Image Classifier Implementation

1. Introduction

The purpose of this project is to create a classifier that can categorize photographs from a subset of the Fashion-MNIST dataset. This study is important because the ability to perform accurate image classification is a key component of machine learning. Therefore, I will implement four different classification algorithms and compare their performance using their respective optimal hyperparameters.

2. Methods

2.1 Pre-process data

Before feeding the data into the models, I implemented normalization to pre-process the data. There are several techniques of normalization, but I applied min-max scaling. By using the min-max scaler, the pixel values in the images were rescaled to a range between 0 and 1, making them more comparable to other features in the dataset. To summarize, min-max scaling improves the performance of machine learning algorithms and makes them more robust to variations in the input data [1].

2.2 Method 1 - Naïve Bayes

This is a probabilistic algorithm based on Bayes' theorem, assuming that the features are independent of each other given the class label [2].

- Firstly, I imported the Gaussian Naïve Bayes classifier from the scikit-learn library as my model, the 'GaussianNB()' function initialized the classifier with default parameters.
- Then I defined a dictionary called 'nb_param_grid' to specify the range of hyperparameters 'var_smoothing'. In my code, I tried the values 1e-9, 1e-8, 1e-7, 1e-6 and 1e-5.
- After that, the 'GridSearchCV' function performed a cross-validation with 'cv=5', which means the data was split into 5 equal parts, and each part was used once as the validation set while the remaining parts were used as the training set. This grid search returned the best parameter value that maximized the accuracy on the validation set.
- Lastly, I fit Gaussian Naïve Bayes classifier with the best hyperparameter value found by the grid search on the preprocessed training data 'X_train_normalized' and the corresponding class labels 'y_train' and save the 'best_nb' model for later comparisons.

2.3 Method 2 – Logistic Regression

Logistic regression is a common classification method used to model the probability of a binary outcome using a logistic function. In this method, I implemented logistic regression using the scikit-learn library [3].

- Firstly, I initialized logistic regression with default hyperparameters by calling the 'LogisticRegression()' function.

- Then I defined the 'lr_param_grid' dictionary to specify the hyperparameters to be tuned by grid search. The regularization penalty could be either 'l1' or 'l2', and the strength 'C', controls the trade-off between fitting the training data and preventing overfitting.
- After that, I used the 'GridSearchCV' function with 5-fold cross-validation to perform a grid search over the hyperparameter spaces specified in the 'lr_param_grid' dictionary. The function returned the best hyperparameter combination.
- Once the grid search was complete, the best estimator was stored as 'best_lr' for later comparisons.

2.4 Method 3 – Decision Tree

Decision tree classifier is another widely used algorithm for classification tasks that divides the feature space into smaller regions based on the input feature values [4]. In this case, I implemented this classifier using the scikit-learn library.

- Firstly, I initialized the decision tree classifier with default hyperparameters by calling 'DecisionTreeClassifier()'.
- Then I use the 'dt_param_grid' dictionary to specify the hyperparameters to be tuned using grid search. The 'max_depth' parameter controls the maximum depth of each tree, and 'min_sample_leaf' sets the minimum number of samples required to be at a leaf node.
- 'GridSearchCV' function performed a grid search over the hyperparameter space of 'dt_param_grid' using 5-fold cross-validation.
- Lastly, I called the 'fit ()' method to perform a grid search over the 'dt_param_grid' and returned the best hypermeter combination.

2.5 Method 4 – Random Forests

Random forest is an ensemble method which combines multiple decision trees to improve the accuracy and robustness of the classification [5].

- The random forest classifier was initialized with default hypermeters by calling 'RandomForestClassifier()' from the scikit-learn library.
- Then I used the 'rf_param_grid' dictionary to specify the hyperparameters to be tuned using grid search. The hyperparameter 'n_estimators' controls the number of decision trees in the random forest, and the 'max_depth' hyperparameter sets the maximum depth of each decision tree.
- Same as how I implemented the previous classifiers, I called the 'GridSearchCV' function to perform a grid search, and 'fit ()" method to fit the random forest classifier on the preprocessed training data.
- Once the grid search was complete, the best hyperparameters were stored in 'rf_best_param', and it can be used to train a final random forest classifier on the entire training data set, then to make predictions on unobserved data.

