Investigating Support Vector Machines

Dongdong Lu

1 Introduction

In this report, we are going to have a brief look at the support vector machines.

"In machine learning, **support-vector machines** (**SVMs**, also **support-vector networks**) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis." - Wikipedia

Classifying data is a common task in machine learning. As a great tool for classification, support vector machines sound like a fancy term. It was initially put forward by Vladimir N. Vapnik in 1963. Then after the adaptation of non-linear kernels in the 1992, it has shown its good performance in a variety of settings and are often regarded as one of the best "out of the box" classifiers. According to the *TechVidvan* website, it has the following astonishing applications:

Inverse Geosounding Problem
Protein Fold and Remote Homology Detection
Speech Recognition
Seismic Liquefaction Potential
Data Classification using SSVM
Facial Expression Classification
Texture Classification using SVM
And so on…

We will begin our discussion from a simple and intuitive classifier called: maximal margin classifier which is a first edition SVM. This method is elegant with a significant drawback that it only applies to the most ideal case where different classes can be separated with a linear boundary. Then we extend it to support vector classifiers which can handle the complex case where both classes "mix with" each other. Finally, we will talk about the support vector machine, which is applying the same classification principle but, with the introduction of kernel function, it is upgraded to accommodate non-linear class boundaries.

People often vaguely call the maximal margin classifier, the support vector classifier, and the support vector machine as "support vector machines". To prevent confusion, we will clearly distinguish between these three terms in this report.

2 Method

Given a training dataset of n points of the form

$$(\overrightarrow{x_1}, y_1), \ldots, (\overrightarrow{x_n}, y_n)$$

y_is are 1 or -1 indicateing the classification of data point.

A hyperplane can be written as the set of points $\vec{x}$ satisfying

$$\vec{w} \cdot \vec{x} - b = 0$$

$\vec{w}$ is the normal vector to the hyperplane.

2.1 Hard Margin

Given the data points are linearly separable, we can generate two parallel hyperplanes separating the two classes of points, so that the width between them is as wide as possible. Margin is the region bounded by these two hyperplanes. The maximum-margin hyperplane is the hyperplane that lies midway between them.

$$\vec{w} \cdot \vec{x} - b = 1$$

(any point on or above this boundary is of the first class, labelled with 1)

$$\vec{w} \cdot \vec{x} - b = -1$$

(any point on or below this boundary is of the second class, labelled with −1).

The distance is computed using the distance from a point to the hyperplane. To avoid data points from falling into the margin, we introduce the following constraint:

$$\vec{w} \cdot \overrightarrow{x_i} - b \geq 1 \text{ if } y_i = 1 \text{ or } \vec{w} \cdot \overrightarrow{x_i} - b \leq -1 \text{ if } y_i = -1$$

That is the maximal margin classifier is to solve the optimization problem

$$\text{Maximize } ||\vec{w}|| \text{ subject to } y_i(\vec{w} \cdot \overrightarrow{x_i} - b) \geq 1 \text{ for } i = 1, \ldots, n$$

