FRANKFURT UNIVERSIY OF APPLIED SCIENCES, MARCH 2019, AI/ML SOFTWARE ENGINEERING SUPERVISED BY DAMIR DOBRIC

Data classification with Support Vector Machine

Thanh Tu Pham

Abstract — Support Vector Machine, which is known as one of the best ”out of the box” supervised classification techniques. SVM has a solid theoretical background, an intuitive geometrical interpretation, and several interesting properties that link the development of kernel space and convex optimization. At the early age of SVM, it suffered from the computational complexity, coming from a large-scale quadratic programming problem, and therefore seemed to be an ideal tool. In this project, I implemented Support Vector Machine (SVM) from scratch to solve linear binary classifier problem, using Simplified Sequential Minimal Optimization (SMO) algorithm for training data. After giving labeled training data input, the algorithm will be used to train and the output data is divided into two classes by a separating hyperplane.

I. INTRODUCTION

Support Vector Machine, which is known as one of the best “out of the box” supervised classification techniques. After giving labeled training data input, the algorithm will be used to train and the output data is divided into two classes by a separating hyperplane. The goal of this project is to solve a binary classification problem with linear classifier using Support Vector Machine technique. Specifically, I implemented Support Vector Machine (SVM) from scratch using Simplified Sequential Minimal Optimization (SMO) algorithm for training data. After training, the program can predict the class of unlabeled data. The project is divided into four main parts: Introduction, Method, Implementation and Result, and Discussion. Introduction section explains the purpose of the report. Next is the most important part, the Method section. In this part, the Support Vector Machine technique is explained briefly and the Simplified SMO algorithm will be explained in details. After that, Implementation and Result section shows step-by-step for applying the algorithm for training data in programming. The result of the object classification is also shown in this part. Last but not least, Discussion section summarizes the main points of the project and the limited points of the program which should be improved in the near future.

II. METHOD

A. Support Vector Machine

Support Vector Machine, so called as SVM, is a supervised learning algorithm which can be used for classification and regression problems as support vector classification (SVC) and support vector regression (SVR). The Support Vector Machine models the situation by creating a feature space, which is a finite-dimensional vector space, each dimension of which represent a “feature” of a particular object. On other words, SVMs tries to find a separating line (or hyperplane) between data of two classes. After getting the data input, SVMs algorithm is used to train and outputs a line that separates those classes if possible. [4]

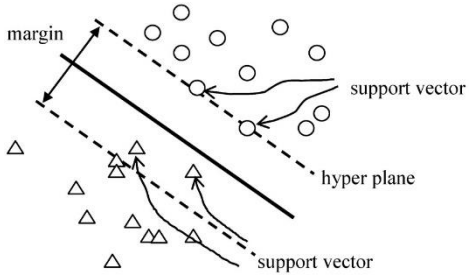

Figure 1: Support Vector Machine

According to the SVMs algorithm, firstly the points, which are closest to the line from both the classes, need to be found. These points are called support vectors. Next step, the distance between the line and the support vectors are calculated, so called the margin. The purpose of SVMs is getting maximum the margin. The hyperplane for which the margin is maximum is the optimal hyperplane. However, when the data is quite complex such as not linearly separable, it is had to choose the line to separate two classes. Therefore, there are some parameters which can be tuned to make the better result. These are called regularization parameter and gamma.

B. Regularization parameter

The Regularization parameter (so called C) explains the SVMs optimization how much you want to avoid misclassifying each training example. When the value of C increases, the width of

margin decreases, the training points will be classified correctly easier. Conversely, the width of margin increases when the value of C decreases, so maybe there are many points of training data having wrong classifier.

C. Gamma

The gamma parameter shows how far the influence of a single training example reaches, with low values meaning "far" and high value meaning "close". In other words, with low gamma, points far away from plausible separation line are considered in calculation for the separation line. Whereas high gamma means the points close to plausible line are considered in calculation. Two figures below show the data set of linear separable data set and no linear separable data set.