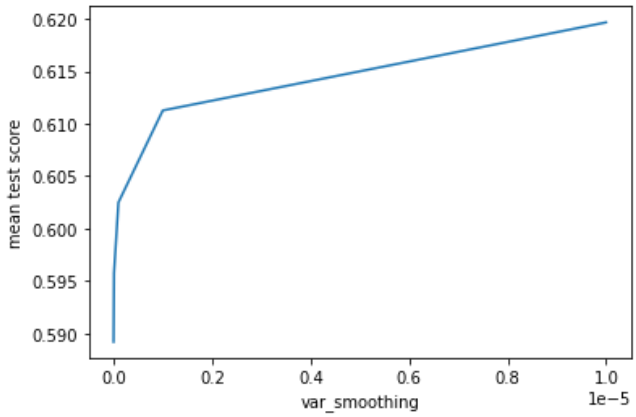
3. Performance, Evaluation & Comparison

3.1. Naïve Bayes

To visualize the performance of Naïve Bayes with respect to different values of 'var_smoothing', a line plot was created using Matplotlib. The graph showed the relationship between the values

of ‘var_smoothing’ and the mean test score of Naïve Bayes. Based on this plot, the best value of ‘var_smoothing’ seems to be around 1e-05.

Line Graph:



3.2. Logistic Regression

Table 1, named ‘lr_cv_results’ was created to summarize the results of the hyperparameter tuning process for Logistic Regression. The table shows the performance of the hyperparameters used for each cross-validation fold; the mean test score obtained for each combination of hyperparameters was sorted by the rank of the test score in ascending order. The best hyperparameters for logistic regression models are ‘C = 0.1’ and penalty = ‘l2’. This indicates that the l2 regularization penalty with a regularization strength of 0.1 provides the best performance on the given dataset.

Table 1:

param_penalty	param_C	mean_test_score	rank_test_score
l2	0.1	0.850367	1
l2	1	0.849333	2
l2	10	0.848500	3
l2	0.01	0.841100	4
l1	0.01	NaN	5
l1	0.1	NaN	5
l1	1	NaN	5
l1	10	NaN	5

3.3. Decision Tree

To summarize the hyperparameter tuning process of Decision Tree, table 2 ‘dt_cv_results’ was generated. It displays the hyperparameters used in each cross-validation fold, the mean test score achieved for each hyperparameter combination, and their respective rank based on their mean test score. Based on the findings, the optimal hyperparameters for decision tree model were determined to be ‘max_depth = 20’ and ‘min_samples_leaf = 5’, which implied that the decision tree model with a maximum depth of 20 and a minimum of 5 samples required at a leaf node exhibited the best performance on the dataset provided.

Table 2:

param_max_depth	param_min_samples_leaf	mean_test_score	rank_test_score
10	5	0.794200	1
30	15	0.792967	2
20	15	0.792633	3
10	10	0.792600	4
10	15	0.791533	5
20	10	0.791267	6
30	10	0.791133	7
20	5	0.787000	8
30	5	0.786867	9

3.4. Random Forests

Table 3 'rf_cv_results' table was displayed to summarize the results of the hyperparameters tuning process of Random Forests method. It demonstrated the hyperparameters used in each cross-validation fold, the mean test score for each combination of hyperparameters, and their corresponding rank based on the mean test score. Based on the results, the optimal hyperparameters for Random Forest model were determined to be 'n_estimtors = 200' and 'max_depthn = 20'. This indicated that employing 200 trees in the forest and setting the maximum depth of the tree to 20 yields the best performance on the dataset provided.

Table 3:

param_n_estimators	param_max_depth	mean_test_score	rank_test_score
200	20	0.875033	1
200	30	0.874400	2
100	20	0.873467	3
100	30	0.871967	4
50	20	0.870267	5
50	30	0.869767	6
200	10	0.850100	7
100	10	0.849533	8
50	10	0.848500	9

3.5. Comparison

Lastly, once the optimal hyperparameters for each algorithm were determined, I would like to compare all classifiers by evaluating their accuracy and running time on the validation set. Table 4 highlighted the accuracy score and running time for each model:

Table 4

```
Best Naive Bayes
Accuracy: 58.51%        time: 2.23s
Best Logistic Regression
Accuracy: 84.15%        time: 8.30s
Best Decision Tree
Accuracy: 76.58%        time: 4.30s
Best Random Forests
Accuracy: 85.80%        time: 26.86s
```

Based on the findings, it is evident that the Random Forest model exhibits the highest level of performance, achieving an accuracy of 85.80%. The second-best model is Logistic Regression, with an accuracy of 84.15%. Following that, the Decision Tree model has an accuracy of 76.58%, while Naive Bayes has the lowest accuracy of 58.51%.

