

CSC345: Big Data and Machine Learning – Coursework 1

Name: Dominic McGinty | Student Number: 951939

Introduction

The objective of this coursework is to solve a popular problem in the machine learning world, to teach a computer how to identify what is in an image. In this particular example, we are provided with two small datasets (10000 images for training and 1000 images for testing, both containing 10 different categories) as well as labels that inform the machine learning algorithm what is in the image to originally train it and then test whether the answer it provides is correct.

My solution to this problem is to implement both a Support Vector Machine (SVM) as well as a Neural Network (NN) in order to solve which algorithm is better for these datasets. The reason that I have chosen to implement SVM is because of the speed the algorithm provides predictions compared to other methods while also being at a reasonable accuracy. As for NN, I have implemented this algorithm because of how popular they are today due to their extremely high potential for accuracy if fine-tuned and given a large dataset [1]. However, due to the size of the dataset, I predict SVM will provide a higher accuracy.

After experimenting with different types of kernels, my implementation to create a SVM will be a radial basis function (RBF) kernel with a C value of 9 as it provided the highest accuracy of 61% closely followed by the polynomial kernel with a C value of 5 at 60% accuracy. Other parameters such as degree, gamma, coef0 as well as linear and sigmoid kernels were also experimented with however any change to these values resulted in a lower overall accuracy. Experimenting with my NN, my implementation will contain 324 input nodes on a singular layer with an epochs value of 230. Notably, other parameters were changed such as optimizers, activations and number of layers but they all produced a lower accuracy.

Method

My method of implementation is to create 2 arrays to contain all the features of firstly the training images and then the testing images. These arrays will be filled by using a for loop containing the computeFeatures function given and then storing the outputs of this function inside the respective arrays. After this, both the training data and the testing data is created and normalized around the array containing all the features of the training images by using a standard scaler provided with sklearn.

To implement SVM, firstly I created variable to set the values for the SVC function used within sklearn.svm such as C, kernelType and gammaType to then create a variable to store the SVC function with these variables used as parameters. This variable is then fitted by loading the features of the training images and giving the algorithm what the correct labels are for these features. SVM then analyses this by creating lines around similar features based off what labels they have been given to then predict what the labels are for the array of features of testing images as they are compared to the SVMs dataset of plotted training features.

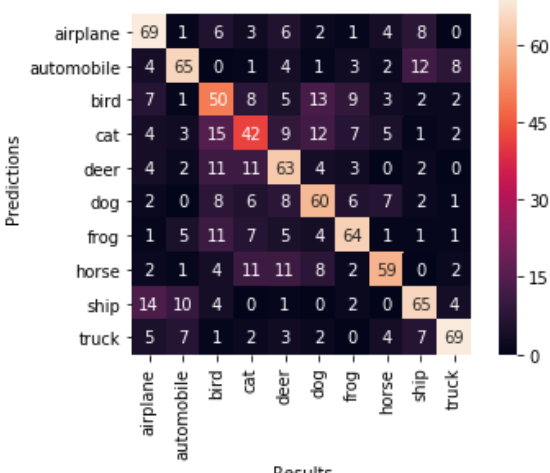
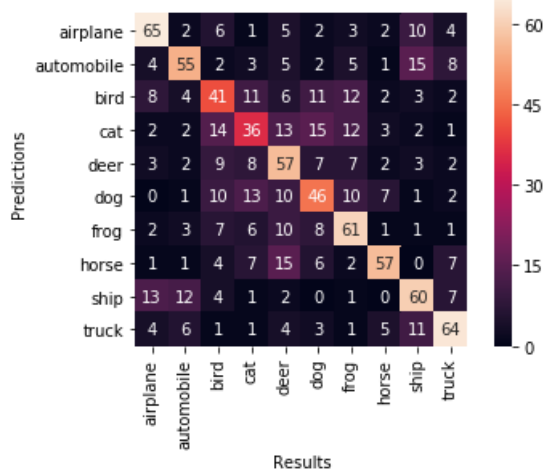
To implement NN, a variable containing the number of classes is created in order to make the number of output nodes be the same as the number of classes. Then I created a model containing keras.sequential() since only a small NN is needed, meaning it is not needed to

manually code a large amount of nodes. After this, the input layer is created to be the size of the number of features contained in all the training data, then 324 nodes are created in the 1 hidden layer with sigmoid activation. A singular output layer is also created with softmax activation and 10 nodes. Finally, the model is optimized using SGD and compiled to then fit training data to their corresponding labels and attempts to predict the labels for the testing data with an epochs value of 230.

Results

For experimental results related to SVM, the best accuracy I had found through trial and error was RBF kernel with a C value of 9 giving 61% accuracy, followed closely by poly kernel with a C value of 5 giving 60% accuracy. Changing gamma to different scales had no effect on the accuracy and changing it to a float value severely ruined the accuracy. The mean squared error of my final implementation gave 7% which is small meaning that the average data point is not too far away from the label line it was given.

For experimental results related to NN, the best accuracy of 54% which is slightly lower than SVM's accuracy. I had found this accuracy through trial and error by using a single hidden layer with 324 nodes (same number as the number of features in a single image) and an epochs value of 230. Changing the number of layers, the epochs value as well as differing the activation and optimizer all resulted in a lower accuracy. The mean squared error of my final implementation gave 8% which is slightly higher than SVM but still small meaning the labels assigned were not too far away from the data on average. NN was also notably much slower to predict the labels than SVM.

SVM	NN
Accuracy on testingData: 60.60%	Accuracy on testingData: 54.20%
Mean squared error on testingData: 7.23%	Mean squared error on testingData: 7.79%
<p>Confusion matrix</p>  <p>Predictions</p> <p>Results</p>	<p>Confusion matrix</p>  <p>Predictions</p> <p>Results</p>

Conclusion

In conclusion, SVM was faster to predict the labels and also more accurate than NN by 7% therefore my hypothesis was correct that SVM is superior in smaller datasets. If I was to improve my methods, I would change to a different NN to see if that would provide a better accuracy than sequential keras.

References

- [1] <https://pdfs.semanticscholar.org/4b09/e01cb21b26d4120077301b359d88aa206b28.pdf>