# INTRODUCTION

In this project the data is related with direct marketing campaigns of a Portuguese banking institution, which were predominantly based on phone calls. Often, more than one contact to the same client was required, in order to assess if the product (bank term deposit) would be (or not) subscribed. The classification goal is to predict if the client is likely to subscribe to a term deposit. For this task we used a pre-existing data set with 45,211 examples.

To accomplish the classification task we made use of the Logistic Regression algorithm, Support Vector Machine algorithm and Neural Networks algorithm. Each of these algorithms was run multiple times using different regularization techniques, hyperparameters and feature transformations.

Before starting off with using the machine learning algorithms, we examined the data and noticed that most of our examples were class 0, which means that any machine learning algorithm we used was more likely to predict class 0.

## Pre-processing the Dataset

In this project, we conducted pre-processing on the raw dataset to make it suitable for machine learning algorithms. The pre-processing steps included cleaning the data, encoding variables, and scaling numerical features.

## Cleaning the Dataset

The initial dataset contained inconsistencies, missing values, and irrelevant features that did not contribute to the prediction of the target variable. To address these issues, we performed the following data cleaning steps:

1. Extracted and organized relevant features, such as age, marital status, education, and balance, among others. We paid a lot of attention to features such as how long the person stayed on the phone and whether or not they previously took out a loan of some kind because those seemed like factors that would weigh heavily.

2. Replaced missing features with default features and analyzed how many missing features there were per entry. Features that were consistently missing such as method of contact were dropped. Roughly 2700 data points were missing with 1840 being in areas such as success or marital status with 2 main choices followed by another 1857 in education. Unknown marital status was converted to single because the quantity of single people in the dataset outweighed the number of married people. Education was generalized to be the lowest level of education as we saw that many of those who did not give out their education were also working jobs that had a lower minimum level of education.

We also started analyzing the data from line 99, skipping the first 98 lines, as they contained metadata and not actual data points. We also transformed the raw input into a more structured format, creating a new CSV file with cleaned data as opposed to using the original arff file.

**Encoding**

Several features in the dataset were categorical with linguistic data. Encoding these categorical variables enabled us to transform the data into a more suitable format for machine learning algorithms. We applied custom encoding functions to map categorical values to numeric ones:

1. For the marital status and outcome of the previous contact features, we used a custom function pos_or_neg that mapped 'divorced' and 'failure' to -1, 'married' and 'success' to 1, and all other cases to 0.
2. For the education feature, we used a custom function map_education that mapped 'secondary' to 1, 'tertiary' to 2, and all other cases to 0.
3. For binary categorical features, such as credit in default, housing loan, and personal loan, we directly converted 'no' to 0 and 'yes' to 1.

**Data Scaling**

After cleaning and encoding the data, we scaled the numerical features using the scale_features function. This function uses the StandardScaler() package from scikit-learn to perform feature scaling. The scaling process involved:

1. Splitting the dataset into training and testing sets where we used a 70/30 split.
2. Applying the StandardScaler() to the training set using the fit_transform method.
3. Applying the same scaler to the test set using the transform method.

This scaling process ensured that all features were on the same scale, allowing the machine learning algorithm to perform more efficiently. The scaled features were then used as input for model training and evaluation. This process was applied individually to all 3 of the models we created.

## Method 1: Logistic Regression

**Introduction:**

One of the first models we decided to use for our classification problem was logistic regression. We used logistic regression because it is a popular and widely successful model used when the goal is to predict the probability of a binary outcome (such as yes or no, true or false, 0 or 1) based on one or more input features. The first time we used logistic regression on the data we had a test accuracy of 87%. So we decided to use hyperparameter tuning to try and improve the accuracy and reduce bias and variance.

**Hyperparameter Tuning:**

To perform hyperparameter tuning, we utilized the GridSearchCV function from the scikit-learn library. The primary objective was to optimize two hyperparameters for the logistic regression model: the penalty type (L1 or L2) and the inverse of regularization strength (C) with values of [0.001, 0.01, 0.1, 1, 10, 100]. We conducted a 5-fold cross-validation during the grid search to estimate the model's performance using different hyperparameter combinations.

