Heriot Watt University

Data Mining and Machine Learning

Coursework Two

Group 2: Report

Authors:

Gordon Rennie (H00155853), Samuel Haley (H00103552), Alexandros Mitrousis (H00307077), Andrew Nowlan (H00104104)

December 2, 2019

Contents

1 Data Pre-processing

The dataset is quite imbalanced, the biggest class (5) has 2160 images and the smallest one (7) only 240. Several methods were tried to balance the data sets. In every attempt the Cohen's kappa statistic and Mean Absolute Error (MAE) was used. The Kappa Statistic is a metric that compares an observed accuracy with an expected accuracy (Random Chance) and it is a reliable metric even if the classes are imbalanced. If the Kappa statistic is lower than the MAE then the algorithm cannot be considered precise or accurate (1). The balanced accuracy score, which shows the average accuracy per class, was also used as it is a good metric when we have to deal with an imbalanced dataset (2).
First, all classes were made equal to the number of images of the smallest class. Then, a middle ground solution, to make the bigger classes equal to 1320 images and to take the max number of the smaller ones and finally, using the full dataset. The results can be seen in the following table.

Dataset (number of images)	Kappa	MAE	Balanced Accuracy
All classes equal (2400)	0.556	1.12	53%
Middle ground solution (9660)	0.709	0.779	63%
Full dataset (12660)	0.759	0.634	68%

Table 1: Kappa vs MAE

The only case in which the kappa statistic was greater than the MAE was the last one when we used the full dataset. Additionally, the balanced accuracy score was also higher in that case. As a result the full dataset was used and any under-sampling effort was abandoned. Finally, to improve efficiency and reduce the number of network inputs (pixels in our case), we used the OpenCV's resize (3) method to reduce the number of pixels while still retaining most of the information contained in our images.

2 Variation in performance with size of the training and testing sets

2.1 Decision Trees

Examining the results of moving 30% and 70% from the training to test dataset (table 2) it can be seen that the results show the accuracy increased (accuracy is used throughout the report to refer to the mean recall value), even though the training data size decreased. This is not normally what would be expected, however, it is consistent with the test set being different from the training (see section 4.1), the model is correctly predicting the 30% of the data moved from the training set (increasing the accuracy) but is not predicting the original test data set well. When 70% is moved the accuracy does decrease slightly, being affected by the small training data set. This can also be seen in table 3 which shows the effect of reducing the training data size but not adding it to the test data set, as the training data size goes down the accuracy when predicting the testing set goes down as well.

Test train split	Recall	Precision	F1 score
original	0.74	0.74	0.64
30% moved to test	0.79	0.79	0.74
70% moved to test	0.76	0.77	0.73

Table 2: Effect of moving training data to test data

Train data size (% of original)	Recall	Precision	F1 score
90	0.75	0.74	0.64
70	0.71	0.71	0.63
50	0.72	0.72	0.61
30	0.71	0.70	0.61
10	0.64	0.64	0.53
5	0.53	0.53	0.43
1	0.46	0.47	0.41

Table 3: Effect of reducing the training data size

2.2 Neural Networks

When the results for the neural network are examined (see GitHub repository) it was found that they mirrored those found with the decision trees.

3 Variation in performance with change in the learning paradigm

For all the different learning paradigms it was ensured that they were all trained and tested on the same data sets to allow them to be compared. Table 4 shows the best accuracy (average recall across all classes) results obtained for each model. By examining the results in table 4, different learning paradigms can be compared. When training on the training data and testing on the test data, the decision trees underperforms in comparison to the other methods. This is most likely due to the decision tree overfitting on the training data. The results in general for decision trees show that they are very prone to overfitting. Neural networks produced the highest accuracy of 93%, however, this is only just above the 91% achieved by the logistic regression (4) model, and it required much more computer power to achieve (the Neural Network takes over two minutes to train, while the Logistic Regression is almost instant). Random forest also produces a high result of 88% accuracy. In terms of time taken it lands almost in the middle of the neural network and logistic regression.

