Report for CISC 3026 Machine Learning
Course Project
Traffic Sign Classification

Garfield KE SIYUN
DB927871

December 12, 2021

Introduction

This project aims to implements some classifiers to classify images of traffic signs into different classes.
A pre-classified dataset were given.

Implementation

The dataset

The dataset given included 15540 images(in *.ppm files) which were classified into 15 classes of traffic signs.
This dataset were separated into training and testing set randomly with test_size=0.2 (i.e. 20% of the dataset were taken as testing set). We set the same random seed [random_state=666] thus we can get same separate result in both trainer and tester.

Pre-process of input images

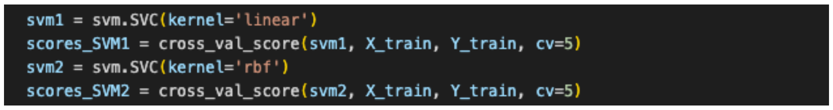
A *.csv file in each folder maintains the record of test images. We read the images and its class according to that record.
The traffic sign images have predefined valid area, to achieve higher efficiency, the edges can be cropped. That information was located in row [3]-[6] of *.csv file.
Then the test images were resized to 32*32 pixels. We tried to convert it to grayscale or even binary(B&W) images for better running speed(by applying np.array(img.convert('1'), 'i')). However we saw a decrease in accuracy (e.g. 0.98 to 0.79-0.87 for LinearSVM) while the running speed is not significantly improved, thus we kept the 3-channel color image. The final test set are simple flatten images, we tried other patterns like flatten local binary patterns but it has not very satisfying performance (0.78 for LinearSVM)

Design of the training problem

The trainer program separate the dataset and generate 3 classifiers for the testing program to use. We implemented SVM, logistic, and random forest classifiers.
Cross validation were used to choose best parameters of models.

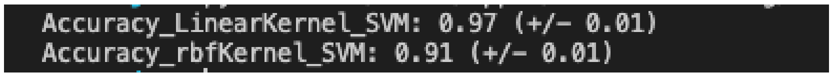
SVM

We implemented a simple linear SVM. We used cross validation(5-fold) to compare the performance of linear kernel and rbf(see code and result below), it turns out that the linear kernel has a better performance.



```
svm1 = svm.SVC(kernel='linear')
scores_SVM1 = cross_val_score(svm1, X_train, Y_train, cv=5)
svm2 = svm.SVC(kernel='rbf')
scores_SVM2 = cross_val_score(svm2, X_train, Y_train, cv=5)
```

Figure 1: SVM cross validation

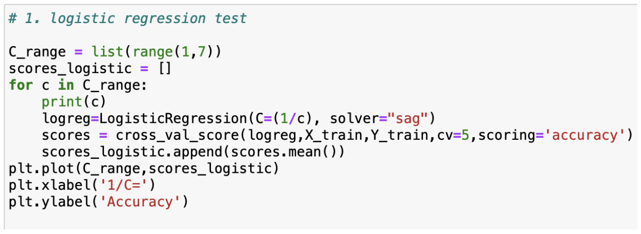


```
Accuracy_LinearKernel_SVM: 0.97 (+/- 0.01)
Accuracy_rbfKernel_SVM: 0.91 (+/- 0.01)
```

Figure 2: SVM cross validation result

Logistic regression

We choose a logistic regression function with parameter C adjustable. Documentations claim that a smaller C should lead to a better regularization.[1] Online resources indicate a high regularization might lower the performance[2] We test the C value from 1, 1/2, 1/3...to 1/7, and find the peak performance at C=1/4=0.25.



```
# 1. logistic regression test

C_range = list(range(1,7))
scores_logistic = []
for c in C_range:
    print(c)
    logreg=LogisticRegression(C=(1/c), solver="sag")
    scores = cross_val_score(logreg,X_train,Y_train,cv=5,scoring='accuracy')
    scores_logistic.append(scores.mean())
plt.plot(C_range,scores_logistic)
plt.xlabel('1/C=')
plt.ylabel('Accuracy')
```

Figure 3: test for best parameter C

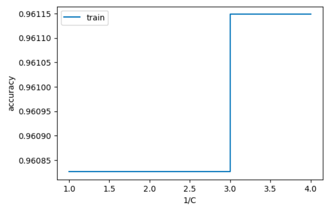


Figure 4:

