Final Project Report – MNIST

By Mark Peters

Introduction

Background

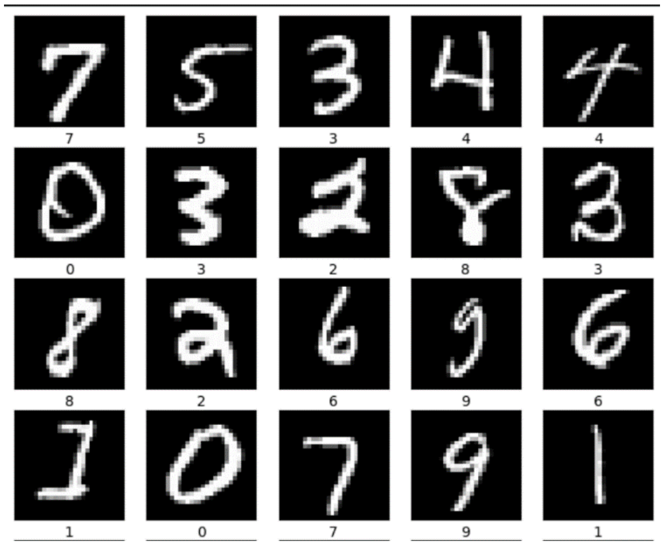
The MNIST dataset is an example a labelled dataset whose observations need to be categorised under 10 different labels – in this case, under each digit from 0 to 9. The aim of this project is to run the data through different categorising models to find the one that predicts the labels most accurately. As this is a problem of classification, the models to be used must be ones applicable in this case. The best model should be able to predict, to a higher degree, the nature of new data in the form of handwritten numerals.

Research Questions

The question of how best to evaluate the results of dataset modelling relies on the nature of the data and the focus you want the model to have. For example, if the point is to predict the medical outcomes of patients, then 'False Positives' need to be avoided at all costs (the example of being right 99% of the time is moot if only 1% of the population contract a certain disease to begin with).

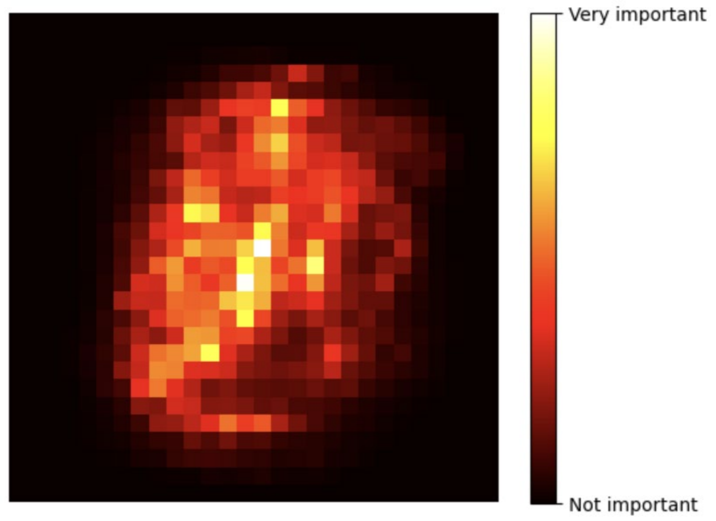
In the case of the MNIST dataset, **accuracy** is more important than precision as the cost of 'false negatives' and 'false positives' are roughly the same. Although I take that to be the most important metric, I will include other results from my models as they may indicate other tests, or methods of research, that may be worth carrying out afterwards.

EDA (Exploratory Data Analysis)



The MNIST dataset is composed of a series of handwritten digits with **784 features**. The full set contains **60,000 training examples** and **10,000 test examples**, formed from the combination of handwriting from two groups; group 1 (SD-3) was composed of employees of the Census Bureau in the United States, the second group (SD-1) was handwriting collected from high school students, which may explain why that set isn't as 'clean' as the former. This makes it even more important that the sampling is random.

In terms of pre-processing, not much needs to be done. This is a clean and much worked dataset. The figures are of the same size and there are no missing values. As the total number of each written digit is not the same for each figure, scaling should be carried out (especially in the case of SVM's).



A hot grid of the grid squares most used for each digit doesn't suggest much about how the data should be modelled, but it does show the importance of the central squares and how that importance lessens as you radiate out. Overall, the deportment of squares is quite democratic.

(Further details and explanations can be found within the accompanying Jupyter notebook)

THEORY: METHOD & MODELLING

I split the data into three groups – train, val, and test – mainly because I hadn't seen it done before and I thought it would make for more interesting results, as I had decided to run the dataset through several models.

The models I used in this project were:

Linear SVC: A linear support vector machine which learns the linear decision boundaries between classes in the training data by maximizing the margin between the closest data points of different classes. It then uses a hinge loss function to train the model, which penalizes misclassifications with a linear function.

SVC: I also used SVC which is a non-linear SVM. It tends to be slower for larger datasets but more flexible (due to its non-linear nature).

Linear SVC uses the Stochastic Gradient Descent (SGD) algorithm which is generally more effective with larger datasets. It's not clear how large is 'large' but the SVC classifier worked much better on the MNIST dataset, with the non-linear element likely being key.

