

Comparison of Machine Learning Methods on Tabular and Numerical Datasets

1st Max Gao

School of Manufacturing Networks and Systems
Arizona State University
Tempe, Arizona, United States of America
mgao18@asu.edu

2nd Yuanhao Chen

School for Engineering of Matter, Transport and Energy
Arizona State University
Tempe, Arizona, United States of America
yche1066@asu.edu

3rd Sai Manish Rao Marru

School of Computing and Augmented Intelligence
Arizona State University
Tempe, Arizona, United States of America
smarru@asu.edu

4th Anaam Mostafiz

School for Engineering of Matter, Transport and Energy
Arizona State University
Tempe, Arizona, United States of America
amostafi@asu.edu

I. INTRODUCTION

Machine Learning is the study of data-driven prediction and analysis, built upon fundamental principles of statistics and probability theory. Machine Learning represents the translation of complex tasks, such as the classification of spam emails from legitimate emails, into a form that is mathematically executable, generally using computers. As the development and evolution of machine learning models is fundamentally based exclusively on data, such translations of complex tasks becomes essentially an approximation attempting to replicate results, with accuracy being limited by the size of the data available. [1]

With Machine Learning appearing in many different forms, it becomes important to gain an understanding of which of these forms, and more so the hyperparameters that govern them, are most important for analyzing different datasets. It is the goal of this study to investigate the usability of four such classifiers on two different data sets, to determine the particular properties of each classifier and the particular impact of their hyperparameters.

A. Datasets Used

In this study, three datasets will be analyzed by various Machine Learning classifiers. The first, the Wisconsin Breast Cancer Diagnostic Dataset, consists of thirty numerical features and a binary diagnostic of whether a present tumor is benign or malignant. [2] The second, UCI Adult, is a similar dataset consisting of categorical features with an observation indicating annual income being above or below \$50,000. [3] Lastly, Fashion MNIST is a numerical dataset of many low-resolution images presenting one of ten apparel items. [4]

B. Model Classes Used

Four classification models will be used in this study, each having similar applications but differing approaches. Firstly, Logistic Regression is a binary classification model that estimates the probability of a particular binary outcome using a logit transformation. The outcome is the log-odds, or the log of the probability of one state over another. [5] Next, Support Vector Machines (SVM), is a class-separator technique that seeks to maximize the margin, or separation between two classes in a data space. [5] Third, k-Nearest Neighbor (k-NN), is a simple classifier that assigns a class to a new data point based on the classes of the nearest k data points. [5] Due to known instability issues with working in high-dimensional data spaces, this will be paired with Principle Component Analysis (PCA), which is a data simplification technique that compresses the dimensionality of complex datasets. [5] Finally, the Neural Network is a versatile classifier featuring many possible layers of data transformation. If working on images, a Convolution filter set can also be applied to pre-transform the image-based data to a format more suitable for a standard Neural Network.

C. Team Contribution

The four team members of this project study contributed in different ways towards its success shows in Table I below.

Team Member	Max Gao	Anaam Mostafiz	Yuanhao Chen	Sai Manish Rao Marru
Contribution	Neural Networks and Report Writing	Logistic Regression and Report Writing	k-NN / PCA and Report Writing	SVM and Report Writing

TABLE I: Team Member Contributions

II. DATASET 1: DIAGNOSTIC BREAST CANCER

A. General Approach

Before implementing all models, the Breast Cancer dataset was preprocessed using functions from the scikit-learn library. [6] This involved using the `StandardScaler()` function to standardize the numerical features, and `train_test_split()` to partition the dataset into 80% training and 20% testing. The numpy library was used to aid in basic array operations. [7]

All hyperparameter tuning was done using grid search, using either scikit-learn's `GridSearchCV` or manually with `itertools`. [8] Where applicable, a 5-fold cross-validation structure was implemented using `KFold()` from scikit-learn. Lastly, the libraries pandas, matplotlib, and seaborn were used for data recording and visualization. [9]–[11] Most of the code was written by the team, with some general coding assistance from ChatGPT. [12]

B. Logistic Regression Setup and Results

Overall, the logistic regression modeling procedure is as follows. First, the data is split into training and test sets. The Fashion MNIST data already imports into training and test sets, while the other datasets are split into 80% training and 20% test data. Then, 5-fold cross-validation is performed on the training set for a parameter grid. Finally, the parameters with the lowest average validation error are used to train the model and evaluate its accuracy with the test set.

Logistic regression is performed with the `LogisticRegression()` function, and k-fold cross-validation of the parameter grid is performed using the `GridSearchCV()` function. This procedure is repeated for logistic regression with L2, L1, both L2 and L1 (Elastic-Net), and no regularization to compare their accuracy. All parameters can be found in Table II.

Reg. Type	Breast Cancer and Adult Datasets		Fashion MNIST Dataset	
	Parameters	Values	Parameters	Values
L2	C: Regularization Parameter	[0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 100]	C: Regularization Parameter	[0.01, 0.1, 1]
	C: Regularization Parameter	[0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 100]	C: Regularization Parameter	[0.01, 0.1, 1]
L1	C: Regularization Parameter	[0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 100]	C: Regularization Parameter	[0.01, 0.1, 1]
	C: Regularization Parameter	[0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 100]	C: Regularization Parameter	[0.01, 0.1, 1]
Elastic-Net	C: Regularization Parameter	[0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 100]	C: Regularization Parameter	[0.01, 0.1, 1]
	l1_ratio: Mixing parameter	[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]	l1_ratio: Mixing parameter	[0.2, 0.4, 0.6, 0.8]
None	n/a	n/a	n/a	n/a

TABLE II: Logistic Regression Parameters for 3 Datasets

For the Breast Cancer dataset, L1 regularization with $C=1$ and Elastic-Net regularization with $C=1$ and $l1_ratio=0.8$ both resulted in the best average validation accuracy of 0.978. Since Elastic-Net was more accurate than L1 regularization for most other values of C , it seems that Elastic-Net is the better overall model. Since $l1_ratio=0.8$, this suggests that the relevant features are sparse and that there are some correlated features.

With this model, the test set confusion matrix is [70, 1; 2, 41]. This suggests that the model is a great predictor of both class 0 and class 1, which is essential when predicting cancer. The test set classification report corroborates this conclusion, which can be found in Table X and Fig. 5.

C. Support Vector Machines Setup and Results

Prior to SVM model training, data preprocessing is conducted to address missing values and normalize feature ranges. The baseline SVM model is established using `SVC()` with default settings (*default parameters: kernel:rbf, C:1.0, γ :scale*).

Hyperparameter tuning is performed via `GridSearchCV()`, with parameters specified in Table IX, employing a 5-fold cross-validation (*training-validation split ratio 80% – 20%*) to find the parameter set that maximizes mean accuracy. A subsequent round of tuning is conducted reduced search space for parameters (C, γ, d), based on the kernel type. Performance comparisons between the tuned model and the baseline are drawn using classification reports, validation curves, and a confusion matrix.

The range of parameter values that have been evaluated are listed in the Table XII (*total candidates evaluated: 5202*) and Table XIII (*total candidates evaluated: 10,000*) and the best parameters obtained are (*kernel: RBF, C: 5.0802, γ : 0.0171*). By tuning the parameters SVM model was able to achieve an *accuracy of 98%* in comparison to the default/baseline model of *accuracy of 97%* (Table XI). With the help of validation curves for C and γ (Fig 6 and Fig 7), we can observe that at (C : 5.0802, γ : 0.0171) both training and test accuracy are high indicating that the model is neither underfitting nor overfitting.

In case of predicting medical cancer, it is important to reduce false positive and false negative predictions, the SVM model was able to do this and with the help of confusion matrix, we can see that (*Number of False positives: 0, Number of False Negatives: 2*). And also, the F1-Score of (*Class Benign: 98% and Class Malignant: 98%*), indicate that both the classes are balanced and the model performs well for both the class.

SVM Kernels and its Parameters details: Appendix Section VII-A, Table IX
 Details related to Hyperparameter tuning: Appendix Section VII-C, (Table XII and Table XIII)
 Validation Curves for Parameters C and γ : Appendix Section VII-C, (Fig 6 and Fig 7)
 SVM model comparison and Classification Report: Section VII-C, Table XI

Confusion matrix:

$$\begin{pmatrix} 70 & 0 \\ 2 & 42 \end{pmatrix}$$

D. *k*-Nearest Neighbors Setup and Results

The PCA + *k*-NN method can efficiently simplify complex data while maintaining critical features for accurate predictive modeling. The PCA was applied to reduce the dimensionality of the datasets, and we chose a variance threshold of 95% for it, ensuring that the majority of the data's variability was retained while simplifying the models. After dimensionality reduction, we employed the *k*-NN algorithm for classification. We conducted hyperparameter tuning, particularly adjusting the *n_neighbors* parameter, to optimize the classifier's performance for each dataset. All hyperparameters are included in Table III below. Finally, we evaluated the model's performance using cross-validation, and identified the optimal parameters through a grid search.

Hyperparameter	Values / Range
k-NN Neighbors	{3, 5, 7, 9, 11}
PCA Variance	95%

TABLE III: Hyperparameters Evaluation in PCA + *k*-NN Model

Particularly, variable *y* is expected to be a one-dimensional array of shape *n_samples*, which holds the target values (labels) for the classification or regression tasks. The *y_reshaped* line checks if *y* is a pandas DataFrame. If yes, then it will convert *y* to a one-dimensional numpy array, which is necessary to be processed here.

The procedure began with a standardized scaling of the dataset, followed by PCA for dimensionality reduction. The *k*-NN classifier was then set up with a hyperparameter grid that defined the range of neighbors to consider. Through a cross-validation approach with a 5-fold split, we sought the optimal number of neighbors, which was determined to be 3, as per the grid search results. For *k*-NN neighbors we tuned from the set {3, 5, 7, 9, 11}.

Our validation approach involves a training-validation split (70%-30%), ensuring a clear and consistent evaluation process. The performance of the model was primarily assessed using precision (F1-score metrics, indicates the model's ability to classify the different classes accurately). The best parameters obtained from hyperparameter tuning were *n_neighbors*: 3, and the classification performance on the test set has an accuracy of 95%, as well as others all being 95%. This indicates a well-balanced model that performs equally well for both classes (B for benign and M for malignant). The training time for the model was a brief 0.56 seconds, emphasizing an efficient computational process. The results, including the detailed classification report and the confusion matrix, are attached in the Table XIV, which demonstrates the model's predictive capabilities.

