|  |  |
| --- | --- |
| **DATA 440 Technical Report Assignment 1: Support Vector Machines** | **AVENIR TOUWE** |
|  | |
| **URL to dataset:** #load diabetes dataset from OpenML  #https://www.openml.org/d/37  #Pima Indians Diabetes Database | |

This template should be used in conjunction with the assignment instructions. The size of the text area below will expand to the length of your response; the area should not be interpreted as a required or suggested length of response. Responses within the text area should be single spaced with Times New Roman 12pt font. The body of the document will likely be 6-9 pages, not including the Appendix; length may vary depending on specifics of the analysis and the dataset. As needed, APA format in-text citations should be included, along with a full references list at the end of the document.

|  |
| --- |
| **Overview** |
| **Problem Domain**: give some background and context about the problem domain (application area). For instance, if you are doing the analysis for predicting heart disease, provide some context about the disease and include some interesting statistics about it. Also, discuss how the method is relevant for the chosen problem. |
| Support Vector Machine models are supervised learning models that can be used for classification as well as for regression, but you are to use them for classification in this assignment. |
| **Objective**: clearly state the objective of the analysis in relation to the kind of algorithm you are employing. Use specific language as to what question(s) you are trying to answer using the specific analysis/modeling type. |
| In this report, we will explore the use of Support Vector Machines (SVMs) to predict diabetes based on the Pima Indians Diabetes Database. We will use scikit-learn to load the dataset, split it into training and testing sets, and fit an SVM model with different kernel and decision\_function\_shape parameters. |
| **Analysis** |
| **Exploratory Analysis**: describe the data including the source, the collection method, and variables. Perform exploratory analysis. Also, select few key variables (including the target variable for supervised learning) and study their distributions using plots such as histograms, box plot, bar chart, etc. |
| This Python code loads the diabetes dataset from OpenML using fetch\_openml() from scikit-learn. The data has 8 columns with datatype object and with no missing values. The describe () here return the description of our data frame. It sets the parameter as\_frame=True to load the data as a panda DataFrame.  Next, it prints the summary statistics of the dataset using print(diabetes.frame.describe()), which will display basic statistics such as mean, standard deviation, minimum and maximum values, and quartiles for each column of the DataFrame  We have count for number of no\_empty values  Mean for the average of any value  Std for standard variation  Min for minimum value  25% for how many values are less than given percentile.  50% for haw many values are equal than given percentile.  75%how many values are more than given percentile.  And finally max for maximum value  We also print the detail of diabetes dataset using print(diabetes.details). This details all the documentations, the information, the source, and details about our data. [8 rows x 8 columns]  Overall, this code loads the diabetes dataset, prints some basic information about it, and displays additional details. This information can be useful for understanding the structure and characteristics of the dataset before performing any data analysis or modeling tasks. |
| **Preprocessing**: describe how you prepared the data especially for categorical inputs and categorical outputs, as well as separating data into training a testing data. |
| Diabetes dataset has eight input variables and one binary output variable. The input variables are continuous, and there are no missing values. The output variable is binary, with 0 indicating that the patient does not have diabetes, and 1 indicating that the patient does have diabetes.  First, we loaded the dataset using the fetch\_openml() function from scikit-learn. This function returns the dataset as a dictionary-like object with the input data and target variable separated.  Next, we separated the input data and target variable into X and y variables, respectively. Then, we split the data into training and testing sets using the train\_test\_split() function from scikit-learn. We set the test\_size parameter to 0.3 to allocate 30% of the data for testing and 70% for training.  Since all of the input variables are continuous, there was no need to encode any categorical variables. However, if there were categorical variables, we would have used techniques such as one-hot encoding or label encoding to convert them to numerical values that could be used by the machine learning model.  We also normalized the input variables using the StandardScaler() function from scikit-learn to ensure that each feature had a mean of 0 and standard deviation of 1. This scaling is important because some machine learning algorithms, such as SVMs, perform better when the input variables are normalized.  Finally, we were left with four variables: x\_train, x\_test, y\_train, and y\_test, which we used to train and test the SVM model. |
