Course Project
CISC3023 Machine Learning

Traffic Sign Classification

(Report)

澳門大學

UNIVERSIDADE DE MACAU
UNIVERSITY OF MACAU

Student :
Wong Kai Yuan (DC026157)

Prof :
Prof Long CHEN

Table of Content :

Introduction

In this project, we are given 15540 images of traffic signs in .ppm format, under 15 different classes where this information is stored in each .csv file. Our goal is to classification on such dataset using 3 classifiers (KNN is excluded)

Although there are many different classifiers that can we use, we choose below 3 classifiers for our project (reason being and parameter will be included in “details of the three classifier used” part)

A) Random Forest

Random Forest (a.k.a Random Decision Forest) is an ensemble classifier. This term was first introduced by and was first proposed by Tin Kam Ho of Bell Labs in 1995. It operates by constructing a multitude of decision trees and output the class selected by most trees.

B) Support Vector Machine (SVM)

Support Vector Machine (SVM) is a supervised learning algorithm used for classification, regression and outlier detection.

C) Neural Network (Multilayer Perceptron MLP)

A neural network is a streamlined representation of how the human brain functions. It simulates a huge number of connected processing units that mimic abstract representations of neurons in order to function. Layers of arrangement make up the processing units.

Multilayer perceptron is a form of feedforward artificial neural network that is fully connected. When used ambiguously, the term "MLP" can refer to any feedforward ANN.

The basic steps for our project will be :

1) Read all training data
2) Do preprocessing on data
3) Split data into training & validation data
4) Define our classifiers
5) Fit data to our classifiers (training process)
6) Get accuracy & confusion matrix on training & validation data
7) Dump our model with best parameters to device for testing purpose

Due to the completeness of the given data with different size and labels, some ubiquitous methods such as preprocessing (more details at “preprocessing” part) and updating the parameter is suitable. Updating parameters can improve the classifiers’ performance in general, so we do this progress on the 3 classifiers. (more details at “set parameter” part)

Instruction to use the code

The project will be written in Python language and below are the package we need :

- Matplotlib
- Csv
- Numpy
- Sklearn
- Pickle
- Os
- Cv2
- Skimage

There are 3 codes for this report, namely :

1) For_report_usage.ipynb :

This notebook of code is just for writing report purposes.

- Try different ways of image preprocessing.
- Trying different parameters to get the best model.

Actual trainer program & testing program is in the notebook below.

2) Final_train_program.ipynb :

This is the actual Training Program, the main program is to run the trainData() function.
Note : need to change training_data_directory to the actual training dataset.

```
training_data_directory = "/Users/kaiyuan/Desktop/TrafficSignData/Training"
```

Input : training_data
Output : will dump out model to device for later testing purpose

```
with open("FinalrandomForest.pkl","wb") as op_file:
    pickle.dump(randomForest, op_file)
```

We will use training data to train 3 models with the best parameters possible. Each model will show its training accuracy, validation accuracy, fitting duration, confusion matrix, cross validation score, cross validation accuracy

```
print("MLP, training accuracy: %.3f, testing accuracy: %.3f" % (
    clf.score(X_train, y_train), clf.score(X_val, y_val)))
print(f"Fitting Duration (seconds) :{duration:.4f}s")
print(f"Confusion Matrix :")
print(model_confusion_matrix)


cv_scores = cross_val_score(clf, X_train, y_train, cv=5)
print("Cross-Validation-Scores: "+str(cv_scores))
print("MAX Cross-Validation-Score: "+str(max(cv_scores)))
print("MIN Cross-Validation-Score: "+str(min(cv_scores)))
print("Cross-Validation-Accuracy: %.6f (+/- %0.2f)" % (cv_scores.mean(), cv_scores.std() * 2))
```

3) Testing_Program.ipynb :

This is Testing Program. The main program for this notebook is testProgram(clf, rootpath) to get output.

Output of the testing program will be :
1) Accuracy of testing data
2) Confusion matrix
of each model

The input of testProgram(clf,rootpath) is :
1) clf can be :
"all" : will run testing data on all models
"RF" : will run testing data on Random Forest model only
"SVM" : will run testing data on SVM model only
"MLP" : will run testing data on MLP neural network model only
2) rootpath :
it's the directory of testing data, can put as "testing_directory"

Note : need to change testing_directory & model_directory to correct directory
(the model_directory included model that are output from previous training notebook)

```
testing_directory = "/Users/kaiyuan/Desktop/Test"
model_directory = '/Users/kaiyuan/Desktop/My_model/'
```

Details of the three classifiers used

Firstly, we need to answer the question “Why are these classifiers chosen? ”

1) Random Forest : This classifier is chosen because it is one of the most accurate algorithms and it has better efficiency handling large databases.
2) SVM : This classifier is chosen because it is effective in high dimensional spaces which means it will be effective handling multi-class classification.
3) Neural Network MLP : This classifier is chosen due to their natural capacity to aggregate tiny elements into complex abstractions, neural networks are excellent at handling complex inputs like images, audio, and text. The meaningless pixels are joined repeatedly to create patterns and patterns of patterns.