Method	Decision tree	Random forest	Logistic Regression	Neural network
Test/Train	0.75	0.88	0.91	0.93
10-fold validation (train)	0.89	-	0.96	0.965

Table 4: Accuracy for different paradigms (best model)

One factor that must be considered is that the neural networks produced are effectively "black boxes", it is difficult to discover how it comes to an answer. This is unlike decision trees (including random forests), in which the tree can be fully examined, even giving the option of manually taking an image of a sign through the tree to classify it, seeing how the model works.

4 Variation in performance with varying learning parameters in Decision Trees

SKLearn allows multiple parameters to be modified, an explanation of those that were examined can be seen below:
max_depth: the maximum number of layers in the tree.
min_samples_split: the minimum number of samples required to split a node.
min_samples_leaf: the minimum number of samples required in a leaf node.
min_impurity_decrease: the impurity value must decrease more than this value if a node is to split. Impurity is a measure of the number of different sample classes in a given node, if all samples are the same class then impurity is zero.
min_weight_fraction_leaf: the minimum weighted fraction of the sum total of required to be at a leaf node. If this value was zero, then all samples must be of the same class in a leaf node.

4.1 Decision Trees

To get a base line for how a tree classifier might perform a ten-fold validation was run on the training data, with no limits to the fitting. This produced a mean accuracy of 0.89 and a mean f1 score of 0.88, with the f1 score being similar for all classes, the lowest being 0.79 for class 7. However, the tree has some difficulty differentiating between classes 0 and 1, similar to the results in the Bayesian analyses (course work 1).

In this section all the results are from the model trained on the training data and used to predict first the test data then the training data (checking for overfitting).
First a base case was run with no limits applied to restrict overfitting, results can be seen in table 5. These results show that the model has overfitted as it explains all the training data (accuracy 100%) but only 74% of the test data.
To see how each parameter of the tree classifier affected how the model overfits, various values for each were tried. Table 5 shows the best results (highest accuracy and f1 score) for each parameter.

		On Test data			On Train data
Model	**Best value**	**Average Recall**	**Average Precision**	**F1 score**	**Average Recall**
Base	-	0.74	0.74	0.64	1.0
min_samples_split	0.001	0.73	0.73	0.63	0.97
max_depth	11	0.75	0.74	0.64	0.94
min_samples_leaf	3	0.75	0.74	0.64	0.97
min_impurity_decrease	0.0004	0.74	0.75	0.64	0.94
min_weight_fraction_leaf	0.0004	0.74	0.74	0.64	0.94
All	All above	0.74	0.75	0.65	0.92

Table 5: Results from trying to prevent overfitting through different means

It can be seen that the parameters had little effect on the accuracy when predicting on the test data. However, examining the trees (see "decision tree output" folder in GitHub depository) when all the parameters were set as shown in table 5, the tree was much smaller, but still obtaining the same accuracy and f1 score. Examining accuracy on train data also shows that overfitting was reduced as the accuracy is no longer 100%.
It was noted that the results from the tenfold validation where significantly better than the best obtained in table 5. To try and explain this the raw data was examined, it was found that the average pixel value in the test set was approximately 4.5 higher than the training data. This perhaps suggests that the images were taken at different times of day when the signs will be darker or lighter. This explains the difficulty of predicting the test set as it is slightly different from the training data (this is examined further in the research section).

4.2 Random Forests

A random forest classifier was used with 100 trees, images were selected for each tree by sampling with replacement. The random forest was trained on the full training data set and then used to predict the test data. This obtained an accuracy of 0.88 and an f1 score of 0.79. This result is noticeably higher than the results from the single decision tree (table 5).