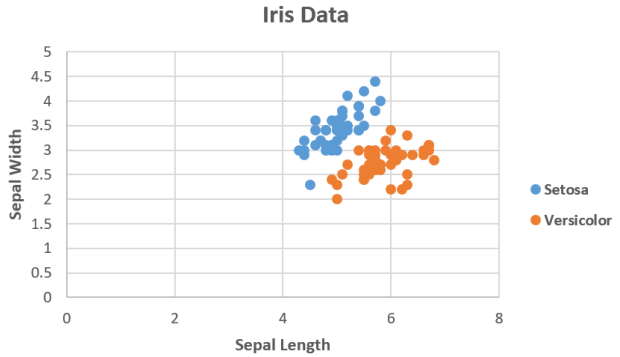

Figure 2: Linearly Separable Support Vector Machine

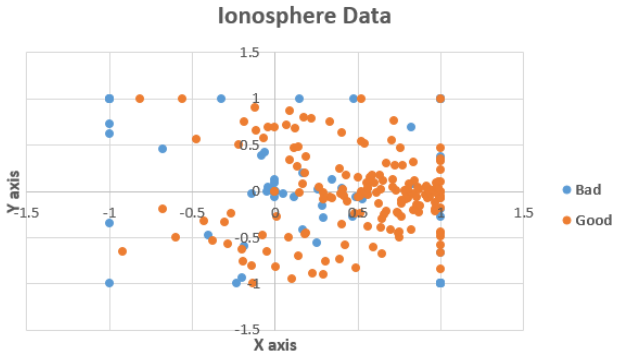

Figure 3: Non-linearly Separable Support Vector Machine

D. Simplified Sequential Minimal Optimization [5]

After getting value of regularization parameter (C) and gamma (γ), we need to find the "perfect" line to classify two classes. In this project, Support Vector Machine uses Simplified Sequential Optimization algorithm for training data [3]. A support vector machine computes a linear classifier of the form

$$f(x) = w^T x + b \tag{1}$$

As considering a linear classifier for a binary classification problem with labels y($y \in [-1,1]$), we will ultimately predict $y = 1$ if $f(x) \geq 0$ and $y = -1$ if $f(x) < 0$, but for now we simply consider the function $f(x)$, which can be expressed as

$$f(x) = \sum_{i=1}^{m} \alpha_i y^{(i)} < x^{(i)}, x > + b \tag{2}$$

Where we can substitute a kernel $K(x^{(i)}, x)$ in place of the inner product. In this project, we are applying the linear kernel, so the dot product will be used. The SMO algorithm gives an efficient way of solving the dual problem of the support vector machine by solving

$$W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j < x^{(i)} x^{(j)} >$$
$$subject\ to\ 0 \leq \alpha_i \leq C, i = 1, \dots, m \tag{3}$$
$$\sum_{i=1}^{m} \alpha_i\, y^{(i)} = 0$$

The Karush-Kuhn-Tucker (KKT) conditions can be used to check for convergence to the optimal point. For this problem the KKT conditions are

$$\begin{aligned} \alpha_i = 0 \quad &\rightarrow \quad y^{(i)} w^T x^{(i)} + b \geq 1 \\ \alpha_i = C \quad &\rightarrow \quad y^{(i)} w^T x^{(i)} + b \leq 1 \\ 0 < \alpha_i < 0 \quad &\rightarrow \quad y^{(i)} w^T x^{(i)} + b = 1 \end{aligned} \tag{4}$$

The SMO algorithm iterates until all these conditions are satisfied (to within a certain tolerance) thereby ensuring convergence. Specifically, the SMO algorithm select two α parameters, α_i and α_j and optimizes the objective value jointly for both these α's. Finally it adjusts the b parameter based on the new α's. This process is repeated until the α's converge.

