**CPSC6114: Fundamentals of Machine Learning**

**Assignment 3: Support Vector Machine Algorithm (SVM) Algorithm**

Model

The goal of this assignment is to build a **Support Vector Machine** algorithm to classify customers into class 1 who are likely to default on a loan borrowed from a bank, and class 0 who are not likely. This will be done based on the observations provided in **original.csv** file.

**SVM** is another classification algorithm that can be used to solve BOTH linear and non-linear data. Its goal is to separate the data using hyperplane with highest margin (distance between support vectors and hyperplane) By allowing to introduce **additional dimensions** (using kernel trick) the algorithm converts non-linear data set into a dataset that can be linearly separated. SVM can be computationally intensive and does not work well with very large sets. It does not handle noise and outliers well. SVM also does not work well in cases where the number of features for each data point exceeds the number of training data samples. However, SVM can be very effective when working with high number of features/dimensions.

The model effectiveness is measured by scores obtained from confusion matrix, precision, recall, F1 score and accuracy.

Dataset Analysis

We start by importing all the relevant libraries into Python, including sklearn, numpy, pandas, and os. We then import the data from **original.csv** into a data frame.

*dataset = pd.read\_csv('original.csv')*

We start by reviewing the data in the dataset. There are two issues with this dataset:

1. There are **three values in Age field** that contain null instead of data:Graphical user interface, text

   Description automatically generated

Since this represents a very small percent of overall values (3 out of 2000), we remove these observations instead of trying to repair them.

*dataset=dataset[dataset['age'].notna()]*

1. **Data** in original.csv file **appears to be unbalanced**: only 282 observations in category 1, with 1717 observations in category 0.

Graphical user interface, text, application

Description automatically generated

We can attempt to solve this problem later on by using under sampling technique and create a new dataset which would keep all 283 observations in category 1, and randomly choosing 283 observations from category 0.

Model Building

Once the file is loaded into a dataframe, we need to define our X and Y (target) values and normalize the data.

*X = dataset[dataset.columns[1:4]]*

*Y = dataset.iloc[:,-1]*

In general when building an SVN model, **data normalization** is necessary to make sure that the scale of all variables are comparable. For data normalization we use **MinMax** approach to convert the original range of values into a new range between 0 and 1. However, scaling must be done on a set of well balanced data. In case where data is unbalanced, data scaling will not be as effective.

For data normalization we use **MinMax** approach to convert the original range of values into a new range between 0 and 1.

*X = (X - np.min(X))/(np.max(X) - np.min(X))*

Once normalized we can split the data into training (**75%**) and testing (**25%**).

*X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.25,random\_state=1)*

We are now ready to choose which the most **appropriate kernel** for our dataset. When plotting the data, it does not appear to be linearly separable. Therefore **linear kernel** is probably not the best one in this case.

Chart, scatter chart

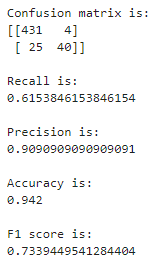
Description automatically generated

Still, we can try to run linear kernel to see what model accuracy we achieve:

*clf=svm.SVC(kernel = 'linear') #Linear kernel*

*clf.fit(X\_train, Y\_train)*

*y\_pred = clf.predict(X\_test)*



We can now try to run **rbf kernel** which works better with non-linear separable data:

*clf=svm.SVC(kernel = 'rbf') #rbf kernel*

*clf.fit(X\_train, Y\_train)*

*y\_pred = clf.predict(X\_test)*

Text

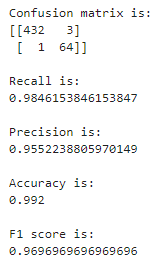
Description automatically generated

Even before calibrating gamma and C values, **Rbf kernel** produced better Recall, Accuracy, and much higher F1 score than linear kernel. Therefore we w**ill use rbf kernel for this assignment**

**Rbf kernal** can be further calibrated by choosing different values for gamma and C values:

* The **gamma parameter** defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’. The lower values of gamma result in models with lower accuracy and the same as the higher values of gamma. It is the intermediate values of gamma which gives a model with good decision boundaries. (https://vitalflux.com/svm-rbf-kernel-parameters-code-sample/)
* **C parameter** is a regularization parameter used to set the tolerance of the model to allow the misclassification of data points in order to achieve lower generalization error. Higher the value of C, lesser is the tolerance and what is trained is a maximum-margin classifier. Smaller the value of C, larger is the tolerance of misclassification and what gets trained is a soft-margin classifier that generalizes better than maximum-margin classifier. The C value controls the penalty of misclassification. A large value of C would result in a higher penalty for misclassification and a smaller value of C will result in a smaller penalty of misclassification. With a larger value of C, a smaller margin will be accepted if the decision function is better at classifying all training points correctly. The model may overfit with the training dataset. A lower C will encourage a larger margin, therefore a simpler decision function, at the cost of training accuracy (https://vitalflux.com/svm-rbf-kernel-parameters-code-sample/)

After trial and error, it seems that the model performance most optimally when **gamma = 65**. Both the values below this number and the values above it result in lower results across all parameters .