E. Neural Networks Setup and Results

For training and evaluating neural networks, the library TensorFlow [13] was used in conjunction with utility libraries such as Multiprocessing, CSV, and itertools for easy mass data collection. The architecture of the script was built to accommodate for the requirements for multi-threading, with a single model creation and evaluation function being executed by a pool of asynchronous workers using Multiprocessing's Pool() and starmap() functions as can be seen in Appendix Sections VII-O, VII-S, and VII-W. A grid search was done over four hyperparameters, with every hyperparameter combination being trained over 100 epochs and evaluated with 5-fold cross validation. Validation loss, accuracy, and training time were recorded.

The four hyperparameters varied in the search were hidden layer width, hidden layer count, batch size, and optimizer function. The variances of these hyperparameters were anticipated to be the most impactful to score. With a small dataset implying fast computation time, at least five potential values of layer width, batch size, and optimizer function were tested. Only two values for hidden layer count were considered, as three or more hidden layers would likely make for an unnecessarily complicated model. For simplicity, all activation functions save for the final one were fixed to be ReLU, and all hidden layers were made the same size.

After searching all combinations, it was found that most trained models were highly accurate, with 190 of 350 models returning a validation accuracy of better than 98%. The best selected model reached a validation accuracy of 98.9% and a test accuracy of 96.5%, implying that the validation results are representative of hypothetical test results on other combinations.

Among top models, actual hyperparameter values varied greatly, but it was found as in Figure 1 that a smaller batch size, greater layer depth, fewer layers, and usage of either SGD, adam, or rmsprop optimizers returned better results. This primarily indicates that a Neural Network model works best with one large and complicated hidden layer when working with this sort of data. It also seems that this is universal when training models with different batch sizes and using different optimizer functions. It should also be noted that the selected best model does not necessarily adhere to this trend, either, as it was trained with two hidden layers of depth 20, with a batch size of 2 on the adam optimizer. A more detailed analysis is available in Appendix VII-D's Table XV.

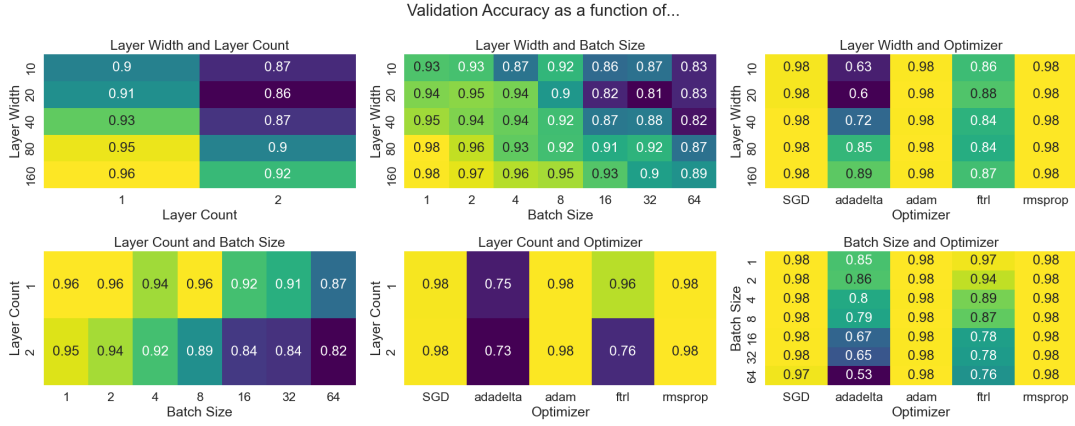


Fig. 1: Dataset 1 Hyperparameter Analysis based on Validation Accuracy

III. DATASET 2: ADULT

A. General Approach

The overall plan for handling the UCI Adult dataset is very similar to that of the Breast Cancer dataset, with only the added step of using scikit-learn's `OneHotEncoder()`, `ColumnTransformer()`, and `Pipeline()` functions. In case of SVM, `LableEncoder()` is used to encode the categorical data. These are only used to convert the mixed tabular and categorical data format into a sparse matrix format that can be interpreted by other machine learning functions. If the dataset is downloaded, then the training and test sets are already provided, so splitting the entire data is not needed.

B. Results pertaining to Logistic Regression

For the Adult dataset, the best logistic regression model was found to be L1 regularization with $C=0.01$, with an average validation accuracy of 0.8203. Since it is L1 regularization, this suggests that the relevant features are sparse.

With this model, the test set confusion matrix is [6395, 350; 1272, 1028]. This suggests that the model is a good predictor of class 0, but not the best predictor of class 1. The test set classification report corroborates this conclusion.

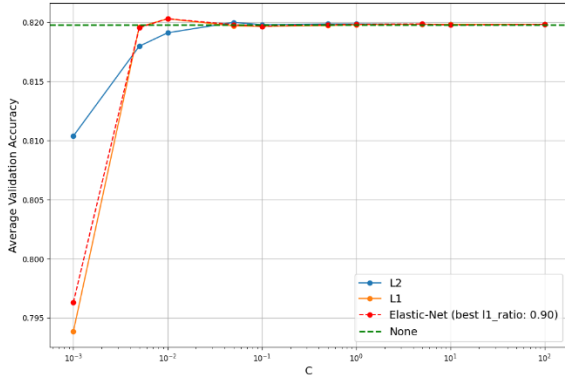


Fig. 2: Average Validation Accuracy vs C

Best parameters: {L1, C: 0.01}

Confusion matrix:

$$\begin{pmatrix} 6395 & 350 \\ 1272 & 1028 \end{pmatrix}$$

Classification report:

	Precision	Recall	F1-Score	Support
$Income \leq 50K$	0.83	0.95	0.89	6745
$Income > 50K$	0.75	0.45	0.56	2300
Accuracy	—	—	0.82	9045
Macro Avg.	0.79	0.70	0.72	9045
Weighted Avg.	0.81	0.82	0.80	9045

TABLE IV: Learning Result for Dataset 2

C. Results pertaining to Support Vector Machines

Similar to the SVM model procedure described in the Section II-C for the UCI Breast Cancer dataset, performed a hyperparameter tuning to find the optimal parameters on the given dataset with a 5-fold cross-validation (*training-validation split ratio* 80% – 20%). The range of parameter values that have been evaluated are listed in the Table XVI (*total candidates evaluated: 1072*) and Table XVII (*total candidates evaluated: 1000*) and the best parameters obtained for this search for parameters are (*kernel: RBF*, $C: 1.9067$, $\gamma: 0.1$), as shown in the validation curves for C (Fig 9) and γ (Fig 8). And also the validation curves for C and γ indicates that at ($C: 1.9067$, $\gamma: 0.1$) the model neither underfits nor overfits, as both training and test accuracy are high.

Despite the hyperparameter tuning, the SVM model's accuracy is approximately 85%, which is nearly the same as the accuracy of the baseline/default model (Table V). This outcome can be observed from the confusion matrix, which indicates

that the model correctly predicts low-income instances $\leq 50K$ with high accuracy, but is less accurate with high-income $> 50K$ predictions, hinting at a potential class imbalance or other complexities in the data that may not be addressed by tuning alone. And also, the lower recall (56%) and F1-Score (64%) for ($Income > 50K$) class, as compared to the ($Income \leq 50K$) class (Table V), indicates that the classes are imbalanced and the model does not perform well on the class with ($Income > 50K$).

SVM Kernels and its Parameters details: Appendix Section VII-A, Table IX

Details related to Hyperparameter tuning: Appendix Section VII-F, (Table XVI and Table XVI)

Validation Curves for Parameters C and γ : Appendix Section VII-F, (Fig 8 and Fig 9)

Confusion matrix:

$$\begin{pmatrix} 10658 & 702 \\ 1621 & 2079 \end{pmatrix}$$

	Model with Tuned Parameters					Baseline/Default Model				
Parameters	<i>kernel</i> : RBF	C : 1.90679072	γ : 0.1			<i>kernel</i> : RBF	C : 1.0	γ : scale, $\frac{1}{n_{features} \times \text{var}(X)}$		
Scoring Metrics		Precision	Recall	F1-Score	Support		Precision	Recall	F1-Score	Support
	$Income \leq 50K$	0.87	0.94	0.90	11360	$Income \leq 50K$	0.87	0.94	0.90	11360
	$Income > 50K$	0.74	0.56	0.64	3700	$Income > 50K$	0.75	0.55	0.64	3700
	Accuracy	–	–	0.85	15060	Accuracy	–	–	0.85	15060
	Macro Avg.	0.81	0.75	0.77	15060	Macro Avg.	0.81	0.75	0.77	15060
	Weighted Avg.	0.84	0.85	0.84	15060	Weighted Avg.	0.84	0.84	0.84	15060

TABLE V: SVM Model Evaluation based on Scoring Metrics for Dataset 2

D. Results pertaining to k -Nearest Neighbors

The confusion matrix now clearly distinguishes between the binary classes, showing a total of 9874 true positives for class '0' and 1969 true positives for class '1'. This indicates that the model is highly effective at identifying class '0' instances, with a high true positive rate of 92%. Conversely, the model exhibits a lower sensitivity towards class '1', with a notable 1585 instances that were false negatives, indicating that these instances were incorrectly classified as class '0'.

The Fig. 10 illustrates the proportion of variance explained by each PCA component. The sharp decline observed in the variance ratio remains consistent with the nature of PCA, where the initial components capture the most significant amount of information. The first component accounts for a substantial portion of the variance, emphasizing the importance of the leading components in the dimensionality reduction process.

The revised accuracy of the model stands at 83%, which suggests a competent level of classification performance. The macro-average F1-score, which considers the balance between precision and recall, is at 75%. This indicates a relatively balanced performance across the binary classes, taking into account the class imbalance. The training duration for this refined model increased slightly to 96.29 seconds, which is understandable given the additional preprocessing step and the complexity of the model. All these analysis are included in Table XVIII.

E. Results pertaining to Neural Networks

With insights from the Neural Network experiment on the first dataset and forecasted challenges involved in processing the larger UCI Adult dataset, changes to the planned hyperparameter search space were necessary. Firstly, with a much greater number of samples, raising the batch size was deemed appropriate to cut down on computation time, despite potential gains in accuracy otherwise. Larger layers were also explored due to the greater number of features in the sparse matrix format. Modifying the optimizer functions was considered, though ultimately decided against due to the change in feature format. Optimizers that performed well previously were not expected to do so in this experiment,

After a complete grid search across the same four parameters, the models trained using the UCI Adult Dataset were found to fare worse than the UCI Breast Cancer counterparts in terms of validation accuracy, despite access to much more data. While all 280 models were found to have validation accuracies of greater than 75%, the best model had a validation accuracy of only 87.7% and a corresponding 81.5% test accuracy. As seen in Appendix VII-G and Table XIX, there is a significant imbalance in performance in predicting labels, possibly caused by imbalance in the training data itself.