During the tuning process, we systematically assessed the impact of the L1 and L2 penalties alongside various values of C. The L1 penalty encourages sparsity, driving some feature weights to zero, while the L2 penalty results in a more balanced distribution of feature weights. By exploring both penalties and a range of C values, we aimed to identify the optimal regularization technique and strength for our model.

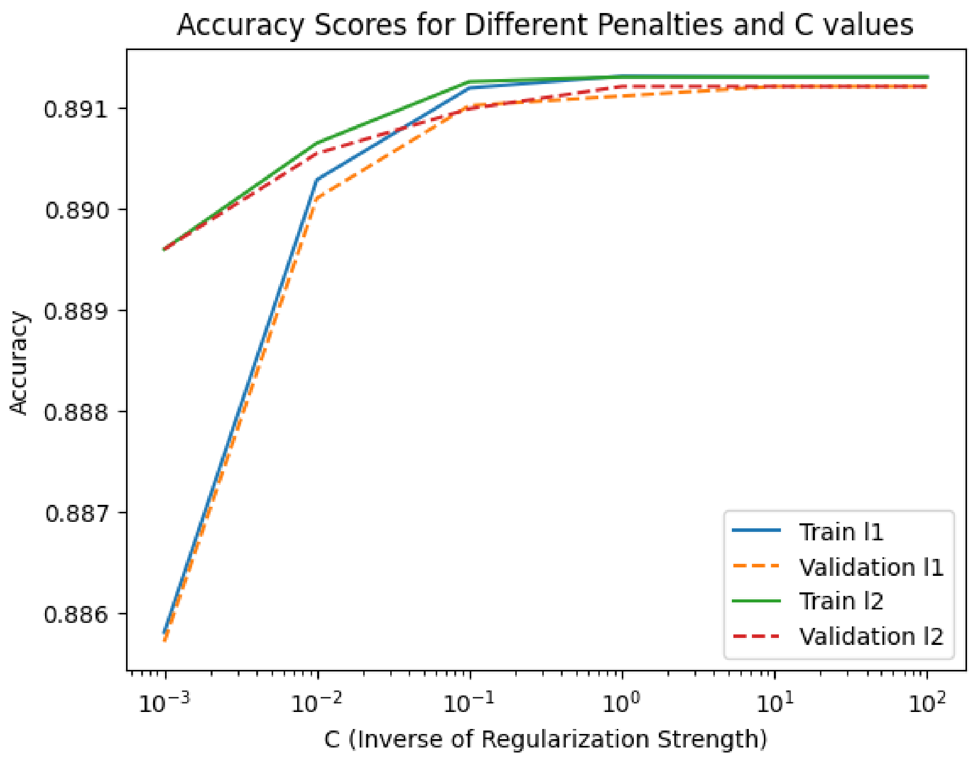

**Figure 1:** L1 and L2 regularization for various C values.

Results received for L1 and L2 regularization along with C values

Results for Penalty: l1
C: 0.001 Train Accuracy: 0.8868 Validation Accuracy: 0.8867
C: 0.01 Train Accuracy: 0.8907 Validation Accuracy: 0.8906
C: 0.1 Train Accuracy: 0.8915 Validation Accuracy: 0.8910
C: 1.0 Train Accuracy: 0.8917 Validation Accuracy: 0.8913
C: 10.0 Train Accuracy: 0.8917 Validation Accuracy: 0.8912
C: 100.0 Train Accuracy: 0.8917 Validation Accuracy: 0.8912
Results for Penalty: l2
C: 0.001 Train Accuracy: 0.8900 Validation Accuracy: 0.8899
C: 0.01 Train Accuracy: 0.8912 Validation Accuracy: 0.8913
C: 0.1 Train Accuracy: 0.8916 Validation Accuracy: 0.8913
C: 1.0 Train Accuracy: 0.8917 Validation Accuracy: 0.8912
C: 10.0 Train Accuracy: 0.8917 Validation Accuracy: 0.8912
C: 100.0 Train Accuracy: 0.8917 Validation Accuracy: 0.8912