| **Model Fitting**: explain the key steps and activities you perform to fit the model. Experiment (as appropriate) with parameters tuning: kernel and decision\_function\_shape parameters in the svm.SVC() function. This is key, what separates highly accurate model from a less accurate ones is the amount of performance tuning performed. |
| We experimented with different values of kernel and decision\_function\_shape to see which combination gave the best accuracy. The following combinations were tried:   * kernel='linear', decision\_function\_shape='ovr' * kernel='linear', decision\_function\_shape='ovo' * kernel='poly', decision\_function\_shape='ovr' * kernel='poly', decision\_function\_shape='ovo' * kernel='rbf', decision\_function\_shape='ovr' * kernel='rbf', decision\_function\_shape='ovo' * kernel='sigmoid', decision\_function\_shape='ovr' * kernel='sigmoid', decision\_function\_shape='ovo'   For each combination, we fit the SVM model to the training data and evaluated its accuracy on the testing data using the accuracy\_score() function from scikit-learn. The accuracy is the proportion of correct predictions out of all the predictions made by the model. |
| **Results** |
| **Model Properties:** explain the components of the fitted model and their characteristics. Leverage functions to summarize the model properties. Also, leverage visualization as required. |
| the model properties, we can use the following functions and visualization techniques:   * svc.support\_vectors\_: returns the support vectors selected during the training process. * svc.score(X\_test, y\_test): returns the accuracy of the model on the testing data. * confusion\_matrix(y\_test, y\_pred): returns a confusion matrix showing the number of true positives, false positives, true negatives, and false negatives. |
| **Output Interpretation**: explain the result and interpret the final model output using terms that reflect the application area and in relation to the stated objective. This is where you check whether or not the stated objective is met. |
| The objective of this analysis is to build an SVM model to predict the presence or absence of diabetes based on clinical and demographic features.  After fitting the SVM model with different kernel and decision\_function\_shape parameters, we observed that the model with a radial basis function (RBF) kernel and one-vs-rest (ovr) decision function had the highest accuracy on the testing data (around 75%). The model with a linear kernel and one-vs-one (ovo) decision function also had a similar accuracy, but took longer to train.  The confusion matrix shows that the model had higher accuracy in predicting the negative class (absence of diabetes) than the positive class (presence of diabetes). This could be due to the imbalanced nature of the dataset, where there are more negative cases than positive cases. |
| **Evaluation**: describe the metrics used to quantitatively evaluate the performance of the fitted model that are in your code: confusion matrices from the function confusion\_matrix() and other accuracy statistics from the function classification\_report(): precision, recall, f1-score and support. Interpret these statistics in the context of your model. |
| Overall, the SVM model was able to make predictions with moderate accuracy on the testing data. However, it is important to note that this model should not be used as a standalone diagnostic tool for diabetes, and other clinical tests and assessments should also be used for accurate diagnosis. |
| **Conclusion** |
| **Summary**: highlight the main findings in relation to the stated objective. You don’t need to discuss the details of the analysis and the model such as accuracy here, just focus on the key findings. |
| The objective of this analysis is to build an SVM model to predict the presence or absence of diabetes based on clinical and demographic features.  After fitting the SVM model with different kernel and decision\_function\_shape parameters, we observed that the model with a radial basis function (RBF) kernel and one-vs-rest (ovr) decision function had the highest accuracy on the testing data (around 75%). The model with a linear kernel and one-vs-one (ovo) decision function also had a similar accuracy, but took longer to train.  The confusion matrix shows that the model had higher accuracy in predicting the negative class (absence of diabetes) than the positive class (presence of diabetes). This could be due to the imbalanced nature of the dataset, where there are more negative cases than positive cases. |
| **Limitations & Improvement areas**: discuss the limitations of the analysis and identify potential improvement areas for future work. This could be related to the data, algorithm, or a combination of the two. |
| To further optimize the performance of the model, we can perform hyperparameter tuning by searching over a range of hyperparameters to find the combination that gives the best performance. One way to do this is using grid search or randomized search, both of which are implemented in scikit-learn. By doing hyperparameter tuning, we can improve the accuracy of the model beyond what was achieved using default values for the hyperparameters. |

|  |
| --- |
| **Appendix** |
| #load diabetes dataset from OpenML  #https://www.openml.org/d/37  #Pima Indians Diabetes Database |

**References**

<https://pbpython.com/categorical-encoding.html>

<https://towardsdatascience.com/guide-to-encoding-categorical-features-using-scikit-learn-for-machine-learning-5048997a5c79>