4.3 User Classifier

A user classifier was also carried out in Weka. This was done on the top 2 attributes from Course work 1. It was found that the user classifier gave a good understanding of how overfitting occurs as to be able to separate out each class, careful drawing of polygons around each class was required to generate the classifier rules. The results obtained from using the user classifier gave an accuracy of 0.55, much lower than any other method used.

5 Variation in performance with varying learning parameters in Neural Networks

We chose our base architecture for our Multi Layer Perception experiments to consist of two hidden layers, each containing 512 neurons and an output layer with 10 neurons (one for each class). The two hidden layers implemented the ReLU activation function while the output layer implemented softmax. This choice of architecture, using 20 Epochs, has been shown to perform well when applied to a similar type of image classification problems, in particular digit recognition as seen in (5). Furthermore, since the Keras library does not provide it's own functionality for ten fold cross validation, we used SKLearn's StratifiedKFold(6) method to split the training data into ten folds of equal proportions. We then trained 10 independent, identical network structures on each fold and averaged the accuracy result.

All experimental models were derived form the base model. In the interest of time and practicality, 10-fold cross validation was not carried out for each experimental variation made to a selected parameter. Instead model performance was measured against the supplied test set. The base model achieves a high training accuracy of 0.93. The average precision and recall of the model is high and reflected in the high f1-score of 0.93. In essence the base model is already high quality and the following sections will explore what effect changing various parameters has.

Table: 6 below shows the results of three experiments. The parameters altered were: Layers - doubled to 4 and halved to 1; Epochs - doubled to 40 and halved to 10; Validation Threshold - set to 1% and 0.01%.

Parameter	Change	Recall	Precision	F1 Score
Base Model	None	0.83	0.89	0.85
Layers	Halved	0.9	0.89	0.88
	Doubled	0.85	0.9	0.86
Epochs	Halved	0.88	0.89	0.88
	Doubled	0.9	0.91	0.9
Validation Threshold	0.01%	0.85	0.87	0.84
	1%	0.81	0.85	0.81

Table 6: Change in Model Performance With Parameter Change

As can be seen from the table all models showed similar results despite the parameter changes. Of particular note however is the effect on training time the parameter change had. When the number of layers was halved the training time dropped by around a third. Further, when the Epochs were halved the time also halved. Suggesting that faster to train smaller models may be just as effective as larger ones. The largest change came from the validation accuracy parameter. Both models ceased training after very few Epochs (Small model 5 and Large 3). This meant that both models trained much faster than any other parameter change but maintained similar levels of accuracy.

5.1 Altering the Number of Neurons

It has been demonstrated (7), that as the number of connections in a network increases, a natural tenancy for overfitting also increases. In order to investigate how the number of neurons affect our own network's classification ability, we ran an experiment in which the number of neurons in both hidden layers of our bases model were altered from 256 to 3048. We observed that the training accuracy initially increased with 512 neurons and thereafter, continued to decline with a shallow gradient. The validation accuracy oscillated as the number of neurons increased, though a peak was observed when the number of neurons was set to 1024. From these results, we are unable draw any definitive collisions relating to the affect the number of neurons have on overfitting in our network. We also observed that increasing the number of neurons in each layer beyond 3000 is very computationally expensive and impracticable to run. With better processing power, we would ideally run further experiments involving a significantly larger number neurons with the hope of obtaining more conclusive results.

5.2 Altering the Learning Rate

Learning rate is a hyperparameter that determines how much the network weights are adjusted with respect to the loss gradient (8). To investigate how changes in this parameter affect the results obtained from our model, we incrementally varied the value of this parameter from a default value of 0.001 to 0.02 and observed an overall decrease in both training and test accuracy. A very shape decline was observed once this value exceeded 0.015. The default value of 0.01 provided the best accuracy for both the test and training set. As described in (8), large learning rate can cause the minimum to be overshot causing failure to converge, or even divergence. Such behaviour is evident in our results.