**Model Training and Evaluation:**

The optimal hyperparameters we got for logistic regression were C = 100 and regularization = L2. After obtaining the best hyperparameters, we trained the logistic regression model accordingly. The training process involved adjusting the model's weights to minimize the error between predicted outcomes and actual target values. With the optimal hyperparameters, our model was expected to perform well on the test set without overfitting or underfitting.

We then evaluated the model's performance by predicting the labels for the test set and calculating the accuracy score, which provided us with a quantitative measure of the model's effectiveness. By comparing the accuracy scores for different hyperparameter configurations, we could determine which combination yielded the best results for our logistic regression model.

**Visualization of Model Performance:**

To gain deeper insights into the model's behavior, we plotted the training and validation accuracy scores for different penalties and C values. The line chart used a logarithmic x-axis to display the C values, allowing for better visualization of the results across a wide range of values. This graphical representation helped us assess the model's performance during the tuning process and identify potential overfitting or underfitting issues.

The plot enabled us to visually examine how the model's performance changed with different hyperparameter configurations. A clear understanding of the trade-offs between the L1 and L2 penalties, as well as the impact of different C values, allowed us to make informed decisions about the best hyperparameter settings for our specific problem.

**Results:**

The grid search revealed the best combination of hyperparameters for the logistic regression model. By visualizing the training and validation accuracy scores for different penalties and C values, we were able to assess the model's performance and identify potential overfitting or underfitting issues. Hyperparameter tuning helped us improve our test accuracy from 87% to 89.1%.

Final Confusion Matrix

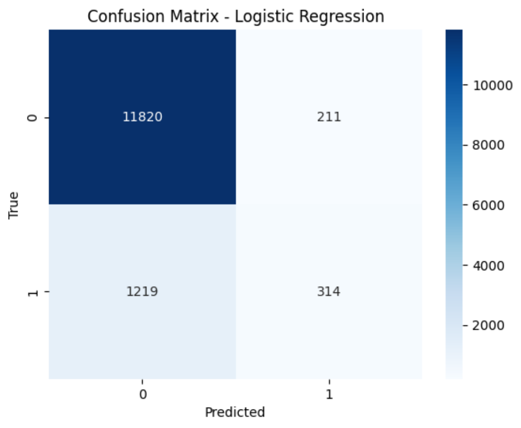

**Figure 2:** Final Confusion Matrix

## Method 2: SVM

**Introduction:**

For our classification problem, we also tried using support vector machines (SVM). SVM is a popular model used for classification problems where the goal is to separate the data into two classes by finding the best hyperplane that maximizes the margin between the two classes. We decided to use SVM as it can handle non-linearly separable data by using a kernel function to transform the input data into a higher dimensional space.

**Hyperparameter Tuning:**

To perform hyperparameter tuning for SVM, we utilized the GridSearchCV function from the scikit-learn library. The primary objective was to optimize three hyperparameters: C with values [0.01, 0.1, 1, 10], kernel type [‘linear’, ‘rbf’], and gamma with a range of values [0.001, 0.01, 0.1, 1]. We conducted a 5-fold cross-validation during the grid search to estimate the model's performance using different hyperparameter combinations.

The C hyperparameter controls the tradeoff between achieving a low training error and a low testing error by balancing the magnitude of the weights. The kernel type hyperparameter specifies the type of kernel function to be used, with linear and radial basis function (RBF) being the most commonly used. The gamma hyperparameter determines the width of the kernel function and controls the smoothness of the decision boundary.