At the same time it seems that **C = 1** is the optimal parameter as well. C below or higher than 1 results in lower results

*clf=svm.SVC(kernel = 'rbf', gamma = 65, C = 1) #rbf kernel*

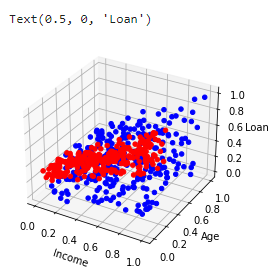
Results on a new dataset with with Unbalanced data issue resolved:

As discussed above, data in original.csv file appeared to be **unbalanced** with only 283 elements in category 1 out of 2000 total observations. We solve this problem by using **under sampling technique**, whereby we keep 283 observations in category 1 and randomly pick 282 observations in category 0. The resulting dataset looks is as follows:

Graphical user interface, text

Description automatically generated

Interestingly, now the data appears more linearly separable. We will try to re-run the SVM model with **linear kernel first**, and then try it with **rbf kernel.**



The new data set has increase in **Recall to 1** from 0.98 in the original file.

SVM with **linear kernel** produced the following results:

Text, letter

Description automatically generated

SVM with **rbf kernel (gamma = 65, C = 1)**, however, produced better results in most categories

Text, letter

Description automatically generated

10-fold cross validation

K-fold validation is a technique that is widely used to assess the skill of machine learning models. The idea is to split unseen data sample into k groups (in our case 10 groups). K-fold validation technique tend to be less biased/optimistic estimates vs simple train/test split.

The steps of the technique are:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each unique group:
   1. Take the group as a hold out or test data set
   2. Take the remaining groups as a training data set
   3. Fit a model on the training set and evaluate it on the test set
   4. Retain the evaluation score and discard the model
4. Summarize the skill of the model using the sample of model evaluation scores

Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model k-1 times.

(<https://machinelearningmastery.com/k-fold-cross-validation/>)

It been experimentally proved that k = 10 is a good default value to use that produces low bias and model variance

In Python, we can use **sklearn’s model\_selection** library and specifically **cross\_val\_score** to perform k-fold cross validation and specificy K to be 10 to perform 10-fold cross validation.

*from sklearn.model\_selection import cross\_val\_score*

*cv\_scores = cross\_val\_score(clf, X, Y, cv=10)*

*print("Cross Validation Scores:")*

*print (cv\_scores)*

*print("")*

*print("Average 10-Fold CV Score: {}".format(np.mean(cv\_scores)))*

When running 10-fold validation on **SVM model with linear kernel** we get the following results:

Text

Description automatically generated

When running 10-fold validation on **SVM model with RBF kernel** we get the following results:

Text, letter

Description automatically generated

This shows that **RBF kernel performs better** (0.98 average score) vs linear kernel (0.93 average score)

Unlike with regular train/test split (75/25) that we used initially, k-fold validation technique allows all parts of the data set to be used as in-sample and out—sample data.

Conclusion

It appears that RBF kernel works best for this dataset vs linear kernel. It also appears that **gamma = 65 and C = 1 produce** the best results for this model. The original dataset seems to be unbalanced, with only **14%** of elements representing category 1. Upon fixing the issue using under sampling and applying RBF kernel, we achieve **better results vs the original dataset**. 10-Cross validation also confirm this model choice.

**Full Python Code:**

*#Import necessary library*

*import os*

*import numpy as np*

*import pandas as pd*

*from sklearn import svm*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.metrics import confusion\_matrix, accuracy\_score, precision\_score, recall\_score, f1\_score*

*#Import data*

*pd.options.mode.use\_inf\_as\_na=True*

*os.chdir('C:/Users/denis/OneDrive/Desktop/CPSC6114\_ML')*

*dataset = pd.read\_csv('original.csv')*

*dataset=dataset[dataset['age'].notna()]*

*#Choose and normalize data for our analysis*

*X = dataset[dataset.columns[1:4]]*

*Y = dataset.iloc[:,-1]*

*#Applying MinMax Scaler formula*

*X = (X - np.min(X))/(np.max(X) - np.min(X))*

*#Split data into train and test as per requirement*

*X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.25,random\_state=1)*

*#Model*

*clf=svm.SVC(kernel = 'rbf', gamma = 65, C = 1) #rbf kernel*

*clf.fit(X\_train, Y\_train)*

*y\_pred = clf.predict(X\_test)*

*#Model Performance*

*ConfusionMatrix = confusion\_matrix(Y\_test,y\_pred)*

*Accuracy = accuracy\_score(Y\_test,y\_pred)*

*Precision = precision\_score(Y\_test,y\_pred)*

*Recall = recall\_score(Y\_test,y\_pred)*

*F1Score = f1\_score(Y\_test,y\_pred)*

*print("Confusion matrix is:")*

*print(ConfusionMatrix)*

*print("")*

*print("Recall is:")*

*print(Recall)*

*print("")*

*print("Precision is:")*

*print(Precision)*

*print("")*

*print("Accuracy is:")*

*print(Accuracy)*

*print("")*

*print("F1 score is:")*

*print(F1Score)*

*#10-FOLD CROSS VALIDATION*

*cv\_scores = cross\_val\_score(clf, X, Y, cv=10)*

*print("Cross Validation Scores:")*

*print (cv\_scores)*

*print("")*

*print("Average 10-Fold CV Score: {}".format(np.mean(cv\_scores)))*