5.3 Altering the Momentum

We chose to use the Adam (9) optimizer in Keras. As such, we had to experiments with an alternative parameter to momentum. Adam, does however, keep an exponentially decaying average of past gradients similar to momentum (10), denoted m_t and given by:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{1}$$

Keras facilitates the modification of β_1 and this enabled us to conduct an experiment in which we investigated values between 0.5 and 1.0. A modest increase in validation accuracy was observed for 0.7 and the accuracy of both test and training models diminish completely when equal to 1.0. As seen from equation (1), this value of β_1 causes the gradient term to become trivial.

6 Variation in performance according to different metrics

Recall (also called TP rate): Is the number of images correctly identified as belonging to a class out of the total actual images of this class.

Precision: Is the number of images correctly identified as belonging to a class out of the total number of images that predicted to belong in this class.

Both these metrics are helpful in multiclass classification because **high Recall** and **high Precision** mean that a class is perfectly handled by the model so we can distinguish which classes are well handled and where our model needs to be better adjusted.

F-Measure: Is the effectiveness of prediction when the same importance is given to recall and precision. This is probably the most important metric in our case because we are using an imbalanced dataset in a multiclass problem and as mentioned above, we must consider both recall and precision in order to measure our models. Other metrics, like accuracy, could lead to wrong assumptions. This is because high accuracy can still exist when underrepresented classes are commonly mistaken. Meaning that although accuracy may be high, the model may still be poor at identifying edge cases.

In a multiclass classifier **Recall, Precision** and **F-Measure** can be calculated in two ways, as macro-level and as micro(weighted)-level. The former gives equal weight in each class, but on the other hand, the latter gives more influence on the bigger classes. As we can see in all our experiments the macro-levels are lower than the micro-levels for these 3 metrics. That is because we have an imbalanced dataset and the smaller classes are not performing as good as the bigger ones.

FP rate: Is the number of images wrongly identified as belonging to one class out of the total number of images that are not belonging in this class (e.g. an image that is not a stop sign incorrectly labelled as a stop sign). This metric is more suitable for binary classification and is not particularly helpful in our case.

ROC Area: This metric tells us how capable the model is in distinguishing between classes. When the area under the curve(AUC) is near to 1 that means we have a good measure of separability and when it is near to 0, we have a bad one. In a multiclass classification, we must use a one vs all methodology so, we need N number of ROC curves where N is the number of classes. It is a good metric if we want to see how well our model can separate one specific class from all the others.

Examining the results obtained, it can be seen that the f1 score, precision and recall all increased and decreased with each other. Therefore, for this data set they can be used interchangeably and would change little in the final report. If the results obtained for the TP, FP are examined they are far more varied, and are a harder metric to understand due to the numbers being a measure of the binary performance, ie. one class Vs all others.
The ROC curve, as stated above, also shows the binary classification of each class. The issue with the

ROC curve being used for non-binary classification is that you could end up with some images not being classified, if the threshold confidence is set to 90% for each class, some images may not have that level of confidence for any class when put through the model, and hence not being classified. For sign recognition in a autonomous car, it is clearly not a good thing if the car ignores a stop sign because it was only 88% confident in the classification.

7 Research Question: What is the Best Approach to ensure Models can Predict Road Signs at Different Times of Day?

As was noted in the tree classifier section, it was found that the test data set had an average pixel offset of around 4.5. This offset could occur due to photos being taken at different times of day. The aim here was to find a way to account for this offset in our models.
The first solution considered was to standardise the data sets (for each picture, subtract the average pixel value from each pixel and divide by the standard deviation). After both the test and training data sets had been standardised a random forest and a decision tree classifier were trained and then used to predict (table 7). For both models the increase was large, showing that standardisation is worthwhile.

Model	Data type	Recall	Precision	F1 score
Random forest	Original data	0.89	0.88	0.88
	Standardised data	0.92	0.93	0.92
Decision tree classifier	Original data	0.74	0.74	0.64
	Standardised data	0.80	0.81	0.80

Table 7: Results from standardising the data sets Vs original data set

We also investigated if a reduction in overfitting may enable our models to better predict non-normalised images. To do this we took the supplied test data set and manipulated it to create a 'day' data set, to do this we increased the value of each pixel by a random number between 10 and 15 as this will simulate a road sign during a sunny period of the day.