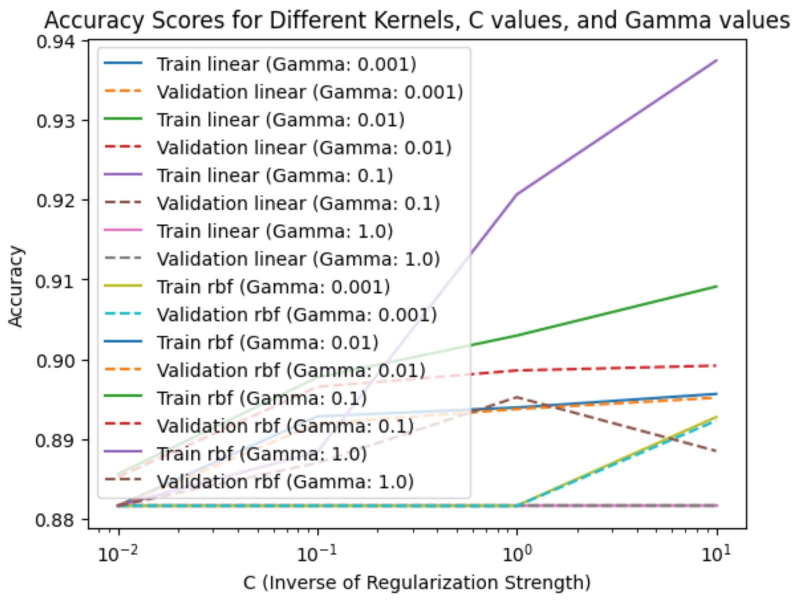

**Figure 3: kernel function and gamma values for various C values.**

Results:

| C | Kernel | Gamma | Train Accuracy | Validation Accuracy |
|---|---|---|---|---|
| 0.01 | linear | 0.001 | 0.881663 | 0.881663 |
| 0.01 | rbf | 0.001 | 0.881663 | 0.881663 |
| 0.01 | linear | 0.010 | 0.881663 | 0.881663 |
| 0.01 | rbf | 0.010 | 0.881663 | 0.881663 |
| 0.01 | linear | 0.100 | 0.881663 | 0.881663 |
| 0.01 | rbf | 0.100 | 0.885629 | 0.885266 |
| 0.01 | linear | 1.000 | 0.881663 | 0.881663 |
| 0.01 | rbf | 1.000 | 0.881663 | 0.881663 |
| 0.10 | linear | 0.001 | 0.881663 | 0.881663 |
| 0.10 | rbf | 0.001 | 0.881663 | 0.881663 |
| 0.10 | linear | 0.010 | 0.881663 | 0.881663 |
| 0.10 | rbf | 0.010 | 0.892849 | 0.891933 |
| 0.10 | linear | 0.100 | 0.881663 | 0.881663 |
| 0.10 | rbf | 0.100 | 0.897731 | 0.896546 |
| 0.10 | linear | 1.000 | 0.881663 | 0.881663 |
| 0.10 | rbf | 1.000 | 0.888402 | 0.887035 |
| 1.00 | linear | 0.001 | 0.881663 | 0.881663 |
| 1.00 | rbf | 0.001 | 0.881671 | 0.881632 |
| 1.00 | linear | 0.010 | 0.881663 | 0.881663 |
| 1.00 | rbf | 0.010 | 0.893955 | 0.893766 |

| | | | | |
|---|---|---|---|---|
| 1.00 | linear | 0.100 | 0.881663 | 0.881663 |
| 1.00 | rbf | 0.100 | 0.902969 | 0.898600 |
| 1.00 | linear | 1.000 | 0.881663 | 0.881663 |
| 1.00 | rbf | 1.000 | 0.920632 | 0.895251 |
| 10.00 | linear | 0.001 | 0.881663 | 0.881663 |
| 10.00 | rbf | 0.001 | 0.892770 | 0.892312 |
| 10.00 | linear | 0.010 | 0.881663 | 0.881663 |
| 10.00 | rbf | 0.010 | 0.895662 | 0.895188 |
| 10.00 | linear | 0.100 | 0.881663 | 0.881663 |
| 10.00 | rbf | 0.100 | 0.909107 | 0.899201 |
| 10.00 | linear | 1.000 | 0.881663 | 0.881663 |
| 10.00 | rbf | 1.000 | 0.937395 | 0.888520 |

**Model Training and Evaluation:**

The grid search revealed the best combination of hyperparameters for the SVM model. The optimal hyperparameters were C = 0.1, kernel = RBF, and gamma = 0.1. After obtaining the best hyperparameters, we trained the SVM model accordingly. The training process involved finding the best decision boundary that maximizes the margin between the two classes.