The same sensitivity analysis as shown in Figure 3 showed that large layer depths, two hidden layers, small batch sizes, and the adam optimizer returned the best results, but only by a slight amount. These relationships suggest very weak overall impact on validation accuracy due to hyperparameter tuning, requiring instead very precise combinations of parameters to see scores marginally better than baseline.

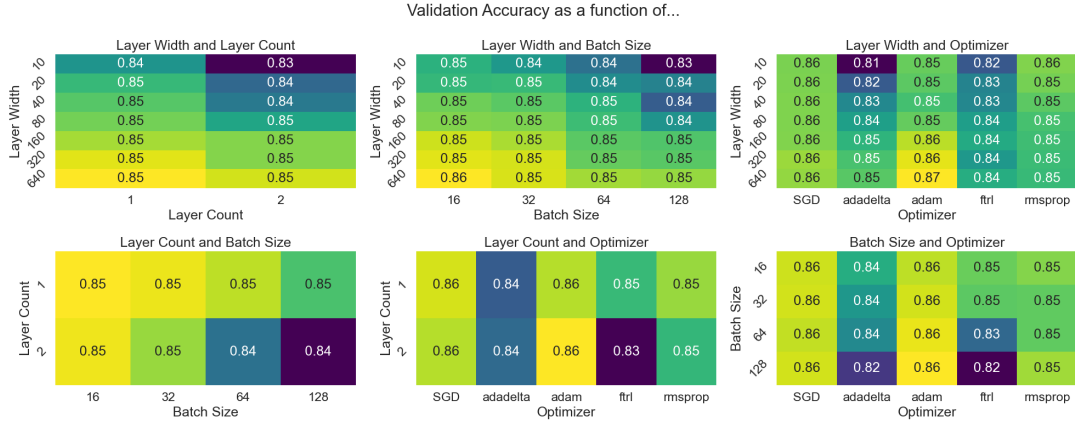


Fig. 3: Dataset 2 Hyperparameter Analysis based on Validation Accuracy

IV. DATASET 3: FASHION MNIST

A. General Approach

The Fashion MNIST datasets are acquired through the torchvision library [14], which are already separated into training and testing components. All data is then normalized by dividing the grayscale pixel values by 255, and then passed on to respective algorithms.

B. Results pertaining to Logistic Regression

Due to the high number of classes and samples in the Fashion MNIST data, it uses a less exhaustive parameter grid to reduce code runtime. All parameters can be found in Table II.

For the Fashion MNIST dataset, the best logistic regression model was found to be Elastic-Net regularization with $C=0.1$ and $l1_ratio=0.2$ (Fig: 11), with an average validation accuracy of 0.8575. Since $l1_ratio=0.2$, this suggests that there are many correlated features, and a few irrelevant features.

The test set confusion matrix (Fig: 12) suggests that the model is a good predictor in general of the classes, with the worst precision for class 6. It is a great predictor for classes 1, 5, 7, 8, and 9. The test set classification report corroborates this conclusion.

Category	0	1	2	3	4	5	6	7	8	9	Average
Precision	0.80	0.98	0.73	0.83	0.74	0.94	0.63	0.91	0.93	0.95	0.84
Recall	0.81	0.96	0.74	0.87	0.76	0.92	0.56	0.94	0.94	0.94	0.84
F1-Score	0.81	0.97	0.73	0.85	0.75	0.93	0.60	0.92	0.94	0.94	0.84
Support	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	10000

TABLE VI: Learning Result for Dataset 3

C. Results pertaining to Support Vector Machines

Due to the computational intensity of training SVM model on the FashionMNIST dataset (*each fit/training time* $\approx 10mins$), reshaped the (28×28) image size to (14×14) without compromising the accuracy of the model (*each fit/training time* $\approx 3mins$) and performed a hyperparameter tuning to find the optimal parameters on the given dataset with a 5-fold cross-validation (*training-validation split ratio* 80% – 20%), similar to the SVM model procedure described in the Section II-C for the UCI Breast Cancer dataset. The range of parameter values that have been evaluated are listed in the Table XX (*total candidates evaluated: 301*) and Table XXI (*total candidates evaluated: 100*) and the best parameters obtained for this search for parameters are (*kernel: RBF, $C: 4.354$, $\gamma: 0.1668$*), as shown in the validation curves for C (Fig 14) and γ (Fig 13). And also the validation curves for C and γ indicates that at ($C: 4.354$, $\gamma: 0.1668$) the model neither underfits nor overfits, as both training and test accuracy are high.

By tuning the parameters SVM model was able to achieve an *accuracy of 90%* in comparison to the default/baseline model of *accuracy of 88%* (Table VII). Based on the *F1 Score(72%)*, Table XXII of class(6: Shirt) compared to the average *F1 Score(90%)* of all classes, indicates that class 6 can be imbalanced and the model may not perform well on classifying class-6.

SVM Kernels and its Parameters details: Appendix Section VII-A, Table IX

Details related to Hyperparameter tuning: Appendix Section VII-J, (Table XX and Table XX)

Validation Curves for Parameters C and γ : Appendix Section VII-J, (Fig 13 and Fig 14)

Classification Report: Appendix Section VII-A, Table XXII

	Model with Tuned Parameters	Baseline/Default Model
Parameters	$kernel$: RBF C : 4.35400465 γ : 0.16681005	$kernel$: RBF C : 1.0 γ : scale, $\frac{1}{n_{features} \times var(X)}$
Accuracy	90%	88%
Best Accuracy: 90%		

TABLE VII: SVM Model Evaluation based on Scoring Metrics for Dataset 3

D. Results pertaining to k-Nearest Neighbors

The chosen hyperparameter for the k-NN algorithm is $k = 5$, which refers to the number of nearest neighbors to consider when making a prediction. The model's overall accuracy reached 86%, and the precision and recall metrics are balanced for most classes. For instance, class '1' (Trouser) has a high precision and recall of 99% and 97% respectively, indicate that the model is particularly adept at identifying this category with few false positives or negatives.

From the confusion matrix (Fig. 15 and Table XXIII), class '6' (Shirt) has a lower f1-score of 61%, suggesting that this class is the most challenging for the model to predict accurately, potentially due to similarity in features with other categories like Tops (class '0') and Coats (class '4'). Next, Fig. 16 cumulative variance plot reveals that to achieve the threshold of 95% explained variance, the model requires a substantial number of principal components. At last, the training time reported is 96.29 seconds.

E. Results pertaining to Neural Networks

In addition to the Neural Network structure established for the prior two datasets, a convolution and a maxpool layer were added to process and downscale the image before passing the refined data to the Feed-Forward Neural Network. Along with the four hyperparameters explored in past experiments, the number of filters in the convolution layer was also to be varied throughout grid search.

However, early testing found an extreme increase in processing time considering a large number of samples, high dimensionality even after maxpool, and the added step of convolution. Because of this, severe restrictions on search space had to be applied, seeing kernel size being locked to 3x3, and significant reductions in how many different hyperparameter values are tested. The number of epochs was also reduced from 100 to 30, accepting losses in accuracy. Despite this, The full list of five optimizer functions were kept as their impact on accuracy was then uncertain.

A comprehensive review of hyperparameter impact on validation accuracy is as follows in Figure 4. We notice that the preferences towards complex single-layer models also appears here, solidifying the trend across all three Neural Network classifiers. Seeing a lack of complex gradients, These plots also suggest little to no cross-correlation between hyperparameters in determining accuracy.

Lastly, as was done with both prior experiments, the best model hyperparameters were selected based on validation accuracy, which was found to be quite high at 96.3%. The test accuracy was comparatively reasonable at 91.1%. With ten labels able to be explored, a confusion matrix was created in Appendix VII-K as Figure 17 to determine which were especially troublesome. It appears that while the model is very accurate overall, a significant number of samples belonging to label 6 are being mistaken for samples belonging to label 0, and vice versa.

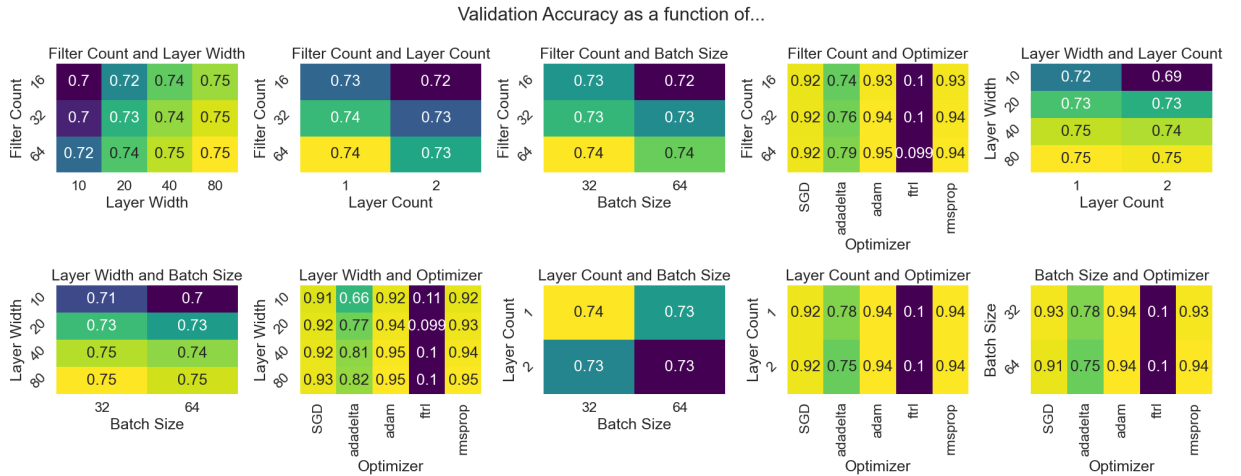


Fig. 4: Dataset 3 Hyperparameter Analysis based on Validation Accuracy

V. VALUE OF CONCEPTS LEARNED

With each classifier having their own unique challenges and architectures, all four members in their own way extracted value out of the concepts and techniques given by the overarching course. Most code used to generate these machine learning results consisted of pre-made libraries and functions, but having an understanding of the underlying mechanics made these algorithms more approachable.