E. Selecting α Parameters

Firstly, we simply iterate over all $\alpha_i, i = 1, \dots, m$. If α_i does not fulfill the KKT conditions to within some numerical tolerance, we select α_j at random from the remaining $m - 1$ α's and attempt to jointly optimize α_i and α_j. If none of the α's are changed after a few iteration over all the α_i 's, then the algorithm terminates.

F. Optimizing α_i and α_j

In order to optimize the Lagrange multipliers α_i and α_j, we must find the bounds L and H to satisfy the constraints.

- If $y^{(i)} \neq y^{(j)}$

$$L = \max(0, \alpha_j - \alpha_i), \quad H = \min(C, C + \alpha_j - \alpha_i) \tag{5}$$

Now we want to find α_j so as to maximize the objective function. If this value ends up lying outside the bounds L and H, we simply clip the value of α_j to lie within this range below:

$$\alpha_j := \begin{cases} H & if\ \alpha_j > H \\ \alpha_j & if\ L \leq \alpha_j \leq H \\ L & if\ \alpha_j < L \end{cases} \tag{6}$$

The value of α_j can be optimized by:

$$\alpha_j := \alpha_j - \frac{y^{(i)}(E_i - E_j)}{\eta} \quad (7)$$

where

$$E_k = f(x^{(k)}) - y^{(k)}$$

$$\eta = 2 < x^{(i)}, x^{(j)} > - < x^{(i)}, x^{(i)} > - < x^{(j)}, x^{(j)} > \quad (8)$$

E_k is the error between the SVMs output on the kth example and the true label $y^{(k)}$. Finally, the value of α_i can be calculated after getting the value of α_j

$$\alpha_i := \alpha_i + y^{(i)} y^{(j)} (\alpha_j{}^{(old)} - \alpha_j) \quad (9)$$

where $\alpha_j{}^{(old)}$ is the value of α_j before optimization.

1) Computing the b threshold

After optimizing α_i and α_j, we select the threshold b such that the KKT conditions are satisfied for the ith and jth examples. If, after optimization, α_i is not at the bound (i.e. $0 < \alpha_i < C$), then the following threshold b_1 is valid, since it forces the SVM to output $y^{(i)}$ when the input is $x^{(i)}$

$$b_1 = b - E_i - y^{(i)}(\alpha_i - \alpha_i{}^{old}) < x^{(i)}, x^{(i)} > - y^{(j)}(\alpha_j - \alpha_j{}^{old}) < x^{(i)}, x^{(j)} > \quad (10)$$

Similarity, the following threshold b_2 is valid if $0 < \alpha_j < C$

$$b_2 = b - E_j - y^{(i)}(\alpha_i - \alpha_i{}^{old}) < x^{(i)}, x^{(j)} > - y^{(j)}(\alpha_j - \alpha_j{}^{old}) < x^{(j)}, x^{(j)} > \quad (11)$$

If both $0 < \alpha_i < C$ and $0 < \alpha_j < C$ then both these threshold are valid, and they will be equal. If both new α's are at the bound (i.e. $\alpha_i = 0$ or $\alpha_i = C$ and $\alpha_j = 0$ or $\alpha_j = C$) then all the threshold between b_1 and b_2 satisfy the KKT conditions, we let $b := \frac{b_1 + b_2}{2}$.

$$b := \begin{cases} b_1 & if\ 0 < \alpha_i < C \\ b_2 & if\ 0 \le \alpha_j \le C \\ \dfrac{b_1 + b_2}{2} & otherwise \end{cases} \quad (12)$$

2) Computing the w weight

After optimizing α_i and α_j, we can also compute w with the below function

$$w = \sum_{i=1}^{m} y^{(i)} \alpha_i x_i \quad (13)$$

III. IMPLEMENTATION AND RESULT

A. Architecture

In this part, the architecture of the solution will be shown in block diagram, including *Data analyzing and processing*, *Training*, and *Testing*. Firstly, the data needs to be checked its existence and correct form before applying for training step. Next, the data is trained with Simplified Sequential Minimal Optimization (SMO) algorithm. Last step is testing data, which predict class of unlabeled data by using the result of training process. With each step, another block diagram also will be presented to explain in details.