Secondly, I will show you the parameter we used for each classifier. (details of choosing parameter will be included in the “set parameter” part)

Random Forest

The main parameter to adjust for random forest is n_estimators & max_features.

```
RandomForestClassifier(n_estimators=40, random_state=0,max_features = "sqrt")
```

- N_estimators = 40
- Max_features = “sqrt”

Support Vector Machine (SVM)

The main parameter to adjust for SVM is kernel & C.

```
clf = svm.SVC(kernel="rbf", C=10.5)
```

- Kernel : “rbf”
- C_parameter = 10.5

Neural Network (MLP)

The main parameter to adjust for MLP is optimizer & hidden_layer.

```
clf = MLPClassifier(solver='sgd', alpha=1e-5,
                    hidden_layer_sizes=(100,), random_state=0)
```

- Optimizer = sgd
- Hidden_layer = 100

The handling of multi-class problem

At this point, we can answer another question: “Why are other classifiers not chosen ?“)

In our lecture, there are many machine learning models / classifiers that are suitable to do classification (excluding KNN) :

- Decision Tree / Random Forest
- Naïve Bayes Classifier
- Linear Regression
- Logistic Regression
- Support Vector Machine
- Neural Network

However, to handle multi-class problems for large datasets, we excluded linear regression (suitable for binary classification) and decision tree (low computational efficiency and complexity for large datasets). Later, we reject the Naïve Bayes Classifier since it requires the assumption of conditional independence of features, which is not realistic for image data.

Second part, we will discuss how each of the chosen models handle the multi-class problem.

1) Random Forest :

   This classifier is designed with the ability to handle multiclass classification.

2) Support Vector Machine :

   The training algorithm uses One-versus-Rest (a.k.a. One-Versus-All) scheme. It will split the data into multiple binary classification problems, for example it will classify as “Label 1” and “Other”, and repeat this classification for all the remaining. It will then predict a class probability and the argmax of these probabilities will be used to predict a class.
   ie.
   Problem 1: Class 00000 vs [Class 00001, Class 00002, …, Class 00014]
   Problem 2: Class 00001 vs [Class 00000, Class 00002, Class 00003…, Class 00014]
   …
   Problem 15: Class 00014 vs [Class 00000, Class 00001, …, Class 00013]

3) Neural Network (MLP) :

   Using Softmax function instead of Sigmoid function as activation function for multi classification problem, cross-entropy loss as loss function to minimises the loss and use stochastic gradient descent rather than regular gradient descent.

How do you preprocess your data to get better result ?

The figure below shows the function readTrafficSigns. Within this function, the images are input and
(1) resized into a given size (32x32), so that all the input pictures are the same size. And the respective label is recorded in order to check its accuracy.
A local contrast enhancement approach called
(2) Contrast Limited Adaptive Histogram Equalization (CLAHE), which makes use of histograms calculated over various tile sections of the image, was also used. Therefore, even in places that are darker or lighter than the majority of the image, local details can be improved. This ought to facilitate feature extraction.
Furthermore, the image are
(3) converted into a grey image from RGB image. Such preprocessing allows to reduce the numbers of channels in the input of the network without decreasing the performance.
(4) Lastly, the image will be flatten, which can smooth the performance of training.

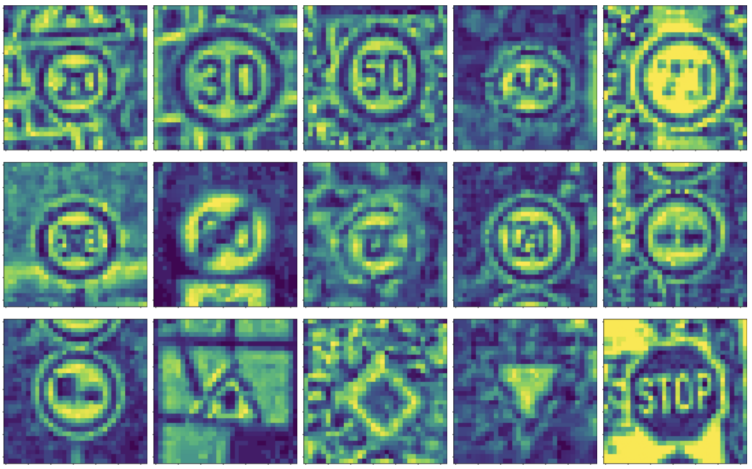
```
def readTrafficSigns(rootpath):
    '''Reads traffic sign data
    Arguments: path to the traffic sign data, for example './TrafficSignData/Training'
    Returns:   list of images, list of corresponding labels'''
    images = [] # images
    labels = [] # corresponding labels
    # loop over N classes, at most we have 42 classes
    N=15
    for c in range(0,N):
        prefix = rootpath + '/' + format(c, '05d') + '/' # subdirectory for class
        gtFile = open(prefix + 'GT-'+ format(c, '05d') + '.csv') # annotations file
        gtReader = csv.reader(gtFile, delimiter=';') # csv parser for annotations file
        #gtReader.next() # skip header
        next(gtReader)
        # loop over all images in current annotations file
        for row in gtReader:
            img=Image.open(prefix + row[0])  # the 1th column is the filename
            # preprocesing image, make sure the images are in the same size
            img=img.resize((32,32), Image.BICUBIC)

            img=np.array(img)
            img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
            img=exposure.equalize_adapthist(img,clip_limit=0.1)
            images.append(img)
            labels.append(row[7]) # the 8th column is the label
        gtFile.close()
    return images, labels
```

Below are images of each class before preprocessing.

Below are images of each class after (2) CLAHE

Below are images of each class after (3) convert into grey



How do you set the parameters of different classifiers ?