Logistic Regression is a computationally efficient and simple algorithm that learns to predict a class based on probability. To avoid overfitting, L2 and/or L1 regularization can be used with tuned regularization strength and ratio of L1. This can reduce the effects of collinear and irrelevant features.

Support Vector Machine (SVM) is a supervised learning algorithm used for classification and regression. SVM tries to maximize the margin between the classes in the feature space, with the closest data points known as support vectors that define the margin. With the help of kernel functions like polynomial and RBF, it can classify the data that is non-linearly separable by mapping the features to higher dimensional space. The parameter C (regularization parameter) balances the trade-off between maximizing the margin and minimizing the classification errors. A Higher C value tries to classify the data correctly, but can lead to overfitting. While a lower C implies a wider margin and can underfit the model. The γ parameter decides the curvature/smoothness of the decision boundary. A low γ value can result in smooth decision boundary, while a high γ value can result in a complex decision boundary which may lead to overfit. Both parameters C and γ are crucial in tuning SVM for optimal performance.

PCA, as a powerful dimensionality reduction technique, allows us to transform a large set of variables into a smaller one that still contains most of the information in the large set. This is particularly valuable in dealing with high-dimensional data, where it helps to alleviate the curse of dimensionality and reduces computational cost. When we analyze datasets with the k-NN classifier, PCA not only expedites the computation but also potentially increases the accuracy by filtering out noise and irrelevant information. The k-NN algorithm's reliance on distance metrics to make predictions benefits significantly from the reduced-dimensional space provided by PCA.

Lastly, even a conceptual understanding of how neural networks function as a mathematical machine can enable one to leverage their strengths much more effectively. This allows for more informed control of the neural network's shape and complexity. On top of that, of course, it is very helpful to be aware of the many common practices and discoveries surrounding common neural networks, such as the fact that the ReLU, or Rectified Linear Unit function is an absolute staple in deciding on activation functions, or that the Adam and SGD optimizers are two of the most powerful and versatile optimizers in use.

VI. CONCLUSION

In the interests of exploring the behavior of varied parameters, we have developed a set of twelve classifier meant to handle three different classification tasks. Through grid search, we have found and selected for each classifier an optimal set of parameters based on their ability to accurately predict values of a carefully-made set of validation data, navigating computational limitations all the meanwhile. At last, these twelve classifiers were evaluated on their ability to make the same predictions on a larger set of test data, their performances appearing in Table VIII.

With the hyperparameters used, the Support Vector Machines classifier performs best in the two binary classification settings using the UCI datasets, while the Convolutional Neural Network does best on image analysis. It is worth mentioning, however, that most classifiers score very similarly on the same data set, perhaps owing to very rigorous hyperparameter searching.

Classifier	Logistic Regression	Support Vector Machines	k-Nearest Neighbors	Neural Networks
Test Accuracy: Dataset 1	97%	98%	95%	96.5%
Test Accuracy: Dataset 2	81%	85%	83%	81.5%
Test Accuracy: Dataset 3	84%	90%	86%	91.1%

TABLE VIII: Classifier Comparisons

Even with these respectable scores and suspected optimization of hyperparameters, there were some concessions made in the process of choosing hyperparameters. The neural network's high score in the Fashion MNIST dataset predictions appears as a success, but many aspects of its convolutional layer were unexplored simply for the sake of time.

Working with UCI Adult and FashionMNIST datasets also presented few challenges. Class imbalance is a significant issue in the Adult dataset, as it can lead to biased models and also the need to maintain data quality by handling missing values. Meanwhile, the FashionMNIST dataset is computationally intensive to work with due to its high dimensionality and generalizing the models to make multi classification add to its complexity. For all these datasets, and most real-world linearly inseparable data, logistic regression is not ideal as it has a linear decision boundary and cannot capture complex, non-linear relationships.

There are also other ways that challenges can manifest, less so in building a model, but in making a model worthwhile. Being one example, PCA and k-NN are extremely sensitive to initial variable variances, making it difficult to make a model of consistent quality. However, the constructive conclusions and insights gathered throughout the process of this study have shed much more light on the classifiers and datasets used so far, advancing our own understanding of Machine Learning.

VII. APPENDIX

A. Equations

Details related to SVM Kernels and its parameters

Kernel Type	Parameters	Kernel Equation	Parameter Descriptions	Tuned Parameters
Linear	C	$K(x, x') = x \cdot x'$	C : Regularization parameter	C
Polynomial	C, γ, r, d	$K(x, x') = (\gamma \cdot x \cdot x' + r)^d$	C : Regularization parameter, γ : Kernel coefficient r : Bias term (set to default value, 0 for hyperparameter tuning) d : Degree of the polynomial	C, γ, d
RBF (Radial Basis Function)	C, γ	$K(x, x') = \exp(-\gamma \cdot \ x - x'\ ^2)$	C : Regularization parameter γ : Kernel coefficient	C, γ

TABLE IX: SVM Kernels and Their Parameters

B. Detailed Results of Logistic Regression Model for Dataset 1

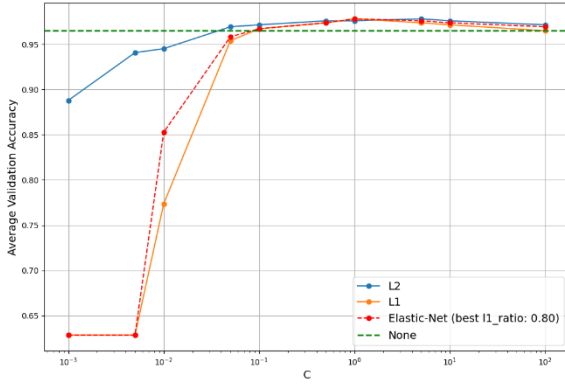


Fig. 5: Average Validation Accuracy vs C

Best parameters: {Elastic-Net, C: 1, L1 ratio: 0.8}

Confusion matrix:

$$\begin{pmatrix} 70 & 1 \\ 2 & 41 \end{pmatrix}$$

Classification report: Table X.

	Precision	Recall	F1-Score	Support
<i>Benign</i>	0.97	0.99	0.98	71
<i>Malign</i>	0.98	0.95	0.96	43
Accuracy	–	–	0.97	114
Macro Avg.	0.97	0.97	0.97	114
Weighted Avg.	0.97	0.97	0.97	114

TABLE X: Logistic Regression Result for Dataset 1

C. Detailed Results of SVM Model for Dataset 1

	Model with Tuned Parameters					Baseline/Default Model				
Parameters	<i>kernel</i> : RBF <i>C</i> : 5.08021804 γ : 0.017190722					<i>kernel</i> : RBF <i>C</i> : 1.0 γ : scale, $\frac{1}{n_features \times \text{var}(X)}$				
Scoring Metrics		Precision	Recall	F1-Score	Support		Precision	Recall	F1-Score	Support
	Benign (Class: 0)	0.97	1.00	0.99	70	Benign (Class: 0)	0.96	1.00	0.98	70
	Malignant (Class: 1)	1.00	0.95	0.98	44	Malignant (Class: 1)	1.00	0.93	0.96	44
	Accuracy	–	–	0.98	114	Accuracy	–	–	0.97	114
	Macro Avg.	0.99	0.98	0.98	114	Macro Avg.	0.98	0.97	0.97	114
	Weighted Avg.	0.98	0.98	0.98	114	Weighted Avg.	0.97	0.97	0.97	114

BestAccuracy: 98%

TABLE XI: SVM Model Evaluation based on Scoring Metrics for Dataset 1

Hyperparameter Tuning:

The search for parameter γ beyond 10^3 is not needed, as it can be observed from the validation curve (Fig 6) that the test accuracy decreases and leads to overfitting. [This is true for all kernels (*polynomial*, *RBF*)]

Similarly, the search for the parameter C beyond 10^6 is not needed, as it can be observed from the validation curve (Fig 7) that the test accuracy begins to decreases from $C \approx 10^2$. [This is true for all kernels (*linear*, *polynomial*, *RBF*)]

Kernel Type	Parameters and its values	Total Candidates Evaluated
Linear	C : 18 equidistant point are selected from the range $[10^{-3}, 10^6]$ on the log scale	18
Polynomial	C : 18 equidistant point are selected from the range $[10^{-3}, 10^6]$ on the log scale γ : 18 equidistant point are selected from the range $[10^{-6}, 10^3]$ on the log scale d : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]	$18 \times 18 \times 15 = 4860$
RBF (Radial Basis Function)	C : 18 equidistant point are selected from the range $[10^{-3}, 10^6]$ on the log scale γ : 18 equidistant point are selected from the range $[10^{-6}, 10^3]$ on the log scale	$18 \times 18 = 324$
Best Parameters for this Search - Kernel: RBF, C : 5.08021804, γ : 0.017190722		

TABLE XII: SVM Hyperparameter Tuning - I, for Dataset 1

Kernel Type	Parameters and its values	Total Candidates Evaluated
RBF (Radial Basis Function)	C : 1000 equidistant point are selected from the range $[0.903, 5.082]$ on the log scale γ : 10 equidistant point are selected from the range $[0.017, 1.719]$ on the log scale	$1000 \times 10 = 10000$
Best Parameters for this Search - Kernel: RBF, C : 5.08021804, γ : 0.017190722		

TABLE XIII: SVM Hyperparameter Tuning - II (refining the parameters with reduced search space)

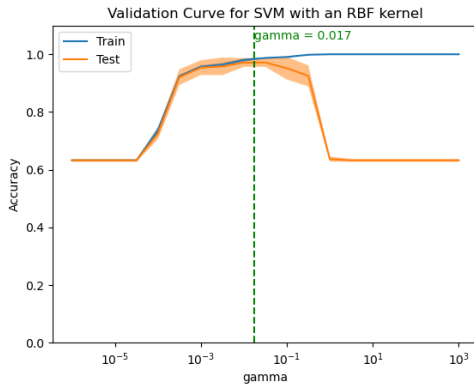


Fig. 6: Validation Curve for SVM with γ parameter

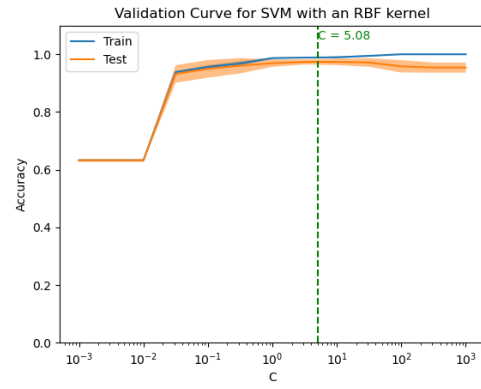


Fig. 7: Validation Curve for SVM with C parameter

D. Detailed Results of k -Nearest Neighbors Model for Dataset 1

Best parameters: {'n_neighbors': 3}

Confusion matrix:

$$\begin{pmatrix} 104 & 4 \\ 4 & 59 \end{pmatrix}$$

Training time: 0.56 seconds

Classification report: Table XIV

	Precision	Recall	F1-Score	Support
B	0.96	0.96	0.96	108
M	0.94	0.94	0.94	63
Accuracy	—	—	0.95	171
Macro Avg.	0.95	0.95	0.95	171
Weighted Avg.	0.95	0.95	0.95	171

TABLE XIV: k -NN Learning Result for Dataset 1

E. Detailed Results of Neural Network Model for Dataset 1

Tested with two 20-width hidden layers, batch size of 2, and 100 epochs on the adam optimizer.