We then evaluated the model's performance by predicting the labels for the test set and calculating the accuracy score, which provided us with a quantitative measure of the model's effectiveness. By comparing the accuracy scores for different hyperparameter configurations, we could determine which combination yielded the best results for our SVM model.

**Visualization of Model Performance:**

To gain deeper insights into the model's behavior, we plotted the training and validation accuracy scores for different hyperparameter configurations. The plot helped us assess the model's performance during the tuning process and identify potential overfitting or underfitting issues.

The SVM model's performance was visualized in the table provided. The model's accuracy was calculated for different combinations of hyperparameters, such as C, kernel, and gamma. The results were plotted in a table with the accuracy scores for the train and validation sets. The table helped us visualize how the SVM model's performance changed with different hyperparameter configurations.

**Results:**

The grid search revealed that the best combination of hyperparameters for the SVM model was C = 0.1, kernel = RBF, and gamma = 0.1. By visualizing the training and validation accuracy scores for different hyperparameter configurations, we were able to assess the model's performance and identify potential overfitting or underfitting issues. Hyperparameter tuning helped us improve our test accuracy from 87.2% to 88.6%. The confusion matrix below shows that the number of false negatives far exceeded the number of positives showing a large problem with our model.

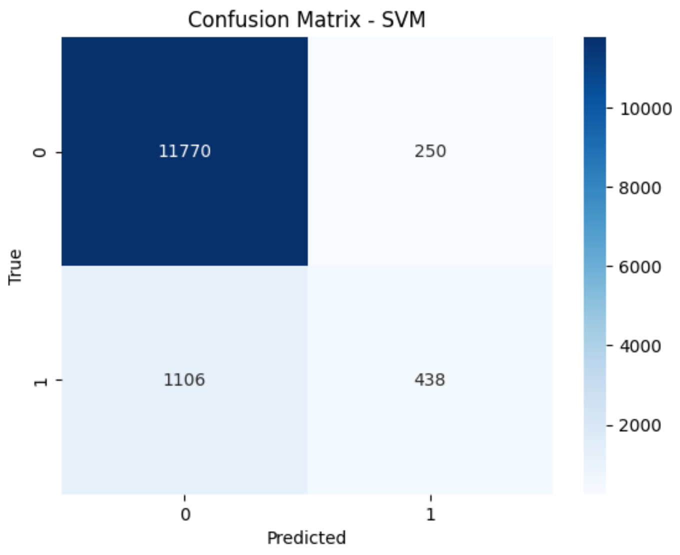

**Figure 4: Final Confusion matrix**

## Method 3: Neural Networks

**Introduction:**

The last model we decided to use for our classification problem was Neural Networks. We used Neural Networks because they are capable of learning complex non-linear relationships between the input features and the output label, which help improve the accuracy of our model. We used a neural network with different hyperparameters to classify our data.

**Hyperparameter Tuning:**

To start, we used a neural network with one hidden layer and varied the size of the layer to [10, 50, 100]. We also used three different activation functions, relu, logistic and tanh. Additionally, we used different regularization parameters (alpha) to prevent overfitting of the model. We trained and tested the neural network model and obtained the training and validation accuracy for each hyperparameter configuration.

To identify the optimal hyperparameter configuration, we compared the training and validation accuracy scores for each configuration. We found that the neural network with a single hidden layer of size 100 and activation function 'tanh' had the highest validation accuracy. We also found that regularization with alpha=0.001 gave the best results.