For each model, we are outputting :

- Training & Validation accuracy
- Confusion Matrix of validation data
- Fitting duration (time in second)

And when we able to find the best parameter possible for each model, we will also output :

- Cross Validation Score
- MAX Cross Validation Score
- MIN Cross Validation Score
- Cross Validation Accuracy

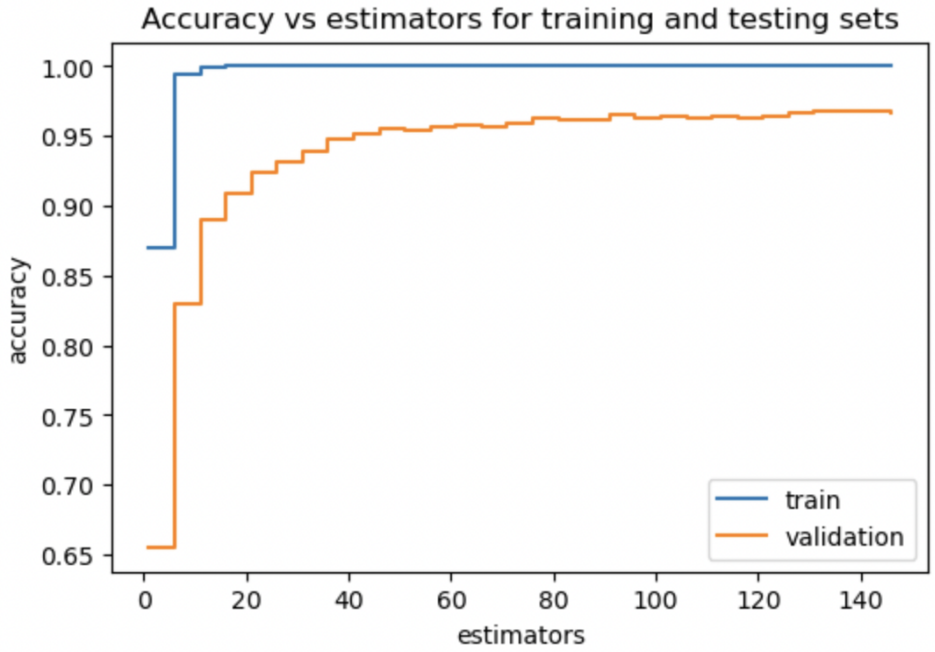
A) Random Forest

- Firstly, we do a Random Forest classifier with default parameters. Below is the result :

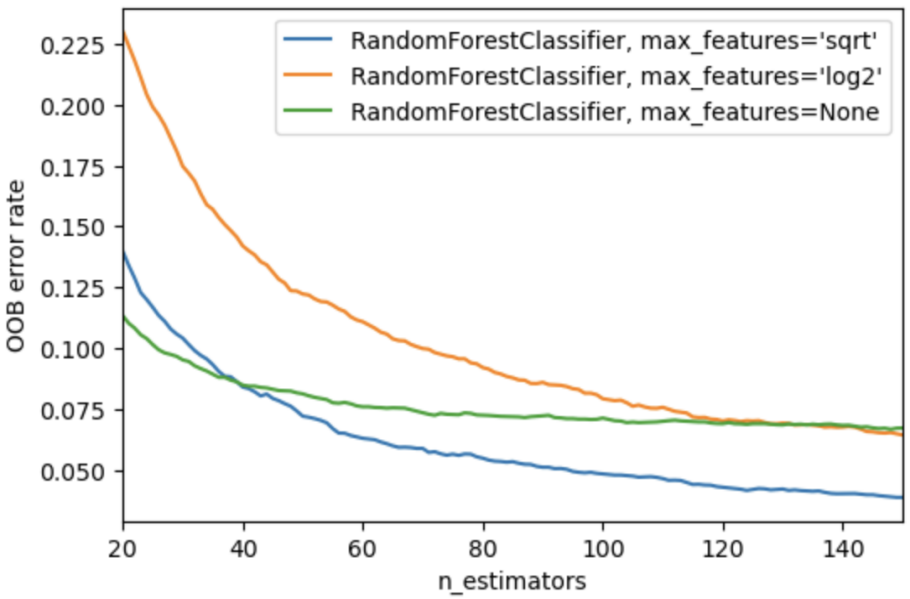
```
randomForest = RandomForestClassifier(n_estimators=10, random_state=0)
start_time = time.time()
randomForest = randomForest.fit(X_train, y_train)
end_time = time.time()
duration = end_time - start_time
model_confusion_matrix = confusion_matrix(y_val,randomForest.predict(X_val))
```

```
RandomForestClassifier, training accuracy: 1.000, validation accuracy: 0.900
Fitting Duration (seconds) :2.10s
Confusion Matrix :
[[  8   3   1   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0 132   6   1   2   5   0   0   0   0   0   0   0   0   0]
 [  0   7 114   6   4   3   0   1   1   0   1   0   0   0   0]
 [  0   5   5  82   0   5   0   1   3   0   0   0   0   0   0]
 [  0   4   2   0 128   0   0   0   0   0   0   0   0   0   0]
 [  0  11  10   5   2  91   0   0   1   0   1   0   2   0   0]
 [  0   0   0   0   0   0  32   0   0   0   0   0   1   0   0]
 [  0   3   2   0   0   6   0  84   3   0   0   0   0   0   0]
 [  0   1   3   2   2   2   0   5  97   0   0   0   0   0   0]
 [  0   2   1   0   0   1   0   1   1  82   0   0   0   1   0]
 [  0   0   1   1   0   0   0   0   0   1 136   0   0   0   0]
 [  0   1   1   1   0   0   0   1   1   0   0  79   0   0   0]
 [  0   1   2   0   0   0   0   0   1   1   0   0 129   0   0]
 [  0   0   0   3   0   0   0   2   0   0   0   1   0 153   0]
 [  0   0   0   0   0   1   1   0   0   0   0   0   0   0  51]]
```

- Because the main parameters we can adjust are n_estimators & max_features, we can plot graphs to estimate the best value for them.