We then created a new model which utilised dropout to reduce overfitting. Dropout is a simple way to reduce overfitting (11) which involves skipping over random neurons during each training Epoch. The idea is that by skipping random neurons the network is prevented from building one 'perfect' representation of the training data. Rather, many different sub-representations will be present in the network. This results in networks which usually do not report as high a training accuracy as those without dropout but those models with dropout do not suffer from as serious overfitting.
Dropout removes a fixed number of random neurons during each training Epoch. This means that usually a larger network is better placed to implement dropout as it allows enough space for the construction of these sub-networks. As such for this experiment we modified the training model used in section 5 and trained it on the original data set with a dropout rate of 20% on every layer. We then tested its ability to predict the day data set.

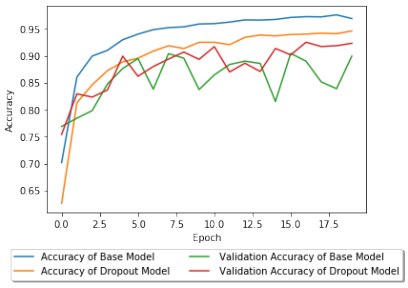

Figure 1: Dropout Effect

Figure: 1 shows a comparison of model accuracy and validation accuracy over the course of 20 training Epochs for our base model and dropout model (when using the 'day' data set for validation). It clearly shows the effect of using dropout. The accuracy of the dropout model is lower but its validation accuracy is much closer to the model accuracy showing that dropout has effectively reduced overfitting and resulted in a model which is capable of predicting images which are very different to the original training set.

Furthermore, what is perhaps most useful about this model is that it's reported accuracy is closer to the validation accuracy than any other neural network model. This is particularly useful in industry, as a high accuracy may sound better but an accuracy that can be trusted on unseen data is better.

8 GitHub Link

https://github.com/anowlan123/DMML_CW2_Group2

9 Contributions

We feel that everyone in the group contributed equally.

References

[1] J. E. T. Akinsola, "Supervised machine learning algorithms: Classification and comparison," *International Journal of Computer Trends and Technology (IJCTT)*, vol. 48, pp. 128 – 138, 06 2017.

[2] K. H. Brodersen, C. S. Ong, K. E. Stephan, and J. M. Buhmann, "The balanced accuracy and its posterior distribution," in *2010 20th International Conference on Pattern Recognition*. IEEE, 2010, pp. 3121–3124.

[3] OpenCV. (2019) Resize method. https://docs.opencv.org/trunk/da/d54/group__imgproc__transform.html#ga47a974309e9102f5f08231edc7e7529d.

[4] Scikit-Learn. (2019) Logisticregression. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html,

[5] LinkedInLearning. (2018) Neural networks and convolutional neural networks essential training. https://www.linkedin.com/learning/neural-networks-and-convolutional-neural-networks-essential-training?trk=search-result_learning_card_title&upsellOrderOrigin=default_guest_learning,

[6] Scikit-learn. (2011) Stratifiedkfold. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html,

[7] N. Buduma. (2015) Preventing overfitting in neural networks. https://www.kdnuggets.com/2015/04/preventing-overfitting-neural-networks.html,

[8] H. Zulkifli. (2018) Understanding learning rates and how it improves performance in deep learning. https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10,

[9] Keras. (2019) Optimizers. https://keras.io/optimizers/,.

[10] S. Ruder. (2019) An overview of gradient descent optimization algorithms. https://ruder.io/optimizing-gradient-descent/index.html#adam,https://www.overleaf.com/1485449344ktgkdsmwbfhy.

[11] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1930 – 1958, 06 2014.