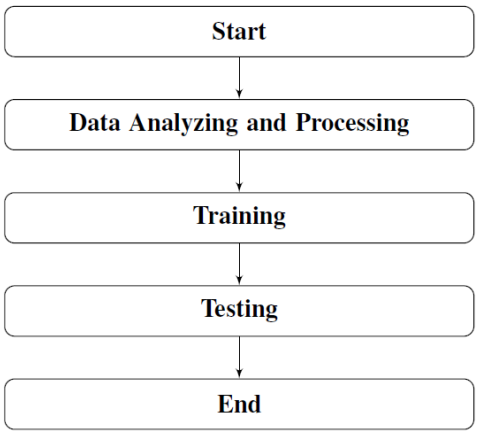

Figure 4: The main architecture of the project

1) Data Analyzing and Processing

Firstly, the data needs to be checked before using for training step through the method VerifyRawDataConsistency. This method will check whether all the samples have same given number of attributes. Furthermore, the error also captured when the data is empty or non-existent. At the beginning, the data is defined as jagged array. However, in order to apply for training process, the data must be converted to 2D array. Therefore, converting data from jagged array to 2D array is done in next step through the method ConvertRawDataToArray. The last step in preparing data is to separate data to input attributes and output label with the methods RawDataInput and RawDataOutput.

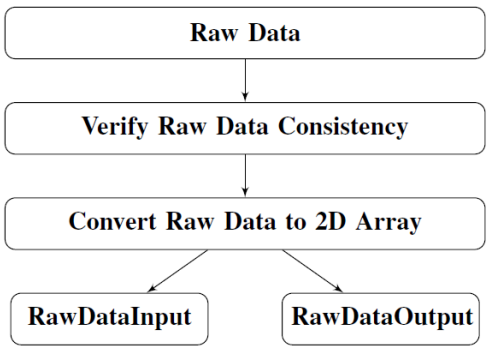

Figure 5: The flowchart of Data Analyzing and Processing step

2) Training

After getting data for input and output, two more parameters need to be defined before training with Simplified SMO algorithm, which are Regularization parameter (C) and tolerance numerical (tol). These two parameters are chosen by user, because they depends on the data. tol is used to test for convergence of SMO algorithm, and is typically set to around

0.001 to 0.01. In this project, the default value of tol is 0.01, but can be changed by ser. C is used to show how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small of C will cause the optimization to look for a larger margin separating hyperplane, even if that hyperplane misclassifies more points.

The data is trained by using Simplified SMO algorithm through the method *Run*. The goal of this algorithm is to find the value of weight and bias, then apply to equation (1) to classify the data. The detail of SMO algorithm is already explained in the above part and can follow easier by flowchart below.