- Above is the graph of train & validation accuracy against estimators with value from 20 to 150. This parameter decides the number of decision trees.



- Above is the graph of OOB error rate against estimators with values from 20 to 150, and each line represents a classifier with different parameters of max_features of {‘sqrt’, ‘log2’, ‘None’}. This parameter will tell the number of features to consider when looking for the best split.

- By using the above graphs, we decide to choose n_estimators = 40, and max_feature = “sqrt” for the best random forest we can do. Below is the result when the best parameters are used :

```
RandomForestClassifier, training accuracy: 1.000, validation accuracy: 0.956
Fitting Duration (seconds) :7.47s
Confusion Matrix :
[[  8   3   0   0   1   0   0   0   0   0   0   0   0   0   0]
 [  0 138   0   0   5   3   0   0   0   0   0   0   0   0   0]
 [  0   1 130   0   5   1   0   0   0   0   0   0   0   0   0]
 [  0   2   2  91   0   3   0   2   1   0   0   0   0   0   0]
 [  0   2   0   0 132   0   0   0   0   0   0   0   0   0   0]
 [  0   8   2   1   0 107   0   1   4   0   0   0   0   0   0]
 [  0   0   0   0   0   0  32   0   0   0   0   0   1   0   0]
 [  0   1   0   0   0   4   0  92   1   0   0   0   0   0   0]
 [  0   0   0   0   0   1   0   2 109   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  86   1   0   1   1   0]
 [  0   0   1   0   0   1   0   0   0   0 137   0   0   0   0]
 [  0   0   1   0   0   1   0   1   0   0   0  81   0   0   0]
 [  0   0   0   0   0   0   0   0   1   0   0   0 133   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   1   1 157   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  53]]
Cross-Validation-Scores: [0.9424589  0.94923132 0.94887379 0.93707544 0.94422596]
MAX Cross-Validation-Score: 0.9492313192706471
MIN Cross-Validation-Score: 0.9370754379692527
Cross-Validation-Accuracy: 0.944373 (+/- 0.01)
```

- As we can see from the above result, the classifier improved in terms of accuracy when parameters were changed.
  - Validation Accuracy before : 0.900
  - Validation Accuracy after : 0.956

-> Cross Validation Accuracy for this classifier is 0.944

B) Support Vector Machine

As mentioned previously, the main parameter to adjust for SVM is kernel & C.

- Kernel: {“linear”, “poly”, “rbf”} To decide which kernel to use, we train the model with each of the models and choose the output with highest accuracy. Below are the results :
- 1) Kernel : “linear”

```
SVM, training accuracy: 1.000, testing accuracy: 0.957
Fitting Duration (seconds) :23.2604s
Confusion Matrix :
[[ 12   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0 136   7   0   0   0   0   0   2   0   0   0   1   0   0]
 [  0   8 124   0   1   2   0   1   0   0   0   0   1   0   0]
 [  0   1   1  95   1   2   0   0   0   1   0   0   0   0   0]
 [  1   2   0   0 131   0   0   0   0   0   0   0   0   0   0]
 [  0   1   4   4   1 111   0   0   1   0   1   0   0   0   0]
 [  0   0   0   0   0   0  33   0   0   0   0   0   0   0   0]
 [  0   1   3   0   0   1   0  92   1   0   0   0   0   0   0]
 [  0   0   0   0   0   1   0   2 108   0   1   0   0   0   0]
 [  0   0   1   0   0   0   0   0   0  86   1   0   0   1   0]
 [  0   0   0   1   0   0   0   0   0   0 138   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  84   0   0   0]
 [  0   0   1   0   0   0   0   2   0   1   0   1 129   0   0]
 [  0   0   1   0   0   1   0   0   1   0   0   1   0 155   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  53]]
```