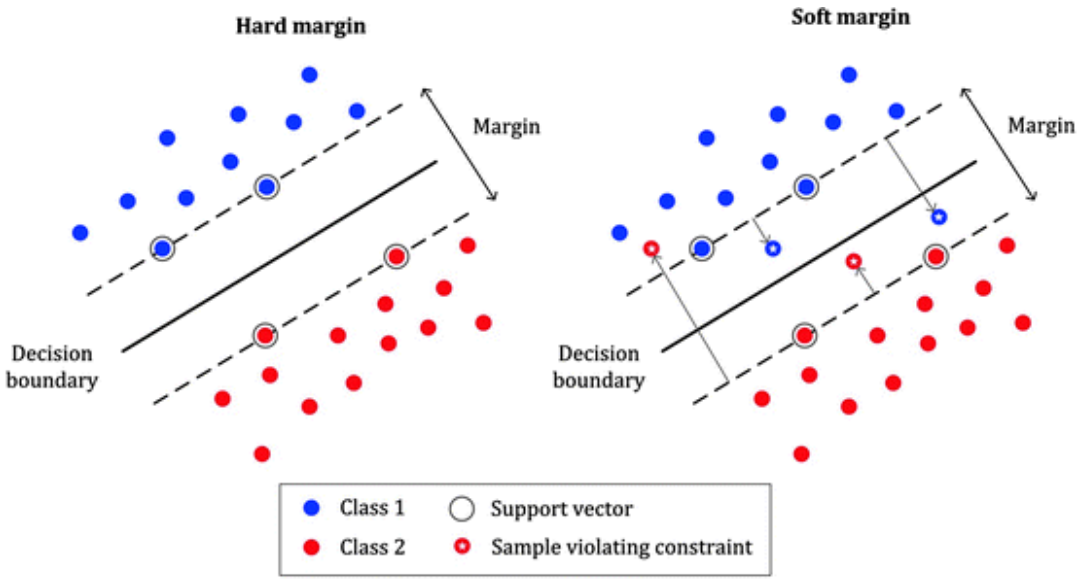

Figure 1 Dataset representation and hard/soft margins (cited from Medium website)

2.2 Soft Margin

When data points are not linearly separable, we add the *hinge loss* function,

$$\max(0,1 - y_i(\vec{w} \cdot \vec{x_i} - b))$$

Notice that $\vec{w} \cdot \vec{x_i} - b$ is the current output, so the function is 0 if $\vec{x_i}$ lies on the correct side of the margin and 1 if not. That is the classification problem is transformed into the following optimization problem

$$minimize\ [\frac{1}{n}\sum_{i=1}^{n} \max\ (0,1 - y_i(\vec{w} \cdot \vec{x_i} - b))] + \lambda||w||\text{^}2$$

Here the λ controls the trade-off between increasing the margin size and the number of $\vec{x_i}$s lie on the right sides of the margin.

2.3 Nonlinear classification

While the original maximum-margin hyperplane algorithm constructed a linear classifier. In 1992, Bernhard E. Boser, Isabelle M. Guyon and Vladimir N. Vapnik suggested a method to create nonlinear classifiers by using the kernel trick. The resulting algorithm remains the same, except that every dot product is substituted with a nonlinear kernel function. This permits the algorithm to fit the new maximum-margin hyperplane in a transformed space. The transformation may be nonlinear in the original feature space and the transformed space has more dimensions. Working in a higher-dimensional feature space increases the generalization error of support-vector machines.

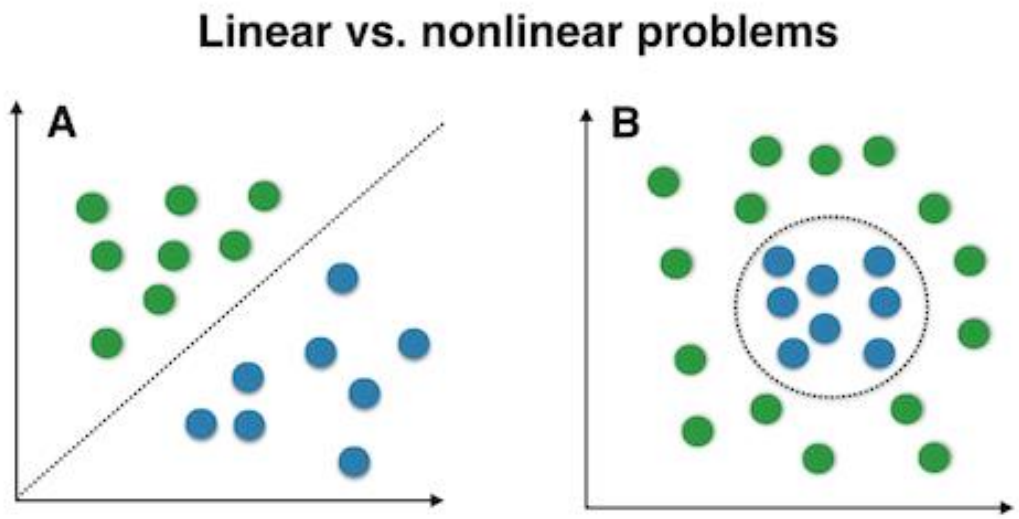

Figure 2 Comparison between linear and non-linear problems (cited from Stackoverflow website)

Some common kernels include:

- Polynomial (homogeneous): $k(\overrightarrow{x_i}, \overrightarrow{x_j}) = (\overrightarrow{x_i} \cdot \overrightarrow{x_j})^d$
- Polynomial (inhomogeneous): $k(\overrightarrow{x_i}, \overrightarrow{x_j}) = (\overrightarrow{x_i} \cdot \overrightarrow{x_j} + 1)^d$
- Gaussian radial basis function: $k(\overrightarrow{x_i}, \overrightarrow{x_j}) = \exp(-\gamma \|\overrightarrow{x_i} - \overrightarrow{x_j}\|^2)$ for $\gamma > 0$
- Hyperbolic tangent: $k(\overrightarrow{x_i}, \overrightarrow{x_j}) = \tanh(k\overrightarrow{x_i} \cdot \overrightarrow{x_j} + c)$ for some $k > 0$ and $c > 0$
- How SVM handles multiple classes?