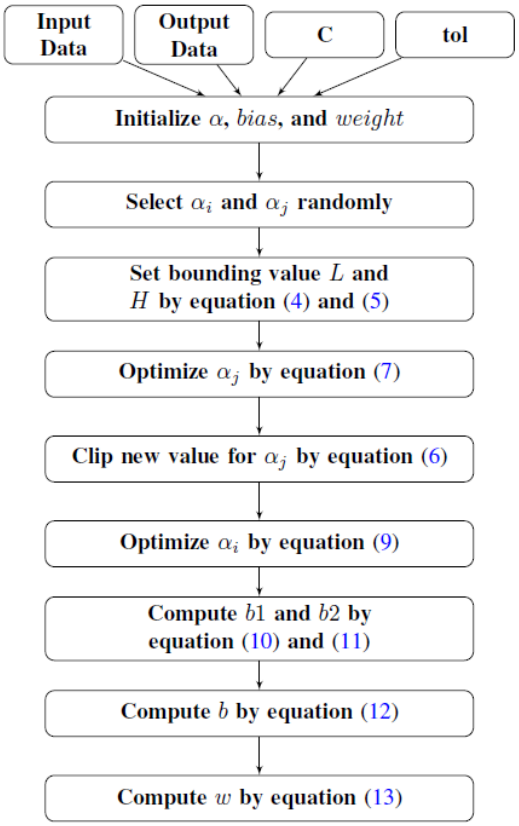

Figure 6: The flowchart of Training step

3) Testing

After training data step, we get values of weight and bias. Next, each data in testing data will be classified by applying the equation (1) through the method *Predict*. The flowchart of testing step is described below.

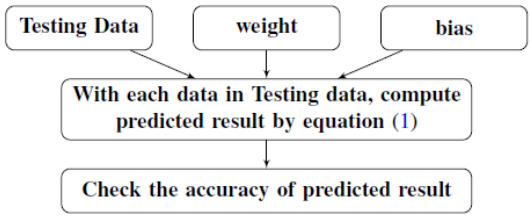

Figure 7: The flowchart of testing step

B. Programming

In this part, coding part including classes and their contents will be explained.

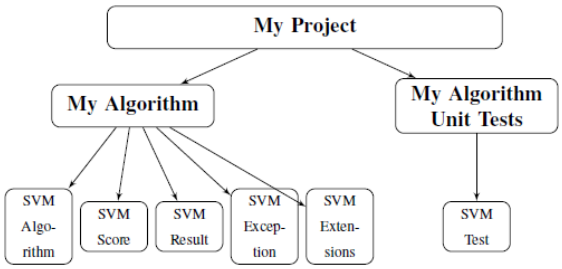

Figure 8: The flowchart of the program

The program is divided into two main projects: *MyAlgorithm* and *MyAlgorithmUnitTests*. Most of the work are done in the *MyAlgorithm* project, including analyzing input and output data, training data function and predicting data function. Whilst in *MyAlgorithmUnitTests* project, it includes testing sample data and testing function. The whole program is run in C#.NET and uses the NuGet package is provided from [2]

1) MyAlgorithm Project

This project includes five classes: SVMAlgorithm.cs, SVMExtensions.cs, SVMException.cs, SVMResult.cs, and SVMScore.cs

- **SVMAlgorithm**: SVMAlgorithm class is the most important class in this project. It includes every methods which need to analyze data, training and testing data. Some main methods are shown below:
  - **public SVMAlgorithm (double C, double tol=0.01)**
    The constructor of this class. It is used to set value of regularization parameter (C) and numerical tolerance (tol), which need to for training process.
  - **public IScore Run(double[][] rawData, IContext ctx)**
    This method is used to train data. The raw data input has the features and its output label, which is given in .csv file. After training by Simplified SMO algorithm, the result will be saved in class SVMScore and used for testing process later.

- **SVMScore**: SVMScore class contains the response object of the SVM training function.
  Parameters: double *Bias*, double[][] *Alphas*, double[] *Weight*, string *Message*.
- **SVMResult**: This class contains the response object of the SVM's predict function
  Parameter: int[] *PredictedResult*
- **SVMException**: This class is used to throw the encountered errors. These error messages are preceded by the function name where the error was encountered.
  Parameters: int *Code*, string *ErrorMessage*.
- **SVMExtensions**: This class is used to call SVM Algorithm through Learning API NuGet.

2) *MyAlgorithmUnitTests*

This project includes testing sample data in .csv file and class SVMTest for testing the performance of the Support Vector Machine. The flowchart of SVMTest class is described below.