Confusion matrix:

$$\begin{pmatrix} 73 & 3 \\ 2 & 36 \end{pmatrix}$$

Training time: 18.62 seconds

Classification report: Table XV

	Precision	Recall	F1-Score	Support
Benign	0.97	0.96	0.97	76
Malignant	0.92	0.95	0.94	38
Accuracy	—	—	0.96	114
Macro Avg.	0.95	0.95	0.95	114
Weighted Avg.	0.96	0.96	0.96	114

TABLE XV: Neural Network Learning Result for Dataset 1

F. Detailed Results of SVM Model for Dataset 2

Hyperparameter Tuning:

The search for parameter γ beyond 10^2 is not needed, as it can be observed from the validation curve (Fig 8) that the test accuracy decreases and leads to overfitting. [This is true for all kernels (*polynomial*, *RBF*), and also going beyond $\gamma = 10^2$ can cause convergence issues for SVM model with *polynomial* kernel.]

Similarly, the search for the parameter C beyond 10^2 is not needed, as it can be observed from the validation curve (Fig 9) that the test accuracy begins to decreases from $C \approx 10^2$. [This is true for all kernels (*linear*, *polynomial*, *RBF*), and also going beyond $C = 10^2$ can cause convergence issues for SVM model with (*polynomial*, *linear*) kernels.]

Kernel Type	Parameters and its values	Total Candidates Evaluated
Linear	C : 11 equidistant point are selected from the range $[10^{-3}, 10^2]$ on the log scale	11
Polynomial	C : 11 equidistant point are selected from the range $[10^{-3}, 10^2]$ on the log scale γ : 8 equidistant point are selected from the range $[10^{-5}, 10^2]$ on the log scale d : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]	$11 \times 8 \times 10 = 880$
RBF (Radial Basis Function)	C : 11 equidistant point are selected from the range $[10^{-3}, 10^2]$ on the log scale γ : 17 equidistant point are selected from the range $[10^{-5}, 10^3]$ on the log scale	$11 \times 17 = 187$
Best Parameters for this Search - Kernel: RBF, C : 3.16227766, γ : 0.1		

TABLE XVI: SVM Hyperparameter Tuning - I, for Dataset 2

Kernel Type	Parameters and its values	Total Candidates Evaluated
RBF (Radial Basis Function)	C : 100 equidistant point are selected from the range $[0.562, 3.162]$ on the log scale γ : 10 equidistant point are selected from the range $[0.1, 10.0]$ on the log scale	$100 \times 10 = 1000$
Best Parameters for this Search - Kernel: RBF, C : 1.90679072, γ : 0.1		

TABLE XVII: SVM Hyperparameter Tuning - II (refining the parameters with reduced search space)

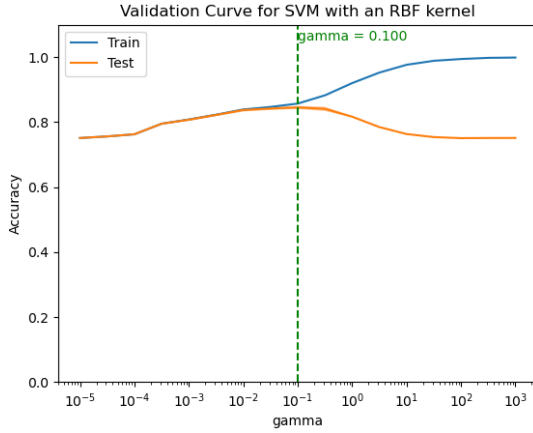


Fig. 8: Validation Curve for SVM with γ parameter

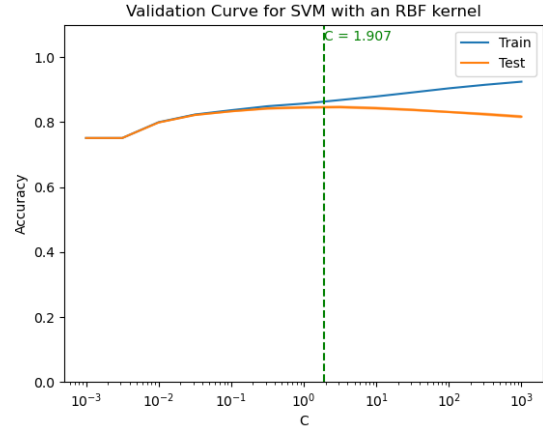


Fig. 9: Validation Curve for SVM with C parameter

G. Detailed Results of k -Nearest Neighbors Model for Dataset 2

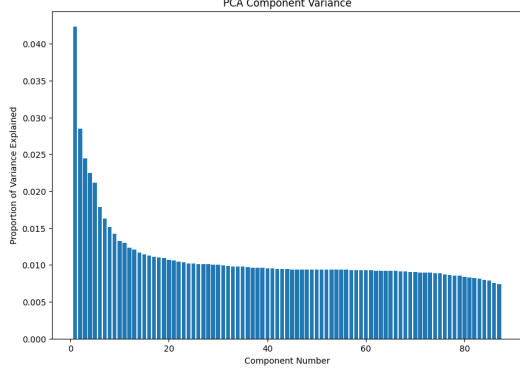


Fig. 10: PCA Component Variance for Dataset 2

Best parameters: {'k-NN_n-neighbors': 11}
Confusion matrix:

$$\begin{pmatrix} 9874 & 859 \\ 1585 & 1969 \end{pmatrix}$$

Training time: 96.29 seconds

Classification report: Table XVIII.

	Precision	Recall	F1-Score	Support
0	0.86	0.92	0.89	10733
1	0.70	0.55	0.62	3554
Accuracy	—	—	0.83	14287
Macro Avg.	0.78	0.74	0.75	14287
Weighted Avg.	0.82	0.83	0.82	14287

TABLE XVIII: k-NN Learning Result for Dataset 2

H. Detailed Results of Neural Network Model for Dataset 2

Tested with two 640-width hidden layers, batch size of 128, and 100 epochs on the adam optimizer.
Confusion matrix:

$$\begin{pmatrix} 6348 & 846 \\ 914 & 1417 \end{pmatrix}$$

Training time: 136.44 seconds

Classification report: Table XIX.

	Precision	Recall	F1-Score	Support
Below \$50K	0.87	0.88	0.88	7194
Above \$50K	0.63	0.61	0.62	2331
Accuracy	—	—	0.82	9525
Macro Avg.	0.75	0.75	0.75	9525
Weighted Avg.	0.81	0.82	0.81	9525

TABLE XIX: Neural Network Learning Result for Dataset 2

I. Detailed Results of Logistic Regression Model for Dataset 3

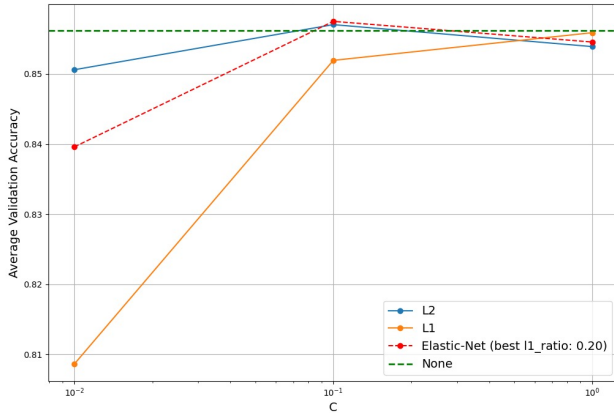


Fig. 11: LR: Average Accuracy VS C for Dataset 3

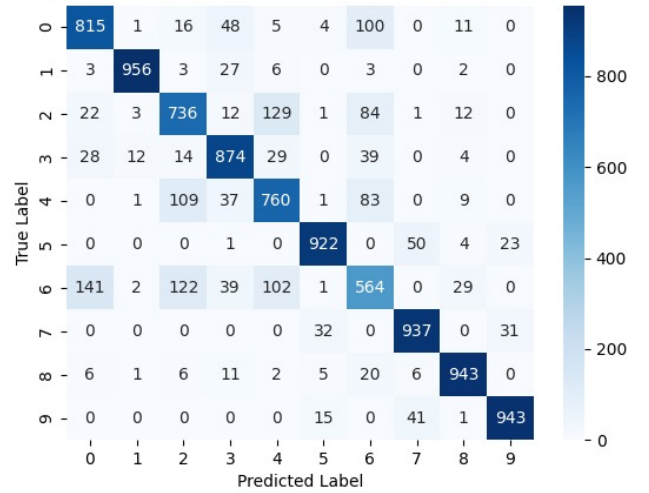


Fig. 12: Logistic Regression: Confusion Matrix for Dataset 3

J. Detailed Results of SVM Model for Dataset 3

Hyperparameter Tuning:

The search for parameter γ beyond 10^2 is not needed, as it can be observed from the validation curve (Fig 13) that the test accuracy decreases and leads to overfitting. [This is true for all kernels (*polynomial*, *RBF*), and also going beyond $\gamma = 10^2$ can cause convergence issues for SVM model with *polynomial* kernel.]

Similarly, the search for the parameter C beyond 10^2 is not needed, as it can be observed from the validation curve (Fig 14) that the test accuracy begins to slightly decreases from $C \approx 10^2$. [This is true for all kernels (*linear*, *polynomial*, *RBF*), and also going beyond $C = 10^2$ can cause convergence issues for SVM model with (*polynomial*, *linear*) kernels.]