2.4 How SVMs handle multiple classes?

There are several proposals for extending SVMs to multiple-class cases. The one-versus-one and one-versus-all are the most used two.

One-versus-one: Suppose we have K (>2) classes, one-versus-one method create $\binom{K}{2}$ SVMs. Each SVM compares a pair of classes. The final classification of an object is designed to the class which it has been assigned most into among the $\binom{K}{2}$ classifications. That is the one class which get the most vote wins.

One-versus-all: We create K SVMs, each SVM compare one of the K classes with the rest of $K-1$ classes. Suppose $\beta_{0k}, \beta_{1k}, \dots, \beta_{pk}$ be the parameters that get from fitting the k th class to the rest K-1 classes. Given $x^*$is a test observation, we assign it to the class where $\beta_{0k} + \beta_{1k}x_1^* + \cdots + \beta_{pk}x_p^*$ attains its maximum since this indicates that this classification has a higher level of confidence while compared with others.

3 Case study

3.1 Case study description

In this case study, we are going to have a look at SVM's performance on the "Pima Indian Diabetes" dataset which is from the National Institute of Diabetes and Digestive and Kidney Diseases. The Objective here is to diagnostically predict whether a patient has diabetes
All 768 patients here are females no less than 21 years old with Pima Indian ancestry. The data set consists of several medical predictor variables and one target variable "Outcome"
Predictor variables include the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

3.2 Data exploration

As usual, we begin our data exploration by looking at the data. Unfortunately, we have found that the dataset's quality is not good. Just for the first 10 patients, 1 is missing the blood pressure and 4 has missing values for skin sickness. Missing values may greatly reduce the quality of our data exploration and the performance of SVM's tuning as we will see later.

Then we want to see correlations among different predictors and the response variable. The correlation plot can be produced easily with the "sns.heatmap" function in Python. The created heat map that represent strong correlations between measures with deep red. Weak, or no, correlation is represented by dark color. The graph showed something expected and something unexpected. For example, we do expect the pregnancies is highly correlated with age. Since it is a common sense that the older the female patient is the more pregnancies, we would expect that she had. But what is unexpected is that glucose level has only 0.49 correlation with the outcome (diabetes or not). In my basic understanding of diabetes, physicians make the diagnose mostly based on the glucose level. So, glucose level should have a near 1 correlation with the outcome.