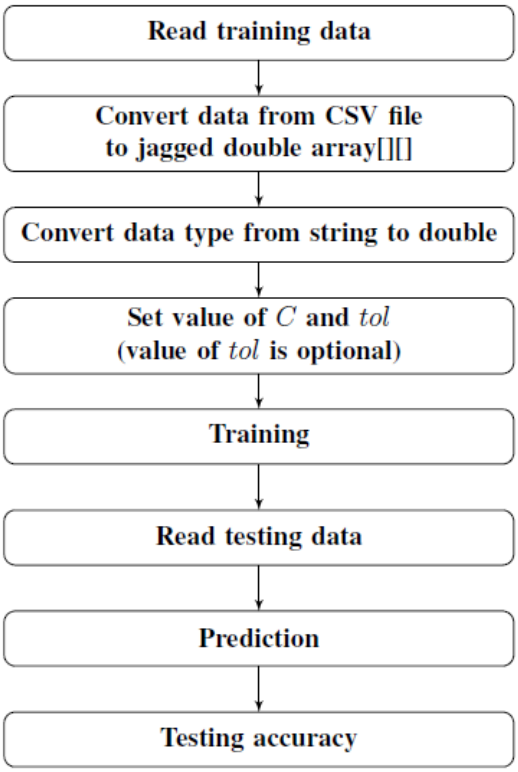

Figure 9: The flowchart of SVMTest class

C. Testing data and result

In class SVMTest, it includes four Test Classes methods (Example, SVMTest_IrisData, SVMTest_BreastCancerData, and SVMTest_IonosphereData). The concept of all four are the same, except the data for training and testing. Therefore, Example test class will be described in details to help readers can understand easily how the program working. The rest test classes can be understood by following Example.

1) *Example Test* [1]

The description of Example Unit Test in figure 10 is shown below.

```
[TestClass]
public class Example
{
    private const string IrisTrainingData = @"Samples\IrisData\iris.csv";
    /// <summary>
    /// The dataset is IRIS, including 100 data samples, which is divided into 2 parts:
    /// Training of the model and Testing of the model.
    /// The training data has 90 data samples saved in Samples\IrisData\iris.csv file.
    /// The testing data set has 10 data samples and it is loaded firectly in the code.
    /// </summary>
    [TestMethod]
    public void SmallIrisDataSetTest()
    {
        var path = System.IO.Path.Combine(Directory.GetCurrentDirectory(), IrisTrainingData);

        LearningApi api = new LearningApi(Helpers.GetDescriptor());

        // Convert data in csv file into double[][]
        api.UseCsvDataProvider(path, ',', true, 0);

        // Convert data from string to double according to the context
        api.UseActionModule<object[][], double[][]>((object[][] data, IContext ctx) =>...);

        //Set the regularization parameter and the tolerance value
        api.UseSVMAlgorithm(0.5);

        //Training
        SVMScore score = api.Run() as SVMScore;

        //Testing data preparation
        double[][] predictingData = new double[10][];

        predictingData[0] = new double[] { 4.6, 3.2, 1.4, 0.2 };
        predictingData[1] = new double[] { 5.3, 3.7, 1.5, 0.2 };
        predictingData[2] = new double[] { 5.0, 3.3, 1.4, 0.2 };
        predictingData[3] = new double[] { 7.0, 3.2, 4.7, 1.4 };
        predictingData[4] = new double[] { 6.4, 3.2, 4.5, 1.5 };
        predictingData[5] = new double[] { 4.4, 2.9, 1.4, 0.2 };
        predictingData[6] = new double[] { 5.7, 3.8, 1.7, 0.3 };
        predictingData[7] = new double[] { 6.7, 3.0, 5.0, 1.7 };
        predictingData[8] = new double[] { 6.3, 2.3, 4.4, 1.3 };
        predictingData[9] = new double[] { 5.7, 2.8, 4.1, 1.3 };
```

Figure 10: Example Test

a) **Read training data**: The modified Iris flower data set is used as training data. It has 100 samples (90 samples for training and 10 samples for testing) including four features (sepal_length, sepal_width, petal_length, and petal_width) and one label (species). The label is divided into two classes: *setosa* and *versicolor*. Normally, we take 80-90% sample data for training and the rest for testing.