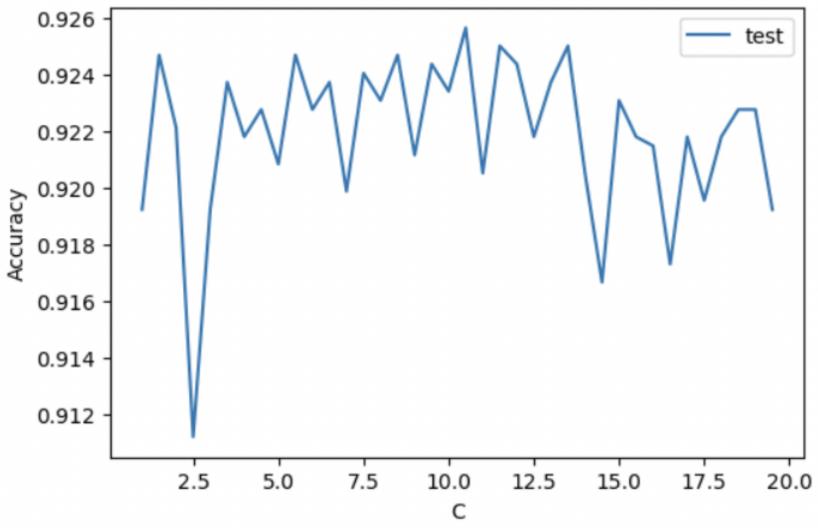
- 2) Kernel : “poly”

```
SVM, training accuracy: 1.000, testing accuracy: 0.967
Fitting Duration (seconds) :23.8326s
Confusion Matrix :
[[ 12   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0 141   3   0   0   0   0   1   0   0   0   0   1   0   0]
 [  0   5 130   0   1   0   0   0   0   0   0   0   1   0   0]
 [  1   2   1  94   0   2   0   1   0   0   0   0   0   0   0]
 [  1   1   1   0 131   0   0   0   0   0   0   0   0   0   0]
 [  0   5   3   2   0 111   0   0   1   0   1   0   0   0   0]
 [  0   0   0   0   0   0  32   0   0   0   0   0   1   0   0]
 [  0   1   3   0   0   1   0  93   0   0   0   0   0   0   0]
 [  0   0   1   0   0   0   0   0 111   0   0   0   0   0   0]
 [  0   0   0   0   0   1   0   0   0  87   0   0   0   1   0]
 [  0   0   0   1   0   0   0   0   0   0 138   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  84   0   0   0]
 [  0   0   0   0   0   0   0   0   0   1   0   1 132   0   0]
 [  2   0   0   1   1   0   0   0   1   0   0   1   0 153   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  53]]
```

- 3) Kernel : “rbf”

```
SVM, training accuracy: 0.996, testing accuracy: 0.969
Fitting Duration (seconds) :55.4071s
Confusion Matrix :
[[ 11   1   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0 137   5   0   2   0   0   1   1   0   0   0   0   0   0]
 [  0   2 132   0   1   1   0   0   0   0   0   0   1   0   0]
 [  0   3   2  92   0   4   0   0   0   0   0   0   0   0   0]
 [  0   2   0   0 132   0   0   0   0   0   0   0   0   0   0]
 [  0   3   1   2   0 115   0   0   1   0   0   0   1   0   0]
 [  0   0   0   0   0   0  32   0   0   0   0   0   1   0   0]
 [  0   0   2   0   0   2   0  93   1   0   0   0   0   0   0]
 [  0   0   1   0   0   2   0   0 109   0   0   0   0   0   0]
 [  0   0   0   1   0   0   0   0   0  87   0   0   0   1   0]
 [  0   0   0   0   0   1   0   0   0   0 138   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  84   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0 134   0   0]
 [  0   0   0   0   0   1   0   0   0   0   0   0   1 157   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  53]]
```

- As we can tell from above results, the kernel that can achieve the best result is Kernel : “rbf” where it can achieve a validation accuracy of 0.969. Therefore, the parameter we choose for kernel is “rbf”
- Next, we need to have a look at the C_parameter. It’s the parameter for regularisation. The higher the C value, the lower the strength of regularisation.



- Above is the graph of accuracy against C value from 1 to 20. . It does not have a nice growing curve to be determined as it is for regularisation. Finally, we decided to use the value C=10.5. This is the value of maximum accuracy in this graph, also this value could reduce misclassifying compared to default value (C=1.0)
- By using the above graphs, we decide to choose kernel = 'rbf', and C_parameter = 10.5 for the best SVM we can do. Below is the result when the best parameters are used :

```
SVM, training accuracy: 1.000, testing accuracy: 0.985
Fitting Duration (seconds) :55.5704s
Confusion Matrix :
[[ 12   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0 144   1   0   0   0   0   0   1   0   0   0   0   0   0]
 [  0   2 134   0   1   0   0   0   0   0   0   0   0   0   0]
 [  0   1   1  98   0   0   0   1   0   0   0   0   0   0   0]
 [  0   0   1   0 133   0   0   0   0   0   0   0   0   0   0]
 [  0   2   1   0   0 118   0   0   1   0   0   0   1   0   0]
 [  0   0   0   0   0   0  32   0   0   0   0   0   1   0   0]
 [  0   0   3   0   0   0   0  95   0   0   0   0   0   0   0]
 [  0   0   0   0   0   1   0   0 111   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  88   0   0   0   1   0]
 [  0   0   0   0   0   1   0   0   0   0 138   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  84   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0 134   0   0]
 [  0   0   0   0   0   1   0   0   0   0   0   0   1 157   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  53]]
Cross-Validation-Scores: [0.97855611 0.98140865 0.98176618 0.98283876 0.9792635 ]
MAX Cross-Validation-Score: 0.9828387558097962
MIN Cross-Validation-Score: 0.9785561115082202
Cross-Validation-Accuracy: 0.980767 (+/- 0.00)
```

- As we can see from the above result, the classifier improved in terms of accuracy when parameters were changed.
  - Validation Accuracy before : 0.969
  - Validation Accuracy after : 0.985

-> Cross Validation Accuracy for this classifier is 0.980

C) Neural Network - MLP

As mentioned previously, the main parameter to adjust for MLP is optimizer & hidden_layer.