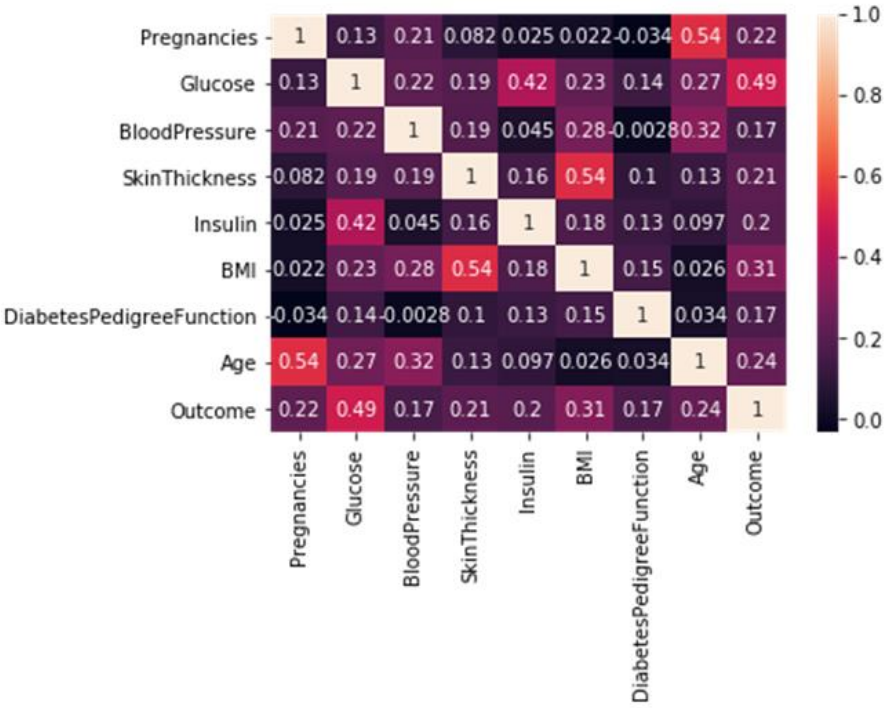

Figure 3 Heat map of correlations (cited from Kaggle website)

Secondly, I have made a graph to show the BMI distribution of diabetic and non-diabetic patients along with age. The motivation of this graph is a science paper I have read which mentions obesity as one of the causes of Type-2 diabetes. BMI is a good indicator of obesity. The graph below clearly shows that diabetic patients clearly has a higher BMI compared with their non-diabetic counterpart in the same age group which again resonate the reasonableness of the previous assumption.

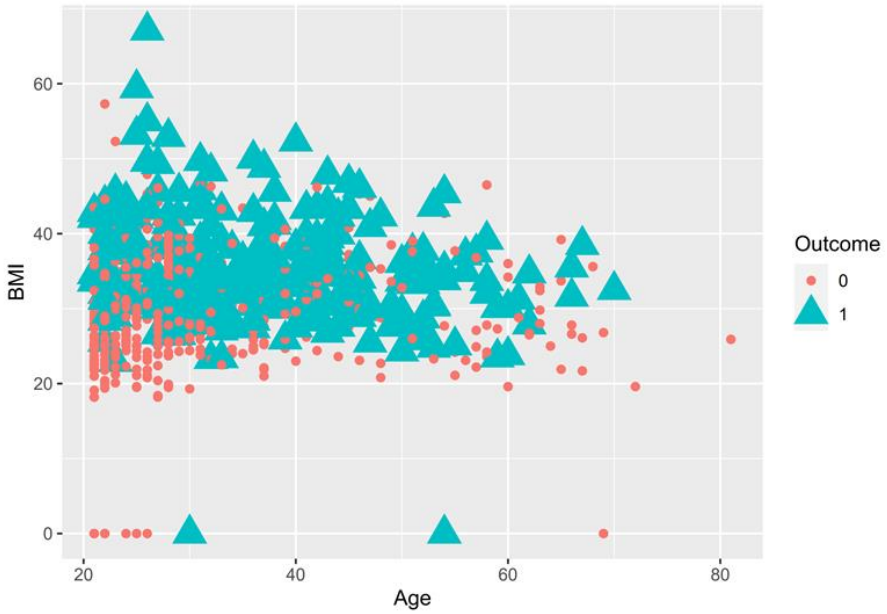

Figure 4 Distribution of BMI with age for different outcomes

3.3 Implementation and results

I first tried to split the data into training and test sets. Then apply the SVM on the training set to predict whether a person has diabetes or not, which give a training accuracy of 82.8%, then apply the model trained on test set, which gives an accuracy of 75%. Later, I tuned model with a range of gammas and costs. The "tune" function gives the best cost = 100 and gamma = 0.01. Plug in these tuned values, calculate the train and test ac curacies again, I find the train accuracy decreased to 80.3% and test accuracy remain the same 75%. This is interesting because the tuning didn't improve the prediction accuracy on the test set. Whether this is due to chance or there is some inner mechanism going on still need to be studied.

Before-tuning Train Accuracy: 0.8283

Train Set	0	1
0	363	75
1	28	134

- Test Accuracy: 0.75

Test Set	0	1
0	99	32
1	10	27

- Kernel: Radial and Tuning Range: Cost: 10^{-2} to 10^{3}, Gamma: $10^{-5}$to 10^{-1}
- Tuned Results: Cost: 100, Gamma: 0.01
- After-Tuning Training Accuracy: 0.8033

Train Set	0	1
0	363	90
1	28	119

Test Accuracy: 0.75

Test Set	0	1
0	98	31
1	11	28

4 Discussion

4.1 Why there is "no" benefit after tuning?

As we can see from the above result, classification result only changed a little bit after tuning, for example, the true positive cases which was successfully classified as positive increased from 27 to 28 but the true negative cases which was successfully classified as negative decreased from 99 to 98. This might due to randomness or/and the poor quality of the dataset which contains a significant amount of missing values. However, as I investigated the publications about SVM's tuning, most models see an improvement after tuning. In the future, when a high-quality dataset about the same research topic appears, doing the same implementation may shed more light into this topic.