SGD Classifier: SGD is an acronym for 'Stochastic Gradient Descent' which is the same algorithm that Linear SVC uses. It shares its linear quality also. It is flexible and efficient, and it works by trying 'to find the optimal hyperplane that separates the classes in the input feature space', which relates to matrix transformations and multi-dimensional space (which is outside the scope of this project).

Logistic Regression: Logistic regression is a supervised learning algorithm that can be used to classify data into categories, or classes, by predicting the probability that an observation falls into a particular class based on its features. It can be used for more than two categories, though it is primarily used for binary problems.

That is why for this project, it was at this point that I created a new target group of just the digit ‘2’. Since LR allows for probability points it was possible to create a ROC graph for the projected accuracy of the model. After some hyperparameter tuning I created a second graph to investigate whether tuning improved the accuracy of the model. It did not do so in my modelling, though with more experimentation better results may well be attained.

Decision Tree: Decision trees can be used for both linear and classification problems. They work by partitioning data into groups based on the components of target values. The goal of the algorithm is to maximise the distance between target groups, or to minimise the variance within the target group. Due to this specificity, there is a danger of over-fitting the data, which is why it is important to tune the hyperparameters so that the model can respond accurately to new data.

Random Forest: The Random Forest model also supports ROC graphs (which are printed in the notebook). As the name might suggest, RF is an ensemble of Decision Trees. It has all the hyperparameters of DT, as well as the added parameters that come with ‘BaggingClassifier’ (which uses the same algorithm but trains the model on different subsets of the training set).

As mentioned earlier, due to the risk of overfitting it is important to tweak the hyperparameters, so the model adapts more responsively to new data. That was what I did in the case of my model, though due to my inexperience I added too many parameter options to my first ‘GridSearchCV’ test and had to wait 1060 minutes for it to run! I won’t make that mistake again.

K-Nearest Neighbours (KNN): This is a simple, but powerful algorithm which considers the positional relationship between data points and assigns observations to groups based on how many ‘neighbours’ it has, dependant on the parameters given to the algorithm.

In the case of the MNIST set, after running a gridsearch the optimal number of ‘neighbours’ was calculated at 3. This low number makes the model more sensitive to noise and outliers, but in the case of MNIST that seemed to be the correct approach to take.

RESULTS & ANALYSIS

RESULTS

A fuller list of all the results can be found within the Jupyter notebook. Here, I have extracted the accuracy score as well as the confusion matrix for each model based on its processing of the untouched ‘test’ set, that had been set aside at the beginning of the project. The reason for this is explained in more detail under the Research section, but essentially accuracy matters more as the stakes of false positives or false negatives are not very high.

Accuracy:

General Modelling of MNIST:

Model	Accuracy
Linear SVC	0.869
SVC	0.96

Modelling to predict for '2':

Model	Accuracy
Logistic Regression	0.979
Logistic Regression (after tuning)	0.977
SGDClassifier	0.957
Decision Tree	0.97
Random Forest	0.989
K Nearest Neighbour	0.976

Confusion Matrix:

The matrix for **LinearSVC** is in the notebook and it contains the matrix for all the digits in the dataset.

SGDClassifier	**Predicted Negative**	**Predicted Positive**
Actual Negative	12251	369
Actual Positive	231	1149

Logistic Regression	**Predicted Negative**	**Predicted Positive**
Actual Negative	12514	106
Actual Positive	195	1185

LR (after tuning)	**Predicted Negative**	**Predicted Positive**
Actual Negative	12578	42
Actual Positive	279	1101

Decision Tree	**Predicted Negative**	**Predicted Positive**
Actual Negative	12403	217
Actual Positive	199	1181

Random Forest Classifier	Predicted Negative	Predicted Positive
Actual Negative	12604	16
Actual Positive	141	1239

K-Nearest Neighbour	Predicted Negative	Predicted Positive
Actual Negative	12545	75
Actual Positive	266	1114

Analysis

After fitting all the models to the training and evaluating sets, adjusting hyperparameters, and then fitting the resultant models to the saved test data, we end up with a set of accuracy scores that are roughly similar. Though since you would expect a 90% return on random selection, small percentage differences are significant in this case.

The clear winner of my testing was a **Random Forest model** with the below hyperparameters and a cross validation of **5 folds**:

{'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}.

It was the most accurate both in terms of identifying which digits weren't a '2' and which were. In the case of every model, identifying positive examples was the more challenging. The 'bagging' (same algorithm for every predictor but with different subsets) of Random Forest aggregates all the predictors into the *statistical mode,* which averages the bias of all predictors so that it is roughly equivalent to each individual example. What is important is that the ensemble testing gives a lower variance than a single predictor trained on the original dataset.

Conclusion

Using accuracy score as our key evaluating metric, the Random Forest model (with the above hyperparameters) was the most effective in identifying handwritten versions of the number '2' as being the number '2', with an accuracy score of **0.989**.

Potential Areas of Future Study

Where I would next like to take this study next would be into an area covered by a future course – the Neural Networks of Deep Learning. What I have read about the subject is that Convolutional Neural Networks (CNN's) are able to form, and then learn to identify, hierarchical representations of input data, which allows them to effectively capture features of the images that are important for classification. Such image delineating and structuring would be extremely effective in the case of the MNIST dataset.