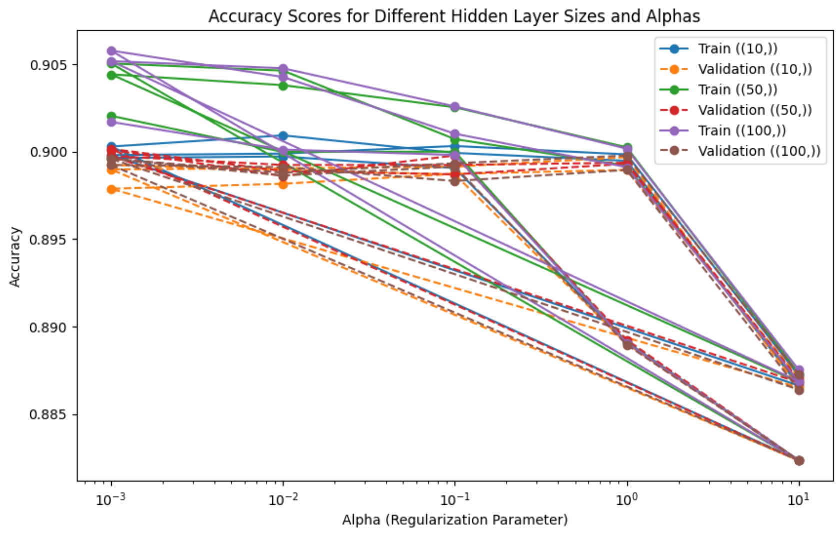

**Figure 5: hidden layer size and activation function type for various alpha values**

Results:

| Hidden Layer Sizes | Activation | Alpha | Train Accuracy | Validation Accuracy |
|---|---|---|---|---|
| (10,) | logistic | 0.001 | 0.901792 | 0.901507 |
| (50,) | logistic | 0.001 | 0.902826 | 0.901570 |
| (100,) | logistic | 0.001 | 0.903458 | 0.901223 |
| (10,) | logistic | 0.010 | 0.901965 | 0.900496 |
| (50,) | logistic | 0.010 | 0.901871 | 0.901570 |
| (100,) | logistic | 0.010 | 0.902139 | 0.901286 |
| (10,) | logistic | 0.100 | 0.900875 | 0.900496 |
| (50,) | logistic | 0.100 | 0.901823 | 0.901634 |
| (100,) | logistic | 0.100 | 0.901705 | 0.901096 |

|  |  |  |  |  |
|---|---|---|---|---|
| (10,) | logistic | 1.000 | 0.892170 | 0.892438 |
| (50,) | logistic | 1.000 | 0.892146 | 0.892186 |
| (100,) | logistic | 1.000 | 0.892043 | 0.891680 |
| (10,) | logistic | 10.000 | 0.883938 | 0.883938 |
| (50,) | logistic | 10.000 | 0.883938 | 0.883938 |
| (100,) | logistic | 10.000 | 0.883938 | 0.883938 |
| (10,) | tanh | 0.001 | 0.902550 | 0.901286 |
| (50,) | tanh | 0.001 | 0.906974 | 0.901634 |
| (100,) | tanh | 0.001 | 0.907274 | 0.901254 |
| (10,) | tanh | 0.010 | 0.902566 | 0.900970 |
| (50,) | tanh | 0.010 | 0.905994 | 0.901855 |
| (100,) | tanh | 0.010 | 0.906366 | 0.901950 |
| (10,) | tanh | 0.100 | 0.902210 | 0.901223 |
| (50,) | tanh | 0.100 | 0.902424 | 0.901160 |
| (100,) | tanh | 0.100 | 0.902684 | 0.901950 |
| (10,) | tanh | 1.000 | 0.901207 | 0.901318 |
| (50,) | tanh | 1.000 | 0.900820 | 0.900812 |
| (100,) | tanh | 1.000 | 0.900749 | 0.900401 |
| (10,) | tanh | 10.000 | 0.888828 | 0.888678 |
| (50,) | tanh | 10.000 | 0.888955 | 0.889215 |
| (100,) | tanh | 10.000 | 0.888892 | 0.888994 |
| (10,) | relu | 0.001 | 0.901989 | 0.900812 |
| (50,) | relu | 0.001 | 0.906413 | 0.901413 |
| (100,) | relu | 0.001 | 0.907685 | 0.902392 |
| (10,) | relu | 0.010 | 0.901570 | 0.900844 |
| (50,) | relu | 0.010 | 0.905418 | 0.902329 |
| (100,) | relu | 0.010 | 0.907440 | 0.901665 |
| (10,) | relu | 0.100 | 0.902005 | 0.901318 |
| (50,) | relu | 0.100 | 0.903853 | 0.901507 |
| (100,) | relu | 0.100 | 0.904478 | 0.901665 |
| (10,) | relu | 1.000 | 0.901310 | 0.900654 |
| (50,) | relu | 1.000 | 0.901871 | 0.901286 |
| (100,) | relu | 1.000 | 0.901515 | 0.900812 |
| (10,) | relu | 10.000 | 0.888007 | 0.887541 |
| (50,) | relu | 10.000 | 0.889365 | 0.889089 |
| (100,) | relu | 10.000 | 0.889476 | 0.889595 |