4.2 What are the next steps?

It is also worth-noticing that we applied the SVM using only radian basis function kernel in our case. I choose this kernel simply because my intuition tells me that it should perform well. There are many other kernels could be tried with such as linear, polynomial, hyperbolic tangent and sigmoid kernels. Each kernel has its optimal performance in different scenarios, for example Laplace kernel would work the best for the Intrusion Detection System and radial basis function is the one for test analysis. May be a compare different kernels should be an extended step of the tuning, in other words, we are not only tuning the parameters and but also the type of kernels using.

4.3 What are the strengths and weaknesses of SVM?

Strengths:

SVM perform relatively better when clear margin of separation between classes exists.

SVM is more effective when the dimension is high.

SVM is effective in cases where number of $p \gg n$.

SVM uses memory more efficiently.

Weaknesses:

SVM does not work efficiently for large datasets.

SVM does not do well when target classes are overlapping.

SVM does not provide a probability for each of the classification.

4.4 Relationship to other methods

While the SVM's underlying idea of finding a hyperplane to separate the data with certain permission for violations seems to be novel as well as its idea of using a kernel to using kernel to fabricate the non-linear boundary at the beginning. But gradually researchers have found internal connections of SVMs with other statistical methods.

Notice that the criterion of fitting support vector classifier can be written as:

$$\underset{\beta_0, \dots, \beta_p}{minimize} \{\sum \max[0, 1 - y_i f(x_i)] + \lambda \sum \beta_j^2\}$$

Recall the "Loss+Penalty" form

$$\underset{\beta_0, \dots, \beta_p}{minimize} \{L(\boldsymbol{X}, \boldsymbol{y}, \beta) + \lambda P(\beta)\}$$

where $L(X, y, \beta)$ is some loss function that measures how well the model fits the data (X, y) and $P(\beta)$ is a penalty function on parameter vector β which is under the control of tuning parameter λ.

In ridge and lasso regression,

$$L(\boldsymbol{X}, \boldsymbol{y}, \beta) = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{i=1}^{n} x_{ij} \beta_j \right)^2$$

While in support vector classifier, it is called hinge loss, stated as:

$$L(\boldsymbol{X}, \boldsymbol{y}, \beta) = \sum_{i=1}^{n} \max\left[0, 1 - y_i \left(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}\right)\right]$$

which is closely related with the ones used by ridge (Placeholder1).

On the other hand, both SVM and Logistic Regression share similar loss functions, so they often produce very similar results.

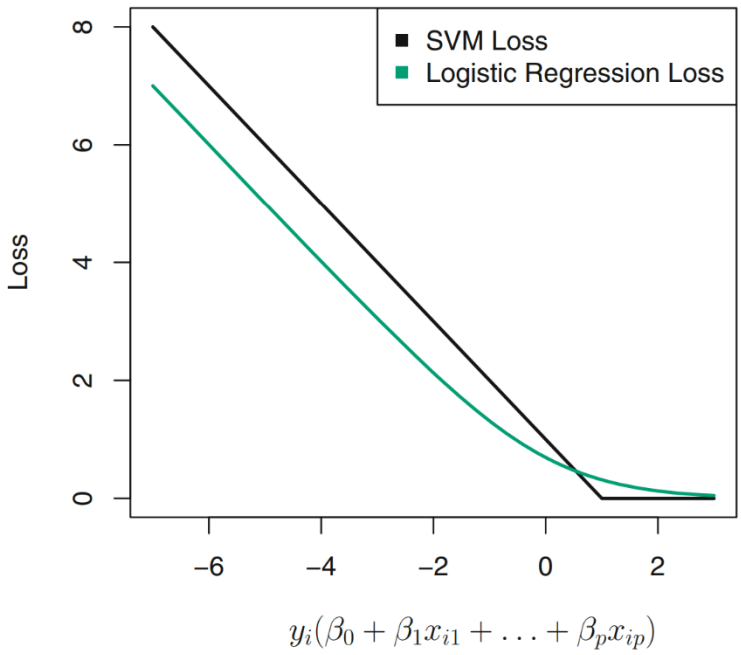

Figure 5 Comparison of SVM and Logistic Regression Loss as a function of $y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip})$*, we see the two loss function share similar performance. (Cited from ISL Book)*

In situations where classes are clearly separated, SVMs often have better performance than logistic regression, but when classes are overlapping with each other, logistic regression typically perform preferably.