Kernel Type	Parameters and its values	Total Candidates Evaluated
Linear	C : 7 equidistant point are selected from the range $[10^{-3}, 10^3]$ on the log scale	7
Polynomial	C : 7 equidistant point are selected from the range $[10^{-3}, 10^3]$ on the log scale γ : 7 equidistant point are selected from the range $[10^{-4}, 10^2]$ on the log scale d : [1, 2, 3, 4, 5]	$7 \times 7 \times 5 = 245$
RBF (Radial Basis Function)	C : 7 equidistant point are selected from the range $[10^{-3}, 10^3]$ on the log scale γ : 7 equidistant point are selected from the range $[10^{-4}, 10^2]$ on the log scale	$7 \times 7 = 49$
Best Parameters for this Search - Kernel: RBF, C: 10.0, γ : 0.1		

TABLE XX: SVM Hyperparameter Tuning - I, for Dataset 3

Kernel Type	Parameters and its values	Total Candidates Evaluated
RBF (Radial Basis Function)	C : 10 equidistant point are selected from the range $[1.778, 100.0]$ on the log scale γ : 10 equidistant point are selected from the range $[0.1, 1.0]$ on the log scale	$10 \times 10 = 100$
Best Parameters for this Search - Kernel: RBF, C: 4.35400465, γ : 0.166810053		

TABLE XXI: SVM Hyperparameter Tuning - II (refining the parameters with reduced search space)

Validation Curves for Parameters C and γ :

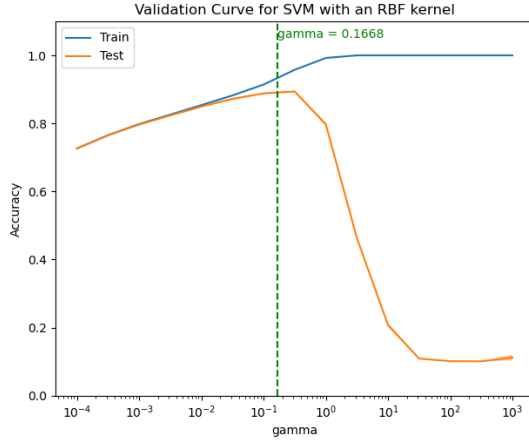


Fig. 13: Validation Curve for SVM with γ parameter

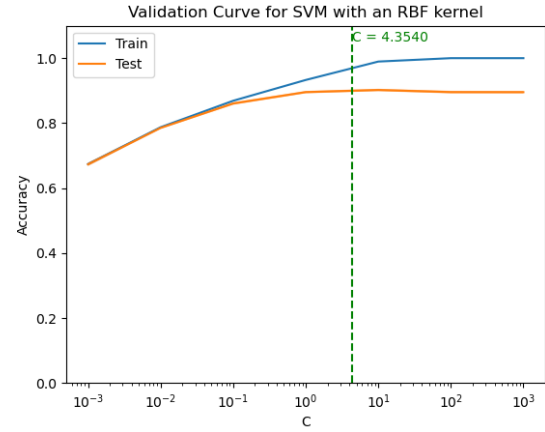


Fig. 14: Validation Curve for SVM with C parameter

Category	0	1	2	3	4	5	6	7	8	9	Average
Precision	0.85	0.99	0.82	0.90	0.83	0.98	0.74	0.95	0.98	0.97	0.90
Recall	0.85	0.97	0.84	0.90	0.84	0.97	0.71	0.97	0.98	0.96	0.90
F1-Score	0.85	0.98	0.83	0.90	0.83	0.98	0.72	0.96	0.98	0.97	0.90
Support	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	10000

TABLE XXII: SVM Model Classification Report for Dataset 3

K. Detailed Results of k -Nearest Neighbors Model for Dataset 3

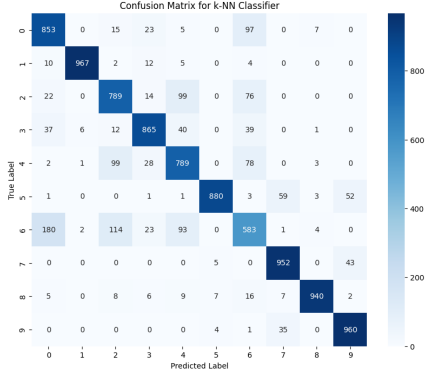


Fig. 15: k-NN Confusion Matrix for Dataset 3

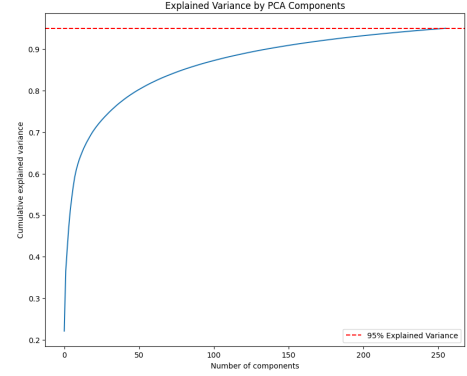


Fig. 16: PCA Component Variance for Dataset 3

Category	0	1	2	3	4	5	6	7	8	9	Average
Precision	0.77	0.99	0.76	0.89	0.76	0.98	0.65	0.90	0.98	0.91	0.86
Recall	0.85	0.97	0.79	0.86	0.79	0.88	0.58	0.95	0.94	0.96	0.86
F1-Score	0.81	0.98	0.77	0.88	0.77	0.93	0.61	0.93	0.96	0.93	0.86
Support	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	10000

TABLE XXIII: k-NN Learning Result for Dataset 3

L. Detailed Results of Neural Network Model for Dataset 3

Tested with 64 filter-convolutional layer, 80-width single hidden layer, batch size of 64, and 30 epochs on the adam optimizer. Training time was 202.93 seconds.

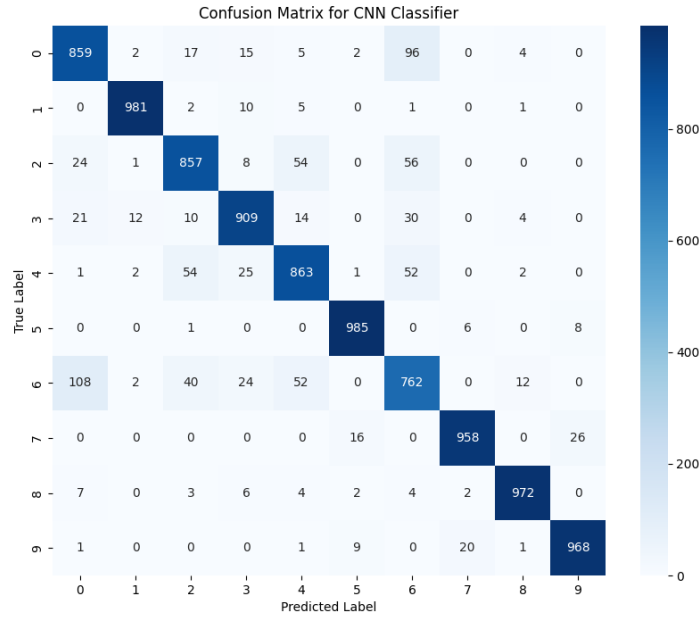


Fig. 17: Dataset 3 Neural Network Confusion Matrix

M. Code used for Dataset 1, Logistic Regression

```
1  ## Code Execution Steps:
2  # - Simply run the logistic_regression.py file. It will print results for each dataset.
3  # - For a cleaner view, run the logistic_regression.ipynb file.
4  # - Here is dataset 1.
5  from sklearn.linear_model import LogisticRegression
6  from sklearn.model_selection import GridSearchCV, cross_val_score
7  import matplotlib.pyplot as plt
8  from sklearn.metrics import classification_report, confusion_matrix
9  import seaborn as sns
10
11 def logreg_pipeline(X_train, X_test, y_train, y_test, dataset='Dataset'):
12     # First Grid Search: L2 and L1
13     param_grid_l2_l1 = {
14         'C': [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 100],
15         'penalty': ['l2', 'l1']
16     }
17
18     # Create Logistic Regression model
19     model = LogisticRegression(solver='saga', max_iter=10000)
20
21     # Perform Grid Search for L2 and L1
22     grid_search_l2_l1 = GridSearchCV(model, param_grid_l2_l1, cv=5, scoring='accuracy')
23     grid_search_l2_l1.fit(X_train, y_train)
24
25     # Second Grid Search: ElasticNet and l1_ratio
26     param_grid_elasticnet = {
27         'C': [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 100],
28         'penalty': ['elasticnet'],
29         'l1_ratio': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
30     }
31
32     # Perform Grid Search for ElasticNet and l1_ratio
33     grid_search_elasticnet = GridSearchCV(model, param_grid_elasticnet, cv=5,
34         scoring='accuracy')
35     grid_search_elasticnet.fit(X_train, y_train)
36
37     # Extract the results
38     results_l2_l1 = grid_search_l2_l1.cv_results_
39     l2_means = results_l2_l1['mean_test_score'][results_l2_l1['param_penalty'] == 'l2']
40     l1_means = results_l2_l1['mean_test_score'][results_l2_l1['param_penalty'] == 'l1']
41
42     results_elasticnet = grid_search_elasticnet.cv_results_
43     best_ratio = grid_search_elasticnet.best_params_['l1_ratio']
44     elastic_means = results_elasticnet['mean_test_score'][results_elasticnet['param_l1_ratio']
45         == best_ratio]
46
47     # Logistic Regression with no regularization
48     model_none = LogisticRegression(solver='saga', max_iter=10000, penalty=None)
49     none_accuracies = cross_val_score(model_none, X_train, y_train, cv=5, scoring='accuracy')
50     none_accuracy = np.mean(none_accuracies)
51
52     # Plot the curves
53     plt.figure(figsize=(12, 8))
54
55     # L2 Average Accuracy Curve
56     plt.plot(param_grid_l2_l1['C'], l2_means, label='L2', marker='o')
57
58     # L1 Average Accuracy Curve
59     plt.plot(param_grid_l2_l1['C'], l1_means, label='L1', marker='o')
60
61     # Best ElasticNet l1_ratio Average Accuracy Curve
62     plt.plot(param_grid_elasticnet['C'], elastic_means, label=f'Elastic-Net (best l1_ratio:
63         {best_ratio:.2f})', linestyle='--', color='red', marker='o')
```

```

63     plt.axhline(y=none_accuracy, color='green', linestyle='--', label='None', linewidth=2)
64
65     # Set plot properties
66     plt.title(f'{dataset} - Logistic Regression Grid Search')
67     plt.xlabel('C', fontsize=14)
68     plt.ylabel('Average Validation Accuracy', fontsize=14)
69     plt.xscale('log')
70     plt.legend(fontsize=14)
71     plt.grid(True)
72     plt.show()
73
74     # Extract the best parameters and scores for each penalty
75     best_params_l2_l1 = grid_search_l2_l1.best_params_
76     best_score_l2_l1 = grid_search_l2_l1.best_score_
77
78     best_params_elasticnet = grid_search_elasticnet.best_params_
79     best_score_elasticnet = grid_search_elasticnet.best_score_
80
81     # Choose the best parameters based on the highest mean test score
82     best_params = max([(best_params_l2_l1, best_score_l2_l1),
83                       (best_params_elasticnet, best_score_elasticnet),
84                       ({'penalty': None, none_accuracy}],
85                       key=lambda x: x[1])[0]
86
87     print("Best Parameters L2/L1:", best_params_l2_l1)
88     print("Best Score L2/L1:", best_score_l2_l1)
89     print("Best Parameters ElasticNet:", best_params_elasticnet)
90     print("Best Score ElasticNet:", best_score_elasticnet)
91     print("Score None:", none_accuracy)
92     print("Best Parameters Overall:", best_params)
93
94     # Fit the best model with the full training set
95     best_model = LogisticRegression(solver='saga', max_iter=10000, **best_params)
96     best_model.fit(X_train, y_train)
97
98     # Evaluate the best model on the test set
99     y_pred = best_model.predict(X_test)
100
101     # Display the classification report and confusion matrix
102     print("Test Classification report:\n", classification_report(y_test, y_pred))
103     print("Test Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
104
105     # Plot the confusion matrix
106     confusion_mat = confusion_matrix(y_test, y_pred)
107     sns.heatmap(confusion_mat, annot=True, fmt='d', cmap='Blues')
108     plt.title(f'{dataset} - Logistic Regression Test Confusion Matrix')
109     plt.xlabel('Predicted Label')
110     plt.ylabel('True Label')
111     plt.show()
112
113     return grid_search_l2_l1, grid_search_elasticnet
114
115 ## Wisconsin Breast Cancer dataset
116 from ucimlrepo import fetch_ucirepo
117 from sklearn.preprocessing import LabelEncoder
118
119 # fetch dataset
120 breast_cancer_wisconsin_diagnostic = fetch_ucirepo(id=17)
121
122 # data (as pandas dataframes)
123 X = breast_cancer_wisconsin_diagnostic.data.features
124 y = breast_cancer_wisconsin_diagnostic.data.targets
125
126 # metadata
127 #print(breast_cancer_wisconsin_diagnostic.metadata)
128
129 # variable information

