Student Number: 246518

1. Introduction

The process of separating items into distinct groups, often referred to as classes, is a fundamental part of machine learning. By using machine learning systems, otherwise hidden connections between data and their corresponding classes can be unveiled, enabling accurate predictions on previously unseen data.

Within this report, I aim to develop and optimise a machine learning system capable of effectively classifying extracted features from a collection of images as either 'happy' or 'sad'. This task will involve implementing several machine learning steps, including data imputation, pre-processing, data splitting, and the use of a hyperparameter-optimised machine learning model.

2. Method Experiments

My intent in this report is to make as few changes to the algorithms & tools used as possible, but instead to try and find a reliably well performing set of parameters, hyperparameters and 'alternative steps' to work with.

Accuracy will be used as a measurement of fitness for all experiments, with the best combination of experiment results being used for the final unseen, unlabelled dataset used for marking.

A 'baseline' accuracy has been calculated as a point of comparison for the following experiments, which uses default values provided with the modules for all the following steps:

1. Impute any missing data.
Using the largest available dataset for training to reduce over-fitting and increase my model's robustness is a priority when starting. 83% of the provided data had missing features, so some form of imputation was needed.

K-Nearest-Neighbour Imputing will be used for data imputation as it is more effective than just finding the commonly used row average, or just filling the missing values with zeros. [2]

2. Normalise all features.
Some ML algorithms, like Support Vector Machines (SVMs), which I use throughout this report, are not scale invariant. Therefore, it is best to normalise all data points to ensure that each feature is treated fairly while training & predicting. [1]
All rows will be normalised to have a mean of 0 and a Standard Deviation of 1.

3. Split the data into a training set and testing set.
Data spitting is necessary to ensure that the labelled data provided can be used to both train and test any classifiers created. A Train-Test split of 8:2 is very commonly used [1] and will be used by default within experiments.

4. Train an SVM & Create predictions on testing data.
A Support Vector Machine will be used to make predictions due to its effectiveness in high-dimensional spaces and its versatility due to the multiple kernel functions available. [1].
Predictions will then be made from the testing portion of the data which was split in the previous step.

5. Calculate the accuracy of predictions.
Once a set of predictions has been made, the accuracy of these predictions will be measured and used as the method's performance metric.

All these steps (or variations of them) will be carried out with several different parameters, hyperparameters and added steps to find an optimal method which yields the highest accuracy.

Using the default values for these steps (all of which are sourced from SciKit-Learn) yielded an accuracy of 0.742. This is our baseline value.

2.1 Data Imputation

As well as the default Neighbourhood size of 5, I tried neighbourhood sizes ranging from 3 to 21, in intervals of 2.

All these neighbourhood sizes were tested 20 times to calculate a mean and ensure a fair test. These mean accuracies of these tests can be seen below.

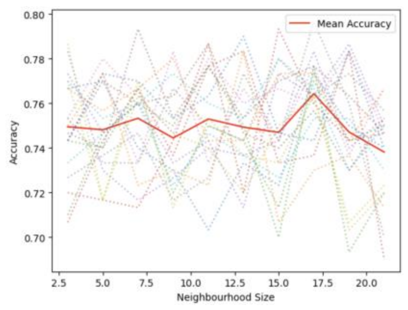

Figure 1: Mean accuracies of different neighbourhood sizes.

While the data is quite noisy, a clear spike in accuracy can be seen with a neighbourhood size of 17.

2.2 Data Normalisation

Data normalisation will be performed in all experiments due to SVMs requiring normalisation.

2.3. Data Splitting

I experimented with two different data splitting methods, a standard 8:2 test-train split, and a 'weighted split'.

This weighted split attempted to incorporate the confidence values into the train-test split by increasing the proportion of 'confident' images (those labelled with a confidence value of 1) within the training data, therefore decreasing the number of possible outliers in the training set.

This was done under the assumption that the confidence value of each set of features could be thought of as an inverse likelihood of a feature set being misclassified during labelling.