b) **Convert data from CSV file to jagged double array[][]**: By using class *UseCsvDataProvider* from NuGet CsvDataProvider, the data is loaded from csv file into double array[][] with some options (file path, delimiter symbol, skip rows).

c) **Convert data type from string to double**: After loading data, data type is defined as string. It needs to convert to double in order to do calculation in training step by using class *UseActionModule*, which is provided already in Learning API.

d) **Set value of C and tol**: At this step, depending on the data, the user will fill in the value of regularization parameter (C) and tolerance numerical (tol). In this example, the value of C is set to 0.5, and the value of tol is set to default value (0.01). If tol equals default value, it does not need to be set.

e) **Training**: At this step, the data, and the value of C and tol are using as input for training step. After training, the value of weight and bias are saved at SVMScore class and will be used in predicting step.

f) **Reading testing data**: Testing data can be loaded by hard-code or in csv file like training data. In this example, ten data is added as hard-code (Figure 10).

g) **Prediction**: After loading testing data, the prediction part is shown in figure 11. In order to predict the data it uses command *SVMResult result = api.Algorithm.Predict() as SVMResult.* After predicting, the result is saved in SVMResult class.

h) **Testing accuracy**: In this part, we can compare the predicted result with the actual result. In order to do that, firstly we need to write the actual value into an array. Then, each predicted result is compared to each actual result by for loop and the accuracy is calculated at last. When the test is finished, the accuracy and predicted result are shown in output window (Figure 12). In this example, the accuracy of prediction is 100%.

```
//Predicting
SVMResult result = api.Algorithm.Predict(predictingData, api.Context) as SVMResult;

//Actual value is used to compare with predicted value
int[] actualValue = new int[] { 1, 1, 1, -1, -1, 1, 1, -1, -1, -1 };

//count the number correct answer of prediction
double checkAccuracy = 0;

for (int i = 0; i < predictingData.GetLength(0); i++)
{
    if (result.PredictedResult[i] == actualValue[i])
    {
        checkAccuracy++;
    }
    else
    {
        continue;
    }
}
// find the accuracy in percentage
double accuracy = checkAccuracy / predictingData.GetLength(0) * 100;

// show result in output
Console.WriteLine("The accuracy is {0} %", accuracy);
foreach (var item in result.PredictedResult)
{
    Console.WriteLine(item);
}
```

Figure 11: Example Prediction part

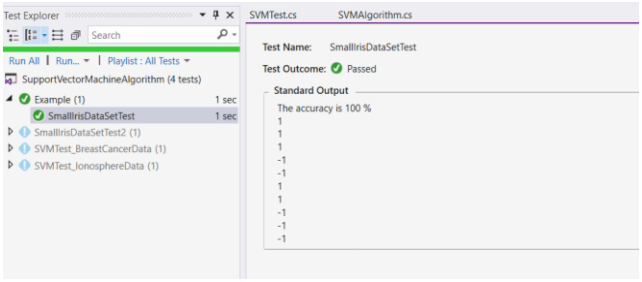

Figure 12: Testing result with modified Iris data

2) Iris data test [1]

This test is similar to Example Test, except Reading testing data. Instead of using hard-code, we can import data from csv file and load it into jagged double array[][].

- Training data: 90 samples
- Testing data: 10 samples
- Accuracy: 100% (Figure 13)

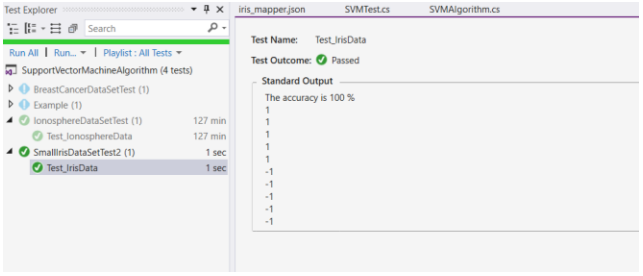

Figure 13: Iris data test result

3) Breast Cancer data set [1]

This dataset has 500 samples, including 3 features (BMI, Glucose, and Insulin) and one label (Class 1 and Class 2).