```



```

130 #print(breast_cancer_wisconsin_diagnostic.variables)
131
132 label_encoder = LabelEncoder()
133
134 X_fix = X.copy()
135 y_fix = y.copy()
136 y_fix['Diagnosis'] = label_encoder.fit_transform(y_fix['Diagnosis'])
137
138 from sklearn.model_selection import train_test_split
139 from sklearn.preprocessing import StandardScaler
140
141 # Split the data into training and testing sets
142 X_train, X_test, y_train, y_test = train_test_split(X_fix, y_fix, test_size=0.2,
143     random_state=42)
144
145 # Standardize the features using StandardScaler
146 scaler = StandardScaler()
147 X_train_scaled = scaler.fit_transform(X_train)
148 X_test_scaled = scaler.transform(X_test)
149
150 cancer_search_l2_l1, cancer_search_elasticnet = logreg_pipeline(X_train_scaled, X_test_scaled,
151     y_train.to_numpy().flatten(), y_test, dataset='Cancer')
152
153 best_cancer_model = LogisticRegression(solver='saga', max_iter=10000,
154     penalty='elasticnet', C=1, l1_ratio=0.8)
155 best_cancer_model.fit(X_train_scaled, y_train.to_numpy().flatten())
156 # Evaluate the best model on the test set
157 y_pred = best_cancer_model.predict(X_test_scaled)
158
159 # Display the classification report and confusion matrix
160 print("Test Classification report:\n", classification_report(y_test, y_pred))
161 print("Test Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
162
163 # Plot the confusion matrix
164 confusion_mat = confusion_matrix(y_test, y_pred)
165 sns.heatmap(confusion_mat, annot=True, fmt='d', cmap='Blues')
166 plt.xlabel('Predicted Label')
167 plt.ylabel('True Label')
168 plt.show()

```

N. Code used for Dataset 1, Support Vector Machines

```
1  ## Code Execution Steps:
2  # - The dataset is already provided in "Data" directory, which is placed parallel to the code.
3  # - If the dataset is not present, download the dataset and place it in the "Data" directory.
4  # - Make sure that the relative path to the dataset is present
5  # - Run the python file. Ex: python ./breastCancer\_svm.py
6  import numpy as np
7  import pandas as pd
8  import matplotlib.pyplot as plt
9  from sklearn.preprocessing import StandardScaler
10 from sklearn.model_selection import train_test_split
11 from sklearn.svm import SVC
12 from sklearn.metrics import classification_report
13 from sklearn.model_selection import GridSearchCV
14 from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score,
15   fl_score
16 import seaborn as sns
17 from sklearn.model_selection import ValidationCurveDisplay
18 from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
19
20 # read data from the breast+cancer+wisconsin+dignostic data set
21   (./breast+cancer+wisconsin+dignostic)
22 wdbcDF = pd.read_csv('./data/breast+cancer+wisconsin+diagnostic/wdbc.data')
23
24 columnNames = [
25     'ID',
26     'Diagnosis',
27     'radius1',
28     'texture1',
29     'perimeter1',
30     'area1',
31     'smoothness1',
32     'compactness1',
33     'concavity1',
34     'concave_points1',
35     'symmetry1',
36     'fractal_dimension1 ',
37     'radius2',
38     'texture2',
39     'perimeter2',
40     'area2',
41     'smoothness2',
42     'compactness2',
43     'concavity2',
44     'concave_points2',
45     'symmetry2',
46     'fractal_dimension2',
47     'radius3',
48     'texture3',
49     'perimeter3',
50     'area3',
51     'smoothness3',
52     'compactness3',
53     'concavity3',
54     'concave_points3',
55     'symmetry3',
56     'fractal_dimension3'
57 ]
58
59 wdbcDF.columns = columnNames
60
61 # drop rows with NaN
62 dropIndex = wdbcDF.isna().sum(axis=1).where(lambda x: x != 0).dropna().index
63 wdbcDF = wdbcDF.drop(dropIndex).reset_index().drop(columns='index')
```

```

63 # sort the ID in ascending order
64 wdbcDF = wdbcDF.sort_values(by='ID', ascending=True).reset_index().drop(columns='index')
65
66 # map the Diagnosis values Malignant (M) = 1 and Benign (B) = 0
67 wdbcDF.loc[wdbcDF['Diagnosis'] == 'M', 'Diagnosis'] = 1
68 wdbcDF.loc[wdbcDF['Diagnosis'] == 'B', 'Diagnosis'] = 0
69 wdbcDF['Diagnosis'] = wdbcDF['Diagnosis'].astype(float)
70
71 # drop ID column
72 wdbcDF = wdbcDF.drop(columns='ID')
73
74 #separate X (data) and Y (label) from the data frame
75 data = wdbcDF.iloc[:,1:].to_numpy()
76 label = wdbcDF.iloc[:,0].to_numpy()
77
78 #use Standard scaler to normalize the data
79 scalar = StandardScaler()
80 data = scalar.fit_transform(data)
81
82 #split the training and test data into 80% to 20% ratio respectively
83 xTrainData, xTestData, yTrainData, yTestData = train_test_split(data, label, test_size=0.2,
84     random_state=23)
85
86 # baseline/default SVM model
87 model = SVC(max_iter=10**8)
88
89 model.fit(xTrainData, yTrainData)
90
91 yPredict = model.predict(xTestData)
92
93 # Score report for the baseline/default SVM model
94 print(classification_report(yTestData,yPredict))
95
96 #Scoring Metrics
97 scoring = {'accuracy' : make_scorer(accuracy_score),
98     'precision' : make_scorer(precision_score, zero_division=0),
99     'recall' : make_scorer(recall_score, zero_division=0),
100     'f1_score' : make_scorer(f1_score, zero_division=0)
101 }
102
103 # hyperparameter tuning and KFold cross validation
104 gammaValues = np.array([10**k for k in np.linspace(-6, 3, 18)])
105 gammaValues = gammaValues.tolist()
106 # Appending 'scale' and 'auto' to the list
107 gammaValues.append('scale')
108 gammaValues.append('auto')
109
110 cValues = [10**k for k in np.linspace(-3, 6, 18)]
111
112 parametersGridCV = [
113     {
114         'kernel' : ['linear'],
115         'C' : cValues
116     },
117     {
118         'kernel' : ['poly'],
119         'C' : cValues,
120         'degree' : list(range(1, 16, 1)),
121         'gamma' : gammaValues
122     },
123     {
124         'kernel' : ['rbf'],
125         'C' : cValues,
126         'gamma' : gammaValues
127     }
128 ]