Moreover, to fit non-linear boundaries, SVM's employment of the kernel technique is not unique. Logistic regression and several other classification methods can apply the non-linear kernel functions too.

5 Reference

Dawson, Carl. “A Guide to SVM Parameter Tuning.” *Medium*, Towards Data Science, 26 Sept. 2019, towardsdatascience.com/a-guide-to-svm-parameter-tuning-8bfe6b8a452c.

Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. An Introduction to Statistical Learning: with Applications in R. New York: Springer, 2013.

Ghisa, Florin, “Linear SVM vs Nonlinear SVM High Dimensional Data.” *Stack Overflow*, stackoverflow.com/questions/44606126/linear-svm-vs-nonlinear-svm-high-dimensional-data.

Hasan, Md. Al & Xu, Shuxiang & Kabir, Mir & Ahmad, Shamim. (2016). Performance Evaluation of Different Kernels for Support Vector Machine Used in Intrusion Detection System. International journal of Computer Networks & Communications. 8. 39-53. 10.5121/ijcnc.2016.8604.

K, Dhiraj. “Top 4 Advantages and Disadvantages of Support Vector Machine or SVM.” *Medium*, Medium, 13 June 2019, medium.com/@dhiraj8899/top-4-advantages-and-disadvantages-of-support-vector-machine-or-svm-a3c06a2b107.

Mourya A.K., ShafqatUlAhsaan, Kaur H. (2020) Performance and Evaluation of Different Kernels in Support Vector Machine for Text Mining. In: Mohanty M., Das S. (eds) Advances in Intelligent Computing and Communication. Lecture Notes in Networks and Systems, vol 109. Springer, Singapore

Support-vector Machine

https://en.wikipedia.org/wiki/Support-vector_machine

India The Diabetes Capital Of World

Biphili - https://www.kaggle.com/biphili/india-the-diabetes-captial-of-world

E1071

https://www.rdocumentation.org/packages/e1071/versions/1.7-3/topics/svm

6 Appendix

6.1 Link to the Pima Indian Diabetes dataset

https://www.kaggle.com/uciml/pima-indians-diabetes-database

6.2 Code:

```{r}
# Input the data
library("e1071")
library(readr)
diabetes <- read_csv("C:/Users/Don/Desktop/SL/Final Project/diabetes.csv")

# Split the data set into train and test sets
set.seed(123)
train_pima <- sample(1:768, size = 600)
train <- diabetes[train_pima, ]
test <- diabetes[-train_pima, ]
```

```{r}
# Use all SVM and all variables to do the classification for outcome on trainning set
svm_mod=svm(Outcome~.,data=train,type='C-classification')

# Summary of the first model
summary(svm_mod)
```

calculate the prediction accuracy for the trainning set

pred = predict(svm_mod,newdata = train)

table(pred,train$Outcome)

mean(pred == train$Outcome)

calculate the prediction accuracy for the test set

pred = predict(svm_mod,newdata = test)

table(pred,test$Outcome)

mean(pred == test$Outcome)

#The test accurate rate is 75%

```

```{r}

#Tune the SVM parameters on training set

svm_tune <- tune(svm, train.x=train, train.y=train$Outcome, kernel="radial",
ranges=list(cost=10^(-2:3), gamma=10^(-5:-1)))

summary(svm_tune)

# Optimal cost is 100 and gamma is 0.01

svm_tuned <- svm(Outcome ~ ., data=train, type='C-classification',kernel="radial", cost=100,
gamma=0.01)

summary(svm_tuned)

```

```{r}

```
Calculate the trainning error

pred <- predict(svm_tuned,train)

system.time(predict(svm_tuned,train))

table(pred,train$Outcome)

mean(pred == train$Outcome)

Calculate the test error

pred <- predict(svm_tuned,test)

system.time(predict(svm_tuned,test))

table(pred,test$Outcome)

mean(pred == test$Outcome)
```