater than 50%, then the model predicts that the instance belongs to that class.
decision_function() – This function outputs a signed distance to the separating hyperplane and tells whether it lies on the hyperplane’s right or left side. This is equal to the confidence score of that sample which says how exact match it was.

PART 3: EVALUATION

(a) Classification accuracy

Accuracy_score function was used to calculate the classification accuracy. It is calculated by dividing the number of all correct predictions by the number of all samples (figure 2). It also contains parameter normalise; when set to False, it returns the number of correctly classified samples.

$$\text{accuracy} = \frac{\text{total correct predictions}}{\text{total predictions made}}$$

Figure 2: Accuracy formula

The classification accuracy of my model is 0.847.

(b) Balanced accuracy

The function called balanced_accuracy_scare was used. It is an especially useful score when the classes are imbalanced. It is calculated by the formula in figure 3 where *tp* = number of true positives, *tn* = number of true negatives, *fp* = number of false positives and *fn* = number of false negatives.

$$\text{balanced accuracy} = \left[\frac{tp}{tp+fp} + \frac{tn}{tn+fn}\right]/2$$

Figure 3: Balanced accuracy formula

The balanced accuracy of my model is 0.854.

(c) Confusion matrix

Unbalanced

		Predicted	
		Negative -	**Positive +**
Actual	**Negative -**	**48**	**11**
	Positive +	**9**	**63**

Balanced

		Predicted	
		Negative -	**Positive +**
Actual	**Negative -**	**50**	**9**
	Positive +	**10**	**62**

Since the data are not significantly imbalanced, there are not any significant differences.

(d) Precision-recall curve and the Average Precision

Precision-recall is a very useful measure when the classes are significantly imbalanced. The formulas for calculating the precision and recall are listed below with the same annotation mentioned above.

$$precision = \frac{tp}{tp + fp} \qquad recall = \frac{tp}{tp + fn}$$

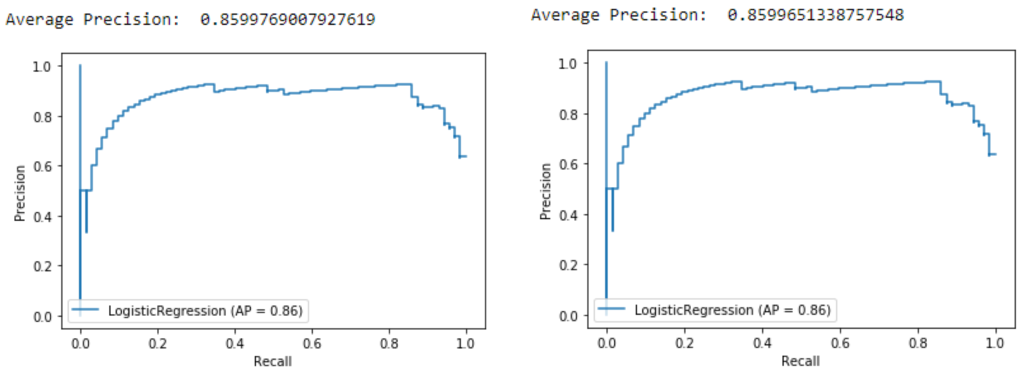
Figure 4: Precision and recall formula

Average Precision is the weighted mean summarisation of a precision-recall curve according to the increase in recall, in figure 5. It basically measures the area under the precision-recall curve; the bigger the area is, the better performance.

$$AP = \sum_{n} (Recall_n - Recall_{n-1}) Precission_n$$

Figure 5: Average Precision

In figure 6, there is the plot of the precision-recall curve with the average precision weight none on the left and balanced on the right.



```
Average Precision:  0.8599769007927619
```

```
Average Precision:  0.8599651338757548
```

Figure 6: Plot of the precision-recall curve (weight none on the left and balanced on the right)

PART 4:

(a) Penalty l2

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} y^i \log(h_\theta(x^i)) + (1 - y^i) \log(1 - h_\theta(x^i))$$

Figure 7: Cost function

$$\frac{\partial}{\partial \theta_j} J(\theta) = \sum_{i=1}^{m} (h_\theta(x_i) - y_i) x^i_j + 2\lambda \sum_{j=1}^{m} \beta_j$$

Figure 8: Gradient

(b) Implementing polynomial

There is an attempt to do the polynomial expansion, but it is overfitting. It made 946 features from 42. This is because the second degree's polynomial will covert, e.g., two features a, b to [1, a, b, a^2, ab, b^2]; thus, the number of features entered to polynomial transformation polynomially grows since it has to do a large number of combinations. The polynomial expansion is powerful because it allows creating a more complex decision boundary. However, there comes a risk of overfitting and interpretation of features that might be confusing.

(c) Comparing regularised and unregularised

As mentioned above, penalty parameter l2 is the norm that penalises weights. It adjusts them, so it makes the fitting function smoother, which prevents overfitting. In this case, the results from logistic regression using the l2 parameter compared to those that are not much different since there is no overfitting. Its effect is usually more visible when the fitting function is polynomial. However, the advantage of regularisation is that it improves numerical stability, so it is often used.

(d) Solvers

Scikit-learn provides various solver that are optimisation techniques that are trying to find a global minimum of the cost function. Logistic regression does not have a closed-form solution, so it finds minimum iteratively.

A second-order optimisation method is 'newton-sg' because it computes the second-order derivatives. The difference is that second-order calculates the hessian (which causes slower performance with larger datasets). The hessian gives it the information about the curvature of the cost function, giving it the ability to change step in iteration appropriately.

Stochastic Average Gradient descent 'sag' is fast because it picks random instances in the training set, computes the average and computes the gradients based on that. It is a combination of gradient descent and incremental aggregated gradient that uses a random sample of previous gradient values.