- Training data: 400 samples
- Testing data: 100 samples
- Accuracy: 100% (Figure 14)

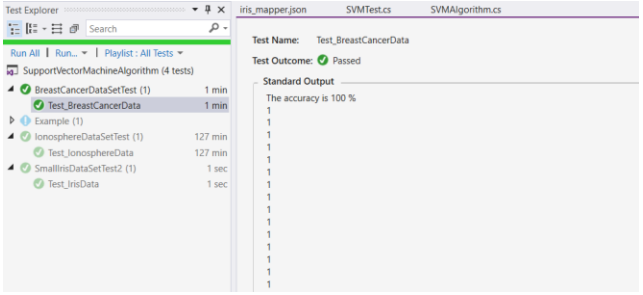

Figure 14: Breast Cancer data test result

4) Ionosphere data test [1]

This dataset has 351 samples, including 34 features (17 pairs of measurement value) and one label (good and bad).

- Training data: 280 samples
- Testing data: 71 samples
- Accuracy: around 93% (Figure 15)

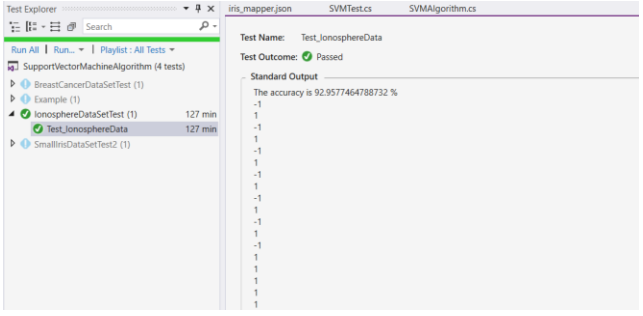

Figure 15: Ionosphere data test result

IV. DISCUSSION

A. Conclusion

Support Vector Machine is one of the best solution for classification problem and it has a lot of useful applications which are applied in real life, such as face detection, hand-writing recognition, text and hypertext categorization,

classification of images, etc. Besides huge advantages in SVM, there are still some disadvantages which can be improved in near future.

- **Pros**
  - It works really well with clear margin of separation.
  - It is effective in high dimensional spaces.
  - It is effective in cases where number of dimensions is greater than the number of samples.
  - It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- **Cons**
  - It doesn't perform well, when we have large data set because the required training time is higher.
  - It also doesn't perform very well, when the data has more noise (i.e. target classes are overlapping).

This project creates a linear classifier for a binary classification problem from the scratch by using Simplified Sequential Minimal Optimization algorithm to train data. After testing the performance, it reaches a very high accuracy, around 93-100%. The whole program is run in C# .NET Core.

B. Future Challenge

There are four main kernel functions: linear kernel, polynomial kernel, radial basis function (RBF) kernel, and sigmoid kernel. In this program, the linear kernel is applied to classify objects. Therefore, the program could be improved by adding new other kernels to classify.

V. REFERENCE

Bibliography

[1] Brownlee, J. (n.d.). *10 Standard Datasets for Practicing Applied Machine Learning*. Retrieved from https://machinelearningmastery.com/standard-machine-learning-datasets/

[2] Frankfurt University of Applied Sciences. (n.d.). *Learning API*. Retrieved from https://github.com/UniversityOfAppliedSciencesFrankfurt/LearningApi

[3] Lee, Y.-J. (n.d.). *Sequential Minimal Optimization (SMO)*. Retrieved from http://jupiter.math.nctu.edu.tw/~yuhjye/assets/file/teaching/2017_machine_learning/SMO%20algorithm.pdf

[4] Patel, S. (n.d.). *SVM (Support Vector Machine) - Theory*. Retrieved from https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72

[5] Platt, J. C. (1998). *Fast Training of Support Vector Machine.*