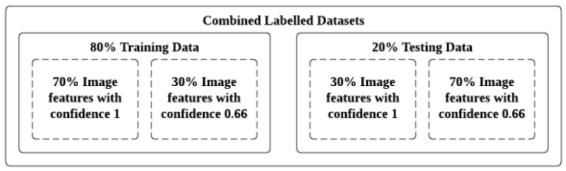
Combined Labelled Datasets
80% Training Data
70% Image features with confidence 1
30% Image features with confidence 0.66
20% Testing Data
30% Image features with confidence 1
70% Image features with confidence 0.66

Figure 2: A visualisation of the dataset's 'weighted split'.

This weighted split ended up performing worse-off than the standard splitting, with an accuracy of 0.743 as opposed to 0.763, so it was not used.

2.4 SVM Training & Predicting

A Grid search was carried out to find the optimal parameters for the SVM.

Before this however, tests found that Linear and Polynomial kernels performed worse-off than RBF and sigmoid kernels. Due to this, only RBF and sigmoid kernels were tested due to the computationally expensive nature of Grid searches.

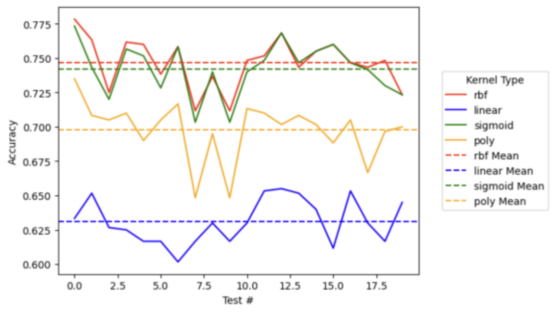

Figure 3: Accuracies of different SVM Kernel Types.

The parameters explored in the Grid search were as follows:
'C' values of 0.1, 1, 10, 25, 50 and 100.
Sigmoid and RBF Kernels.
Gamma values of: 'scale, auto, 0.01, 0.1, 0.5, 1 and 1.5.

This grid search found the optimal parameters were a C value of 1, a gamma value of 'scale' and a sigmoid kernel, yielding an accuracy of 0.763.

3. Results

Through all the experiments carried out, I believe to have found an optimal set of parameters, hyperparameters and steps for my classifier, achieving an accuracy of 0.763.

These steps can be seen below and will be applied to the testing data supplied for the coursework's predictions.

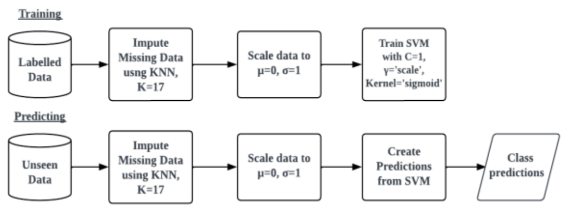
Training
Labelled Data
Impute Missing Data usng KNN, K=17
Scale data to μ=0, σ=1
Train SVM with C=1, γ='scale', Kernel='sigmoid'
Predicting
Unseen Data
Impute Missing Data using KNN, K=17
Scale data to μ=0, σ=1
Create Predictions from SVM
Class predictions

Figure 4: Process flowchart for classifier training and labelling of unseen data.

4. Discussion

While these results aren't amazing, I am still pleased with my experiments and my classifier's accuracy.

However, I feel as if the only (hyper)parameter which made a noticeable difference from the default values was the kernel used in my SVM.

SciKit-Learn's developers seem to have already optimised most of their ML library parameters.

Additionally, most other accuracy variation within my experiments looks as if it is due to random chance, and I predict that an optimal/suboptimal train-test split caused the accuracy variation between experiments.

If I were to carry this out again, I would perhaps try to experiment with more steps such as Principal Component Analysis, splitting the dataset's GIST & CNN Features, and perhaps tried other ML models.

5. Conclusion

To Conclude, this report has detailed the development and optimisation of a simple Classifier to class extracted image features as either 'happy' or 'sad' through the use of pre-processing techniques, data splitting and a machine learning model.

6. References

[1] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B. a., Blondel, M., . . . Brucher. (2011). Scikit-learn: Machine Learning in {P}ython. *Journal of Machine Learning Research*, *12*, 825-2830.

[2] Troyanskaya, O. a. (2001). Missing value estimation methods for DNA microarrays. *Bioinformatics, 17*(6).