- Optimizer: {"adam", "lbfgs", "sgd"} To decide which optimizer to use, we train the model with each of the models and choose the output with highest accuracy. Below are the results :

- 1) Optimizer : “adam”

```
MLP, training accuracy: 1.000, testing accuracy: 0.927
Fitting Duration (seconds) :35.4454s
Confusion Matrix :
[[ 11   0   0   0   0   0   0   0   0   0   0   0   1   0   0]
 [  1 132   6   0   1   0   0   1   4   0   0   0   1   0   0]
 [  0   3 120   2   1   7   0   3   0   0   0   0   1   0   0]
 [  0   1   2  89   0   8   0   0   0   1   0   0   0   0   0]
 [  1   2   0   0 126   3   0   0   2   0   0   0   0   0   0]
 [  1   3   2   4   0 106   0   1   1   0   3   0   2   0   0]
 [  0   0   0   0   0   0  32   0   0   0   0   0   1   0   0]
 [  0   1   2   1   0   3   0  86   3   0   2   0   0   0   0]
 [  0   1   2   2   0   1   0   1 105   0   0   0   0   0   0]
 [  0   0   0   1   0   0   0   0   0  83   2   0   2   1   0]
 [  0   0   0   1   0   2   0   0   0   0 136   0   0   0   0]
 [  0   0   1   0   0   0   0   0   0   0   1  82   0   0   0]
 [  0   0   1   0   0   1   0   0   0   4   3   0 125   0   0]
 [  1   1   1   0   0   0   0   0   1   0   0   0   0 155   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  53]]
```

click to scroll output; double click to hide

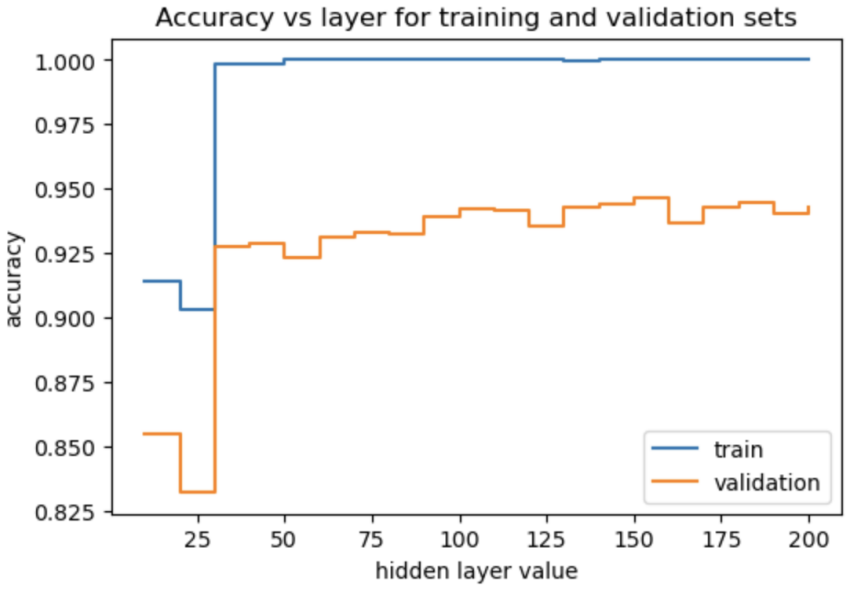
- 2) Optimizer : “lbfgs”

```
MLP, training accuracy: 0.994, testing accuracy: 0.927
Fitting Duration (seconds) :12.0686s
Confusion Matrix :
[[  8   2   0   0   0   1   0   0   0   0   0   0   1   0   0]
 [  2 130   7   1   2   1   0   0   2   0   0   0   1   0   0]
 [  0   2 122   4   2   3   0   3   0   0   0   0   1   0   0]
 [  0   1   1  92   0   6   0   0   0   1   0   0   0   0   0]
 [  0   3   0   0 128   3   0   0   0   0   0   0   0   0   0]
 [  0   3   2   8   1 101   0   1   1   0   2   0   2   2   0]
 [  0   0   0   0   0   0  33   0   0   0   0   0   0   0   0]
 [  0   0   1   1   1   5   0  86   2   1   0   1   0   0   0]
 [  0   0   0   1   2   1   0   2 105   0   1   0   0   0   0]
 [  0   0   2   1   0   0   0   0   1  83   1   0   1   0   0]
 [  0   0   0   0   1   2   0   0   0   0 136   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  84   0   0   0]
 [  0   0   2   0   0   0   0   1   0   2   2   1 126   0   0]
 [  1   2   0   0   0   1   0   0   1   0   0   0   0 154   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  53]]
```

- 3) Optimizer : “sgd”

```
MLP, training accuracy: 0.968, testing accuracy: 0.932
Fitting Duration (seconds) :31.8551s
Confusion Matrix :
[[  8   3   0   0   0   1   0   0   0   0   0   0   0   0   0]
 [  2 128  11   0   3   0   0   0   1   0   0   0   1   0   0]
 [  0   2 123   3   1   6   0   0   0   1   0   0   1   0   0]
 [  0   1   1  92   0   5   0   1   0   0   0   0   0   1   0]
 [  1   2   0   0 130   0   0   0   1   0   0   0   0   0   0]
 [  1   4   5   3   0 107   0   0   1   0   1   1   0   0   0]
 [  0   0   0   0   0   0  33   0   0   0   0   0   0   0   0]
 [  0   2   0   0   1   4   0  86   3   2   0   0   0   0   0]
 [  0   0   2   1   1   1   0   1 105   1   0   0   0   0   0]
 [  0   0   1   0   0   0   0   0   1  86   0   0   1   0   0]
 [  0   0   0   1   1   4   0   0   0   1 132   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   1  83   0   0   0]
 [  0   1   0   0   0   0   0   0   0   0   1   1 130   1   0]
 [  1   2   0   0   0   1   0   1   0   0   0   0   0 154   0]
 [  0   1   0   0   0   0   0   0   0   0   0   0   0   0  52]]
```

- As we can tell from above results, the optimizer that can achieve the best result is solver : “sgd” where it can achieve a validation accuracy of 0.932. Therefore, the parameter we choose for solver is “sgd”
- Next, we need to have a look at the hidden layer. Below is the graph of accuracy of train & validation data against hidden layer value.