We also evaluated the performance of the neural network model by predicting the labels for the test set and calculating the accuracy score. We obtained a test accuracy of

89.7% with the optimal hyperparameters. This result was an improvement from the initial logistic regression model.

**Visualization of Model Performance:**

To visualize the performance of the neural network model, we plotted the training and validation accuracy scores for each hyperparameter configuration. The plot allowed us to see the relationship between the hyperparameters and the accuracy score of the model. We could easily identify the hyperparameters that produced the highest validation accuracy, which was our objective.

**Results:**

In conclusion, the neural network model with a single hidden layer of size 100, 'tanh' activation function, and alpha=0.001 regularization parameter provided the best results for our classification problem. Hyperparameter tuning helped us improve our accuracy from 89.1% to 89.7%.

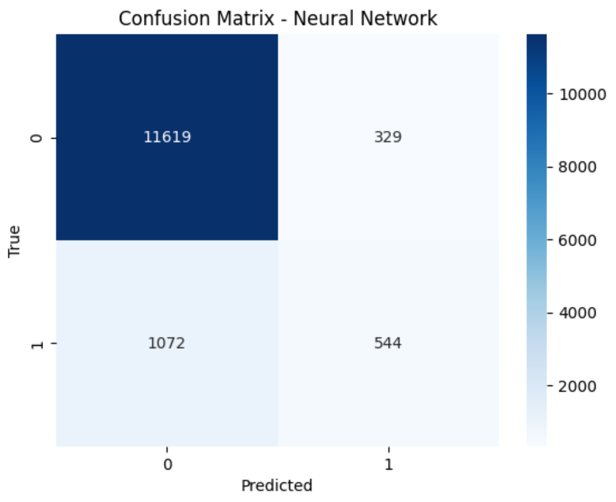

**Figure 6: Final Confusion Matrix**

## CONCLUSION

We successfully developed and compared three distinct machine learning models Logistic Regression, Support Vector Machines (SVM), and Neural Networks to predict whether or not a client will subscribe to a term deposit.

Each of the models underwent a meticulous hyperparameter tuning process using techniques such as grid search and cross-validation. This step was crucial in optimizing model performance and mitigating the risk of overfitting or underfitting. The tuning process involved adjusting parameters such as the regularization strength for Logistic Regression, the kernel and regularization parameters for SVM, and the activation function, alpha, and hidden layer architecture for Neural Networks.