The factors which affect performance among the models can be various. For instance, Naïve Bayes assumes independence between features, which may not hold true in real-world situations [6]. Random Forests outperform Decision Trees in terms of accuracy, it is because Random Forests use an ensemble of Decisions Trees to reduce overfitting. By creating multiple Decision Trees on different subsets of the data and features, Random Forests captured more of the complexity of the data and provided more accurate predictions [7].

Regarding the training and inference time consumption, Naïve Bayes is the fastest, with a training time of 2.23 seconds while Random Forest takes longer to train the data. This is possibly because Naïve Bayes is a relatively simple model, with a training process that only calculates the mean and variance of each feature. In contrast, the Random Forest model requires training multiple decision trees and combining their outputs, which takes more time. It is also notable that Logistic Regression is faster than Random Forests, so it might be a good option if training time is a critical factor in the application.

4. Conclusion

To conclude, Random Forest is the most effective model based on the experimental results, as it strikes a balance between accuracy and training time. Therefore, I will use it as the best model to fit on the normalized training dataset and make predictions on new, unseen data which is the normalized test data 'X_test_normalized'. It then saves the output to a CSV file named 'test_output.csv'.

In future work, it would be worthwhile to investigate additional classification algorithms, such as SVM, KNN and Neural Networks, to determine whether they can achieve higher performance than the current models.

References

[1] Patro, S. Gopal Krishna, and Kishore Kumar Sahu. Normalization: A Preprocessing Stage. arXiv, 19 Mar. 2015. arXiv.org, http://arxiv.org/abs/1503.06462.

[2] L, Meghana, and N. Deepika. 'A Survey on Different Classification Techniques In Data Mining'. International Journal of Science and Engineering Applications, vol. 6, no. 1, Jan. 2017, pp. 001–07. DOI.org (Crossref), https://doi.org/10.7753/IJSEA0601.1001.

[3] Stoltzfus, J. C. (2011). Logistic regression: A brief primer: logistic regression: a brief primer. Academic Emergency Medicine, 18(10), 1099–1104. https://doi.org/10.1111/j.1553-2712.2011.01185.x

[4] Friedl, M. A., & Brodley, C. E. (1997). Decision tree classification of land cover from remotely sensed data. Remote Sensing of Environment, 61(3), 399–409. https://doi.org/10.1016/S0034-4257(97)00049-7

[5] Latha, C. B. C., & Jeeva, S. C. (2019). Improving the accuracy of prediction of heart disease risk based on ensemble classification techniques. Informatics in Medicine Unlocked, 16, 100203. https://doi.org/10.1016/j.imu.2019.100203

[6] Chen, S., Webb, G. I., Liu, L., & Ma, X. (2020). A novel selective naïve Bayes algorithm. Knowledge-Based Systems, 192, 105361. https://doi.org/10.1016/j.knosys.2019.105361

[7] Fawagreh, K., Gaber, M. M., & Elyan, E. (2014). Random forests: From early developments to recent advancements. Systems Science & Control Engineering, 2(1), 602–609. https://doi.org/10.1080/21642583.2014.956265

Appendix

Appendix A: Running the Code

To run the code in Google Colab, please follow the steps below:

1. Make sure you have access to Jupyter Notebook and all necessary libraries and packages. You can install them by running 'pip install -r requirements.txt' in your terminal or command prompt.
2. Download the Jupyter Notebook containing the full code of this assignment.
3. Open Google Colab.
4. Click on File > Upload notebook.
5. Upload the Jupyter Notebook you downloaded in step 2.
6. Follow the instructions within the notebook to run the code.
7. Make sure to save any changes you make to the notebook before closing.

Appendix B: Hardware and Software Environment

The code was developed and tested on a Google Colab notebook using the following hardware and software environment:

- CPU: Intel(R) Xeon(R) CPU @ 2.30GHz
- RAM: 12 GB
- Operating System: Ubuntu 18.04.5 LTS (Bionic Beaver)
- Python version: 3.9.16

***Writing Style (2 points)**

Your writing is concise, clear, and free of grammatical and spelling errors. You use appropriate technical terminology. Your paragraphs and sentences are well connected and follow a clear logic. There is a distinction between the essential parts of the report and less important material (use the appendix).