- By using the above graphs, we decide to choose solver = “sgd”, and hidden_layer = 100 for the best MLP we can do. Below is the result when the best parameters are used :

```
MLP, training accuracy: 0.969, testing accuracy: 0.933
Fitting Duration (seconds) :49.8882s
Confusion Matrix :
[[  8   3   0   0   0   1   0   0   0   0   0   0   0   0   0]
 [  1 131  10   0   2   0   0   0   1   0   0   0   1   0   0]
 [  0   3 123   2   2   6   0   0   0   1   0   0   0   0   0]
 [  0   2   1  91   0   4   0   2   1   0   0   0   0   0   0]
 [  1   3   0   0 127   0   0   0   3   0   0   0   0   0   0]
 [  0   5   3   5   0 106   0   0   1   0   2   0   0   1   0]
 [  0   0   0   0   0   0  33   0   0   0   0   0   0   0   0]
 [  0   1   0   0   0   4   0  90   1   1   1   0   0   0   0]
 [  0   1   2   0   0   2   0   0 106   0   1   0   0   0   0]
 [  0   0   1   0   0   0   0   0   1  84   2   0   1   0   0]
 [  0   0   0   0   1   5   0   0   0   0 133   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   1  83   0   0   0]
 [  0   1   1   0   0   0   0   0   0   1   1   1 129   0   0]
 [  1   2   0   0   0   1   0   0   1   0   0   0   0 154   0]
 [  0   1   0   0   0   0   0   0   0   0   0   0   0   0  52]]

Cross-Validation-Scores: [0.93102216 0.93886307 0.93457276 0.93814802 0.92670719]
MAX Cross-Validation-Score: 0.938863067572399
MIN Cross-Validation-Score: 0.9267071862710047
Cross-Validation-Accuracy: 0.933863 (+/- 0.01)
```

- As we can see from the above result, the classifier improved in terms of accuracy when parameters were changed.
  - Validation Accuracy before : 0.932
  - Validation Accuracy after : 0.933

-> Cross Validation Accuracy for this classifier is 0.9339

What is the performance of three classifiers ?

From above results, we are able to get 3 classifiers with their own best possible parameters. I will show the output again below for further discussion.

1) Random Forest

```
RandomForestClassifier, training accuracy: 1.000, validation accuracy: 0.956
Fitting Duration (seconds) :7.47s
Confusion Matrix :
[[  8   3   0   0   1   0   0   0   0   0   0   0   0   0   0]
 [  0 138   0   0   5   3   0   0   0   0   0   0   0   0   0]
 [  0   1 130   0   5   1   0   0   0   0   0   0   0   0   0]
 [  0   2   2  91   0   3   0   2   1   0   0   0   0   0   0]
 [  0   2   0   0 132   0   0   0   0   0   0   0   0   0   0]
 [  0   8   2   1   0 107   0   1   4   0   0   0   0   0   0]
 [  0   0   0   0   0   0  32   0   0   0   0   0   1   0   0]
 [  0   1   0   0   0   4   0  92   1   0   0   0   0   0   0]
 [  0   0   0   0   0   1   0   2 109   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  86   1   0   1   1   0]
 [  0   0   1   0   0   1   0   0   0   0 137   0   0   0   0]
 [  0   0   1   0   0   1   0   1   0   0   0  81   0   0   0]
 [  0   0   0   0   0   0   0   0   0   1   0   0 133   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   1   1 157   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  53]]
Cross-Validation-Scores: [0.9424589  0.94923132 0.94887379 0.93707544 0.94422596]
MAX Cross-Validation-Score: 0.9492313192706471
MIN Cross-Validation-Score: 0.9370754379692527
Cross-Validation-Accuracy: 0.944373 (+/- 0.01)
```

2) Support Vector Machine (SVM)

```
SVM, training accuracy: 1.000, testing accuracy: 0.985
Fitting Duration (seconds) :55.5704s
Confusion Matrix :
[[ 12   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0 144   1   0   0   0   0   0   1   0   0   0   0   0   0]
 [  0   2 134   0   1   0   0   0   0   0   0   0   0   0   0]
 [  0   1   1  98   0   0   0   1   0   0   0   0   0   0   0]
 [  0   0   1   0 133   0   0   0   0   0   0   0   0   0   0]
 [  0   2   1   0   0 118   0   0   1   0   0   0   1   0   0]
 [  0   0   0   0   0   0  32   0   0   0   0   0   1   0   0]
 [  0   0   3   0   0   0   0  95   0   0   0   0   0   0   0]
 [  0   0   0   0   0   1   0   0 111   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  88   0   0   0   1   0]
 [  0   0   0   0   0   1   0   0   0   0 138   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  84   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0 134   0   0]
 [  0   0   0   0   0   1   0   0   0   0   0   0   1 157   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  53]]
Cross-Validation-Scores: [0.97855611 0.98140865 0.98176618 0.98283876 0.9792635 ]
MAX Cross-Validation-Score: 0.9828387558097962
MIN Cross-Validation-Score: 0.9785561115082202
Cross-Validation-Accuracy: 0.980767 (+/- 0.00)
```