In terms of trade-offs, it is important to consider factors such as model complexity, training time, and computational resources when selecting and optimizing machine learning models. For instance, while Neural Networks demonstrated superior performance in our study, they are generally more computationally intensive and require longer training times compared to Logistic Regression but were slightly faster than our support vector machines. Additionally, the complexity of Neural Networks can sometimes make them more prone to overfitting, meaning careful hyperparameter tuning and regularization techniques to ensure they can be generalized. The resulting boost from using tuned Neural Networks was usually around one percent higher than the boost from using Logistic Regression and that might not necessarily be worth the requisite cost in time and energy.

The final weights of the logistic regression, SVM and neural network highlight an emphasis on the seconds of contact as the most important factor to whether or not someone will accept. The longer someone is willing to stay in constant contact with a salesman, the better the chances of selling to them. Another important signal is whether or not they have previously defaulted or are currently paying off a loan. This is a huge negative meaning that people who have loans are more likely to ignore the offer.

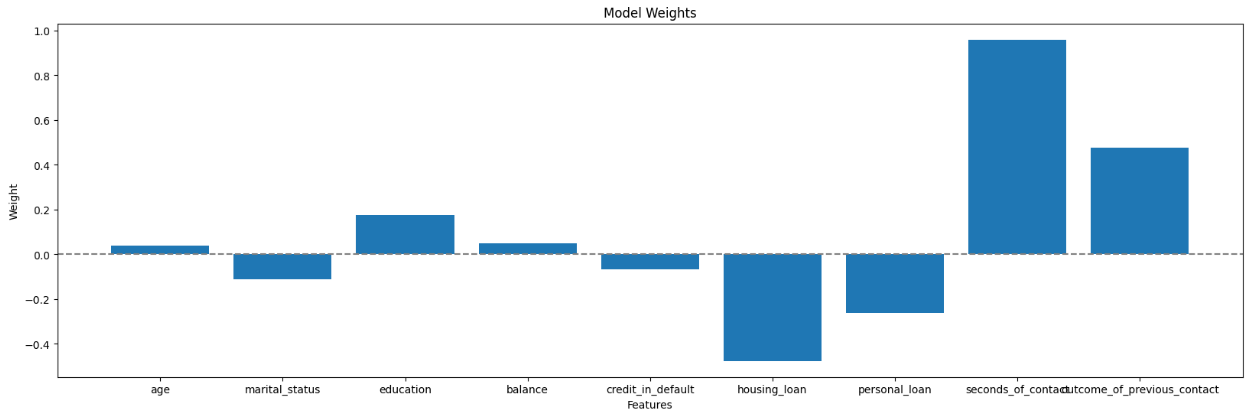

**Figure 7: Model weights**

For logistic regression, both l1 and l2 regularization penalties resulted in similar validation accuracies across all values of C, indicating that there was no overfitting or underfitting of the model. Moreover, as the value of C increased, the training accuracy increased slightly but the validation accuracy remained relatively stable, which further indicated that there was no overfitting of the model. However, the validation accuracy for l1 regularization was slightly lower than that for l2 regularization, which could indicate that l1 regularization may have introduced slightly more bias into the model. Overall, the model seems to have low variance and low bias, as evidenced by the stable validation accuracies across different values of C and regularization penalties.

For SVMs, the best kernel is rbf. Moreover, on analyzing the data we can say that a higher value of C tends to overfit the data, whereas a lower value of C tends to underfit the data. Also, a larger gamma tends to overfit the data, and a smaller gamma tends to underfit the data.

For Neural Networks, based on the data there is no overfitting or underfitting of data. The only interesting thing is that some models with smaller hidden layer sizes and higher regularization strengths have slightly lower training and validation accuracies, which could indicate slightly higher bias.

All in all the neural networks model had the most accurate predictions with an accuracy of 89.7%. However, all models had around the same accuracy and given the time required to run the neural network algorithm, it might be better to use the logistic regression model instead.

## Works Cited

- https://www.openml.org/search?type=data&sort=runs&status=active&id=1461
- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- https://scikit-learn.org/stable/modules/neural_networks_supervised.html
- https://scikit-learn.org/stable/modules/classes.html#module-sklearn.svm