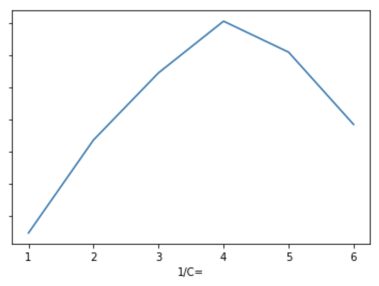
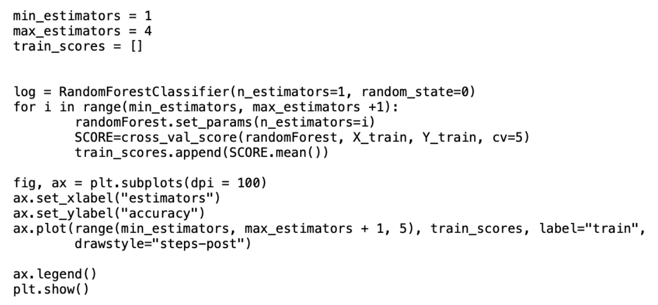


Figure 5:

Random forest

Our last classifier is random forest, the parameter n_estimators were tested using cross validation. And we choose 60 for final parameter. As we believe performance seems not increase greatly after 60 and the training time would be too long according to the figure.



```
min_estimators = 1
max_estimators = 4
train_scores = []


log = RandomForestClassifier(n_estimators=1, random_state=0)
for i in range(min_estimators, max_estimators +1):
        randomForest.set_params(n_estimators=i)
        SCORE=cross_val_score(randomForest, X_train, Y_train, cv=5)
        train_scores.append(SCORE.mean())

fig, ax = plt.subplots(dpi = 100)
ax.set_xlabel("estimators")
ax.set_ylabel("accuracy")
ax.plot(range(min_estimators, max_estimators + 1, 5), train_scores, label="train",
        drawstyle="steps-post")

ax.legend()
plt.show()
```

Figure 6: test for best estimator

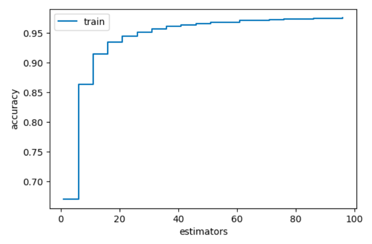


Figure 7: Best estimator

Save the model for future use

The model were saved by utilizing joblib. Three classifiers were saved in *.model files.

Design of the testing program

The testing program(tester.py) has a same procedure for reading image and pre-processing. We used a same random seed(666) to get a identical test set in trainer training program.
The testing program will analyse the accuracy of different classes. the result will be mentioned later.

Utilization & answers to given questions

The trainer are combined with all three classifiers and function to read images. The classifiers use best parameters found in cross validation (details can be found in Implementation section above). After fitting(training) the classifier, it would then save the model for tester to use.
The tester also includes function to read and process images. And it will separate the dataset into training and testing set in the same manner of the training set. This is to get the testing images that were not used to train classifiers.
It then read the models saved by trainer and predict the images in testing set. Then compare the real class and predicted class to calculate the accuracy. It outputs the accuracy of each class using each classifiers. Additionally it also generate an average accuracy table for 15 classed using three classifiers.

Answers to given questions 1-5

(1) Three classifiers used

In this research, as mentioned in the Implementation section, we constructed SVM, logistic regression, and random forest classifiers.

(2) The handling of multi-class problem

i) We used kernel=”linear” in the SVC for multi-class classification. scikit-learn documentations[3] reads it handle multiclass tasks according to a one-vs-one scheme.
ii) For logistic regression, the parameter solver= ”sag” handles multinomial losses in multiclass problems. Another parameter multi_class is set to default which capable for classifying into varied classes.
iii) For random Forest, we did not perform many special adjustment for support of multiclass tasks. Since that algorithm originally has multiclass problem support, by referring the nodes/leaves into different classes.

(3) How do you preprocess your data to get a better result?

Refer to what mentioned in previous section (Implementation/Pre-process of input images), we cropped the edges of images and resized them into 32*32, using recorded info in *.csv file of each classes. That improved the accuracy and slightly shorten the running time.

(4) How do you set the parameters of different classifiers?

As said in previous part (Design of the training problem) we used cross validation to set best classifier. Details and process had been explained in that subsection I will not repeat it here.
Generally we compare the performance (as measured by accuracy) of parameters through cross validation and chose the best one seen.

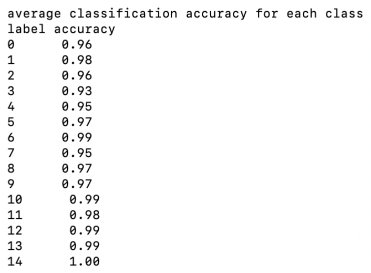
(5) How do you split the data into training and validation sets? (How about cross-validation?)

We firstly randomly separate the training and testing set randomly, as we wish to spare some untouched data for the tester program. 20% of images were reserved. Then we use 5-fold cross validation on the 80% training data while testing for best parameters, where the dataset were split into 5 parts for validations

Answers to given questions 6-7, comparison of different classifiers

(6) What is the performance of the three classifiers?