3) Neural Network (MLP)

```
MLP, training accuracy: 0.969, testing accuracy: 0.933
Fitting Duration (seconds) :49.8882s
Confusion Matrix :
[[  8   3   0   0   0   1   0   0   0   0   0   0   0   0   0]
 [  1 131  10   0   2   0   0   0   1   0   0   0   1   0   0]
 [  0   3 123   2   2   6   0   0   0   1   0   0   0   0   0]
 [  0   2   1  91   0   4   0   2   1   0   0   0   0   0   0]
 [  1   3   0   0 127   0   0   0   3   0   0   0   0   0   0]
 [  0   5   3   5   0 106   0   0   1   0   2   0   0   1   0]
 [  0   0   0   0   0   0  33   0   0   0   0   0   0   0   0]
 [  0   1   0   0   0   4   0  90   1   1   1   0   0   0   0]
 [  0   1   2   0   0   2   0   0 106   0   1   0   0   0   0]
 [  0   0   1   0   0   0   0   0   1  84   2   0   1   0   0]
 [  0   0   0   0   1   5   0   0   0   0 133   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   1  83   0   0   0]
 [  0   1   1   0   0   0   0   0   0   1   1   1 129   0   0]
 [  1   2   0   0   0   1   0   0   1   0   0   0   0 154   0]
 [  0   1   0   0   0   0   0   0   0   0   0   0   0   0  52]]
/Users/kaiyuan/opt/anaconda3/lib/python3.9/site-packages/sklearn/neural_network/_multilaye
ergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimizati
  warnings.warn(
/Users/kaiyuan/opt/anaconda3/lib/python3.9/site-packages/sklearn/neural_network/_multilaye
ergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimizati
  warnings.warn(
/Users/kaiyuan/opt/anaconda3/lib/python3.9/site-packages/sklearn/neural_network/_multilaye
ergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimizati
  warnings.warn(
/Users/kaiyuan/opt/anaconda3/lib/python3.9/site-packages/sklearn/neural_network/_multilaye
ergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimizati
  warnings.warn(
Cross-Validation-Scores: [0.93102216 0.93886307 0.93457276 0.93814802 0.92670719]
MAX Cross-Validation-Score: 0.938863067572399
MIN Cross-Validation-Score: 0.9267071862710047
Cross-Validation-Accuracy: 0.933863 (+/- 0.01)
```

Random Forest training accuracy :	1.00
Random Forest testing accuracy :	0.956
SVM training accuracy :	1.00
SVM testing accuracy :	0.985
MLP training accuracy :	0.969
MLP testing accuracy :	0.933

Random Forest Cross Validation Accuracy	0.944373 (+/- 0.01)
SVM Cross Validation Accuracy	0.980767 (+/- 0.00)
MLP Cross Validation Accuracy	0.933863 (+/- 0.01)

As we can see from above, by using Cross Validation Accuracy, we are able to choose the best and worst classifier from the 3.

The best classifier : Support Vector Machine with cross validation accuracy of 0.98
The worst classifier : Neural Network (MLP) with cross validation accuracy of 0.933

However, for the Neural Network (MLP), although the accuracy is smaller than Random Forest, for the multiclass problem , the Neural Network is stronger in distinguishing different labels. Also, the Neural Network has a relatively small size. Though the duration of the Neural Network is not as short as KNN, it has a better performance and requires less time than Random Forest.

What are the labels that are hard to be classified ?

I try to identify the labels that are hard to be classified looking inside the confusion matrix that we have output above. The diagonal elements in the confusion matrix is the number of labels given correct, other elements are the number of wrong labels i attached to the actual label j data where i is not equal to j.

With such knowledge, I can get to know that the labels that are hard to be classified are class 2 and class 3 because there is where the most misclassification happens.

How do you handle the testing data ?

According to the “Testing_program.ipynb”, it will proceed the testing data as the same way as “training_program.ipynb”. Firstly, read the testing data, do preprocessing, load image, load model, run the testProgram with correct directory and correct input. Lastly, the output will show the accuracy and confusion matrix.

Confusion & Discussion

In a nutshell, I believe I have successfully reached our goal of doing classification on such traffic sign dataset using 3 classifiers. Along the way, I have learnt many valuable lessons not only from lectures and tutorials, but have the experience of solving a real life problem using Machine Learning. For example, when it comes to large datasets and multiclass classification, especially for image classification problems, I initially believed that neural networks should be the best option. After finishing this project, I discovered that the neural network did not function as well as we had anticipated. Also, I know that each classifier also has their own advantages and disadvantages. Like Random Forest, it returns the output very fast even with huge datasets, but it has the problem of overfitting which results in low prediction accuracy. While SVM is effective in high dimensional spaces, it is sensitive to noise and time consuming when having large datasets.

Note that the number of distinct class labels in the given data is not equal, indicating some bias in the data. But because SVM, Random Forest, and Neural Network all exhibit excellent final accuracy, we assume that this attribute has no impact on categorization.

Further Improvement :
Firstly, we can consider using a sampling technique to lessen the impact of bias, but given the high accuracy, this approach appears to be not valuable because there isn't much room for advancement. There are various warnings while working on the classifier in the SVM and neural network. It appears that under specific conditions, both models were unable to converge. So, if there are other models with comparable performance, we shouldn't try to use them.