COMP5318 Assignment 1: Report

490182143

Introduction

The aim of this study is to establish a comparison between three prevalent classification algorithms that exist today and report the findings of the best classifier for the correct prediction of the fashion-MNIST dataset, which contains images of articles of clothing.

This study is important as it highlights the differences in accuracies between classification algorithms, as well as shows competent ways of pre-processing data using PCA and normalisation, as well as hyper-parameter tuning using grid search for finding optimal parameters for each of the three classifiers executed in the study.

Methods

Pre-Processing

There are two pre-processing techniques employed in this study: normalisation (or min-max scaling) and PCA.

The normalisation function is denoted as:
$x' = \frac{x - min(x)}{max(x) - min(x)}$, where the aim is to assign the minimum value as a 0 and the maximum value in the data as 1, with all remaining values being on a scale of 0 and 1.

Principal Component Analysis (PCA) refers to a statistical method of dimension reduction, aimed at reducing information loss whilst maintaining the same interpretability and allowing classifiers to take less time during training due to a more efficient data structure, and allowing some classification algorithms such as k-Nearest Neighbours to not be affected by the curse of dimensionality. Since the dataset consists of images of 28 by 28 pixels for each article of clothing, we aim to reduce the components required for training the classifiers from 784 pixels to a more manageable 100 pixels.

With normalisation and PCA of the initial raw data, we can conclude the pre-processing phase of the dataset.

Classification Algorithms

The first classifier used is the **K-Nearest Neighbours** which works by calculating the distance between the test data with the closest K neighbours, or data points, to the test data. Whilst the accuracy is the highest, it should be noted that the runtime of kNN is the highest during testing as it needs to calculate the distance between the test data and all training data. Upon using parameter tuning using grid search, using Manhattan distance as the metric and selecting 5 as the k value yielded the highest accuracy and was chosen as the best classifier.

The second classifier used was the **Naïve Bayes** classifier, and it works by calculating that probability of the test data, given the probability of each of its attributes belonging to any given class. For the solution, a Gaussian Naïve Bayes algorithm was used under the assumption of the data adhering to a normal distribution. Parameter tuning for Naive Bayes only resulted in the tuning of one parameter, var_smoothing, which artificially widens or smooths the curve by modifying the distribution's variance. An optimal value was reached.

The third and final classifier used was a **Decision Tree** classifier, and it reaches a solution by utilising the concept of entropy and information gain from the rules-based learning that allows it to construct "trees". A gain ratio can be found for attributes, with the highest ratio contributing towards a splitting attribute. A similar method can be used, named the Gini index, with the minimum Gini value being the splitting attribute. Therefore, for the parameter tuning of the Decision Tree classifier, between the two methods, the grid search found the entropy method to yield a higher accuracy rate.

Experiments and Results

The experiment design consisted of loading the pre-processed data set and running the kNN, Naive Bayes and Decision Tree classifiers initially with default or conventional values, and their accuracies recorded. The next step was to compare their accuracies using their hyper-parameters, which required executing Grid Search and selecting parameters and running each of the three classifiers a second time, with the optimal parameters being stored along with the highest accuracy achieved. Upon comparing, the following were the results of the studies:

Classifier	No Tuning	Hyper-Parameters
K-Nearest Neighbours	85.52% k = 5; Distance = Minkowksi	**85.88%** k = 5; Distance = Euclidean
Naïve Bayes	76.66% var_smoothing = 1e-9	**76.8%** var_smoothing ~ 0.00035
Decision Tree*	76.38% Criterion = Entropy	76.38% Criterion = Entropy

* Note that due to the nature of the Decision Tree classifier and the single parameter tuned, there were no changes made from the initial untuned execution, hence yielding in the same optimal accuracy of 76.38%.

Based on the results, the kNN classifier yields the highest accuracy of the three classifiers used, however, there is a rather minute increase when changing the distance metric from Minkowski to Euclidean, as the solution converged from the grid search declares. Whilst the accuracy for kNN is the highest, it is also by the far the most computationally expensive during testing and took the longest of the three to execute.

Conclusion

Evident from the results of the experiment, the kNN classification algorithm yields the highest accuracy, albeit with a minute increase when using the hyper-parameters. It is also found that the kNN algorithm takes much longer than Naïve Bayes and the Decision Tree classifiers. Therefore, it is arguable that the kNN may not be the winner if the metrics for the best classifier are determined by efficiency and runtime. However, if accuracy scores are the sole criterion, then it is abundantly clear that kNN is the winner. For future work related to this topic, it would be perhaps more insightful to experiment and compare with multi-layer perceptrons, namely a convolutional neural network as they have been historically much more accurate when it comes to the classification of image data, such as the one used in this experiment. It could also be insightful in ranking algorithms based on an index of accuracy, complexity and runtime as the other two components are equally paramount in real-world scenarios. A slightly slower accuracy may be favoured if it allows for a faster runtime and a computationally inexpensive solution.

Appendix

Hardware used for Experiment:
MacBook Pro (13-inch Mid 2018)
Intel Core i5-8259U @ 2300 MHz
8GB 2133 MHz LPDDR3 SDRAM

Software used for Experiment:
OS: MacOS Big Sur 11.6
Python 3.8.8 (default, Apr 13 2021, 12:59:45)
Jupyter Notebook 6.3.0

Code in the Jupyter notebook can be run by first installing the libraries and dependencies used, and then cells executed in order of their appearance. Requires libraries: numpy, h5py, matplotlib and sklearn.

References

Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011

sklearn.neighbors.KNeighborsClassifier — scikit-learn 0.24.2 documentation. (2021). Retrieved 24 September 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

sklearn.naive_bayes.GaussianNB — scikit-learn 0.24.2 documentation. (2021). Retrieved 24 September 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

sklearn.tree.DecisionTreeClassifier — scikit-learn 0.24.2 documentation. (2021). Retrieved 24 September 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

sklearn.model_selection.GridSearchCV — scikit-learn 0.24.2 documentation. (2021). Retrieved 24 September 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

sklearn.preprocessing.StandardScaler — scikit-learn 0.24.2 documentation. (2021). Retrieved 24 September 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

Lovell, C. (2021). h5py: reading and writing HDF5 files in Python. Retrieved 24 September 2021, from https://www.christopherlovell.co.uk/blog/2016/04/27/h5py-intro.html