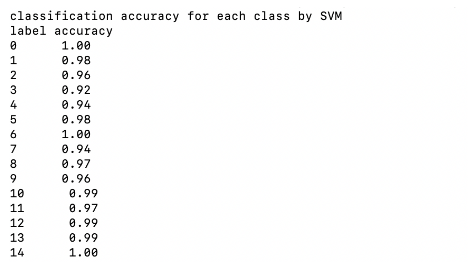
Here is the performance table mentioned above in the beginning of this section:
i) Overall performance: (classes among three classifiers)



```
average classification accuracy for each class
label accuracy
0      0.96
1      0.98
2      0.96
3      0.93
4      0.95
5      0.97
6      0.99
7      0.95
8      0.97
9      0.97
10      0.99
11      0.98
12      0.99
13      0.99
14      1.00
```

Figure 8: Overall performance

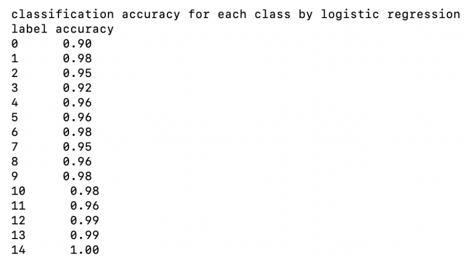
ii) Performance of SVM on each classes:



```
classification accuracy for each class by SVM
label accuracy
0      1.00
1      0.98
2      0.96
3      0.92
4      0.94
5      0.98
6      1.00
7      0.94
8      0.97
9      0.96
10      0.99
11      0.97
12      0.99
13      0.99
14      1.00
```

Figure 9: SVM

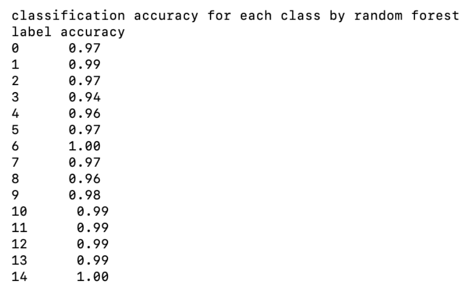
iii) Performance of logistic regression on each classes:



```
classification accuracy for each class by logistic regression
label accuracy
0      0.90
1      0.98
2      0.95
3      0.92
4      0.96
5      0.96
6      0.98
7      0.95
8      0.96
9      0.98
10      0.98
11      0.96
12      0.99
13      0.99
14      1.00
```

Figure 10: Logistic Regression

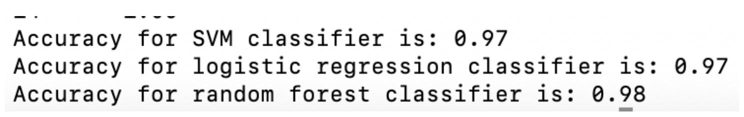
iv) Performance of random forest on each classes:



```
classification accuracy for each class by random forest
label accuracy
0      0.97
1      0.99
2      0.97
3      0.94
4      0.96
5      0.97
6      1.00
7      0.97
8      0.96
9      0.98
10      0.99
11      0.99
12      0.99
13      0.99
14      1.00
```

Figure 11: Random Forest

v) Overall performance of classifiers



```
Accuracy for SVM classifier is: 0.97
Accuracy for logistic regression classifier is: 0.97
Accuracy for random forest classifier is: 0.98
```

Figure 12: Performance of classifiers

(7) What are the labels that are hard to be classified?

Refer to the tables displayed in Question (6) above, we figure out class labelled 00003, 00004, and 00007 are having relatively low accuracy. These 3 were hard to be classified and class 3 is the most one. (We also directly examined the *.ppm images and it turns out the images in those 3 classes have many purely black photos which possibly make them hard to be classified)

Conclusion

This research briefly introduced the implementation of three classifiers, the method to adjust parameters, and the process of comparing the results. With limitation in time and computation power, we could not fully test and observe every influence of parameters/methods.
I would suggest possible future study could be focused on i) Whether a different, more rapid/efficient pattern for input X could be used instead of a flatten array? ii) How the quality of images affect the accuracy (i.e. the accuracy of label 3) and how to resolve it (high-robust method on quality?)?

Remarks

This project report was delivered by Ke Siyun (Garfield DB927871), the author's group partner Liu Pu(DB928051) also contributes to the research.

References

[1] sklearn.linear_model.LogisticRegression. (n.d.). Scikit-Learn. Retrieved December 12, 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[2] How does regularization parameter work in regularization? (n.d.). Stack Overflow. Retrieved December 12, 2021, from https://stackoverflow.com/questions/44742122/how-does-regularization-parameter-work-in-regularization/45420575

[3] sklearn.svm.SVC. (n.d.). Scikit-Learn. Retrieved December 12, 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html