```

```

129 gridCV = GridSearchCV(estimator=model, param_grid=parametersGridCV, cv=5, scoring=scoring,
    refit='accuracy', verbose=3, n_jobs=-1)
130
131 # fitting the model
132 gridCV.fit(xTrainData, yTrainData)
133 yPredict = gridCV.predict(xTestData)
134
135 # Display the best parameters
136 print(f'Best Parameters : {gridCV.best_params_}')
137
138 # Score report with best parameters after hyperparameter tuning for SVM model
139 print(classification_report(yTestData, yPredict))
140
141 # refine the C and gamma parameter
142 C = gridCV.best_params_['C']
143 gamma = gridCV.best_params_['gamma']
144 refineParametersGridCV = {
145     'kernel' : [gridCV.best_params_['kernel']],
146     'C' : [10**k for k in np.linspace(np.log10(C) - 0.75, np.log10(C) + 2, 1000)],
147     'gamma' : [10**k for k in np.linspace(np.log10(gamma), np.log10(gamma) + 2, 10 )]
148 }
149
150 refineGridCV = GridSearchCV(estimator=model, param_grid=refineParametersGridCV, cv=5,
    scoring=scoring, refit='accuracy', verbose=3, n_jobs=-1)
151
152 # fitting the model
153 refineGridCV.fit(xTrainData, yTrainData)
154
155 # Display the best parameters
156 print(f'Best Parameters : {refineGridCV.best_params_}')
157
158 # predict on the test data
159 yPredict = refineGridCV.predict(xTestData)
160
161 # Score report with best parameters after hyperparameter tuning for SVM model
162 print(classification_report(yTestData, yPredict))
163
164 # Confusion Marix
165 confusionMatrix = confusion_matrix(yTestData, yPredict)
166 dispConfusionMatrix = ConfusionMatrixDisplay(confusion_matrix=confusionMatrix,
    display_labels=['Benign (Class = 0)', 'Malignant (Class = 1)'])
167 dispConfusionMatrix.plot(cmap=plt.cm.Blues)
168 plt.title("Confusion Matrix for SVM Classifier")
169
170 # Display validation curves for training and test
171 gammaValue = 0.017190722018585746
172 validationCurve = ValidationCurveDisplay.from_estimator(
173     SVC(kernel='rbf'),
174     xTrainData,
175
176     yTrainData,
177     param_name="gamma",
178     param_range=np.logspace(-6, 3, 19),
179     score_type="both",
180     n_jobs=2,
181     score_name="Accuracy",
182 )
183
184 validationCurve.ax_.set_title("Validation Curve for SVM with an RBF kernel")
185 validationCurve.ax_.set_xlabel(r"gamma")
186 validationCurve.ax_.set_ylim(0.0, 1.1)
187 validationCurve.ax_.axvline(x=gammaValue, color='g', linestyle='--')
188 validationCurve.ax_.text(gammaValue, 1.05, 'gamma = {:.3f}'.format(gammaValue), color='g')
189 plt.show()
190
191 cValue = 5.080218046913023
192 validationCurve = ValidationCurveDisplay.from_estimator(

```

```
193     SVC(kernel='rbf', gamma=0.01719),
194     xTrainData,
195     yTrainData,
196     param_name="C",
197     param_range=np.logspace(-3, 3, 13),
198     score_type="both",
199     n_jobs=2,
200     score_name="Accuracy",
201 )
202 validationCurve.ax_.set_title("Validation Curve for SVM with an RBF kernel")
203 validationCurve.ax_.set_xlabel(r"C")
204 validationCurve.ax_.set_ylim(0.0, 1.1)
205 validationCurve.ax_.axvline(x=cValue, color='g', linestyle='--')
206 validationCurve.ax_.text(cValue, 1.05, 'C = {:.2f}'.format(cValue), color='g')
207 plt.show()
```

O. Code used for Dataset 1, k-Nearest Neighbors

```
1  ## Code Execution Steps:
2  # - The %pip command will automatically install the dataset.
3  # - This code file was programmed via Google Colab. Change %pip to an appropriate syntax when
4  #   running in environments other than that (i.e., pip or pip3).
5  # - Run the python file "PCA+k\NN.py". Each section of code will provide analysis solution
6  #   for dataset 1, 2 and 3. Here is dataset 1.
7  %pip install ucimlrepo
8  import numpy as np
9  import matplotlib.pyplot as plt
10 import torch, torchvision
11 import pandas as pd
12 import seaborn as sns
13 import time
14 from sklearn.decomposition import PCA
15 from sklearn.impute import SimpleImputer
16 from sklearn.model_selection import train_test_split, GridSearchCV
17 from sklearn.neighbors import KNeighborsClassifier
18 from sklearn.pipeline import Pipeline
19 from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
20 from sklearn.metrics import classification_report, confusion_matrix
21 from sklearn.compose import ColumnTransformer
22 from ucimlrepo import fetch_ucirepo
23
24 # Start the timer
25 start_time = time.time()
26 breast_cancer_wisconsin_diagnostic = fetch_ucirepo(id=17)
27
28 # data (as pandas dataframes)
29 X = breast_cancer_wisconsin_diagnostic.data.features
30 y = breast_cancer_wisconsin_diagnostic.data.targets
31
32 # Step 1: Data Preprocessing
33 scaler = StandardScaler()
34 X_scaled = scaler.fit_transform(X)
35
36 # Step 2: PCA for Dimensionality Reduction
37 pca = PCA(n_components=0.95) # Choosing components that explain 95% of the variance
38 X_pca = pca.fit_transform(X_scaled)
39
40 # Step 3: k-NN Classifier Setup
41 knn = KNeighborsClassifier()
42
43 # Step 4: Cross-Validation and Model Evaluation
44 y_resaped = y.values.ravel() if isinstance(y, pd.DataFrame) else y.ravel()
45 X_train, X_test, y_train, y_test = train_test_split(X_pca, y_resaped, test_size=0.3,
46                                                    random_state=42)
47
48 # Hyperparameter Tuning
49 param_grid = {'n_neighbors': [3, 5, 7, 9, 11]}
50 grid = GridSearchCV(knn, param_grid, cv=5)
51 grid.fit(X_train, y_train)
52 end_time = time.time()
53 training_time = end_time - start_time
54
55 # Best parameters and model evaluation
56 print("Best parameters:", grid.best_params_)
57 print("Classification report:\n", classification_report(y_test, grid.predict(X_test)))
58 print("Confusion matrix:\n", confusion_matrix(y_test, grid.predict(X_test)))
59 print(f"Training time: {training_time:.2f} seconds")
60 sns.heatmap(confusion_matrix(y_test, grid.predict(X_test)), annot=True)
61 plt.title('Confusion Matrix for k-NN Classifier')
62 plt.xlabel('Predicted Label')
63 plt.ylabel('True Label')
64 plt.show()
```

P. Code used for Dataset 1, Neural Networks

```
1  ## Code Execution Steps:
2  # - Run "pip3 install numpy matplotlib seaborn multiprocessing tensorflow scikit-learn
   ucimlrepo itertools pandas csv"
3  # - Modify Hyperparameter lists to explore different grid search arrangements.
4  # - Run python file using any Python >3.9 interpreter with the above libraries installed.
5  import numpy as np
6  import numpy.random as random
7  import matplotlib.pyplot as plt
8  import seaborn as sns
9  import multiprocessing
10 import os
11 import tensorflow as tf
12 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3' # Suppresses noisy outputs
13 tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.FATAL)
14 from sklearn.preprocessing import StandardScaler
15 from sklearn.model_selection import KFold
16 from ucimlrepo import fetch_ucirepo
17 import itertools
18 import pandas as pd
19 import csv
20 import time
21
22 def main():
23     ## Hyperparameters
24     PROPORTION_TRAIN = 0.8
25     LAYER Depths = [10, 20, 40, 80, 160]
26     HIDDEN_LAYERS = [1, 2]
27     EPOCHS = 100
28     BATCH_SIZES = [1, 2, 4, 8, 16, 32, 64]
29     ACTIVATION_FUNCTIONS = "relu"
30     LOSS_FUNCTION = "binary_crossentropy"
31     OPTIMIZERS = ["SGD", "rmsprop", "adam", "adadelta", "ftrl"]
32
33     filename = "FNN Results.csv"
34     n_folds = 5
35     kf = KFold(n_splits=n_folds)
36
37     ## Data Pre-Processing
38     #region
39     breast_cancer_wisconsin_diagnostic = fetch_ucirepo(id=17)
40     # data (as pandas dataframes)
41     X = breast_cancer_wisconsin_diagnostic.data.features
42     y = breast_cancer_wisconsin_diagnostic.data.targets
43
44     # metadata
45     # print(breast_cancer_wisconsin_diagnostic.metadata)
46     # variable information
47     # print(breast_cancer_wisconsin_diagnostic.variables)
48
49     # Convert data to numpy
50     X_combined = X.to_numpy()
51     y_combined = y.to_numpy()
52
53     # Labels are either 'M' or 'B', converting to 1 and 0.
54     y_combined = np.where(y_combined == 'M', 1, 0)
55
56     NUM = X_combined.shape[0]
57     DIMS = X_combined.shape[1]
58
59     # Shuffle data
60     order_all = random.permutation(NUM)
61     X_combined = X_combined[order_all,:]
62     y_combined = y_combined[order_all,:]
63
64     # Split data by train and test.
```

```

65 splitIndex = int(PROPORTION_TRAIN * NUM)
66 X_train = X_combined[:splitIndex,:] # Shape 455x30
67 y_train = y_combined[:splitIndex,:] # Shape 455x1
68 X_test = X_combined[splitIndex:,:] # Shape 114x30
69 y_test = y_combined[splitIndex:,:] # Shape 114x1
70
71 # Apply scaling to feature data, on the basis of the training dataset
72 scaler = StandardScaler()
73 X_train = scaler.fit_transform(X_train)
74 X_test = scaler.transform(X_test)
75
76 #endregion
77
78 #region
79 # Prepare grid search over all combinations of hyperparameters
80 num_param_states = len(LAYER_DEPTHS) * len(HIDDEN_LAYERS) * len(BATCH_SIZES) *
    len(OPTIMIZERS)
81 combinations = list(itertools.product(LAYER_DEPTHS, HIDDEN_LAYERS, BATCH_SIZES,
    OPTIMIZERS))
82 # Append indices to this large list
83 combinations = [(x + (i,)) for i, x in enumerate(combinations)]
84
85 # Prepare gigantic list of complete parameters.
86 args_full = [(combo, DIMS, ACTIVATION_FUNCTIONS, LOSS_FUNCTION, X_train, y_train, EPOCHS,
    kf, n_folds) for combo in combinations]
87
88 print(f"ready to iterate over {num_param_states} combinations")
89
90 # Start running parallel trainings over all combinations
91 outputData = []
92 with multiprocessing.Pool() as pool:
93     results = pool.starmap(eval_model, args_full)
94
95     # Open the CSV file once and write all results
96     with open(filename, 'w', newline='') as csvfile:
97         writer = csv.writer(csvfile, delimiter=',')
98         header = [
99             ["Index"], ["Depth"], ["Layers"], ["Batch Size"], ["Optimizer"], ["Valid
100             Loss"], ["Valid Acc"], ["Time"]]
101         writer.writerow(header)
102         for result in results:
103             outputData.append(result)
104             writer.writerow(result)
105
106 #endregion
107
108 # Pick top 1 in terms of validation loss and get test error
109 sorted_outputData = sorted(outputData, key=lambda x: x[5])
110 top_1 = sorted_outputData[0]
111 index, depth, hLayers, batchSize, optimizer, _, _, _ = top_1
112 result = test_model((depth, hLayers, batchSize, optimizer, index), DIMS,
    ACTIVATION_FUNCTIONS, LOSS_FUNCTION, X_train, y_train, X_test, y_test, EPOCHS)
113 print("Index, Depth, Layers, Batch Size, Optimizer, Test Loss, Test Accuracy, Training
    Time")
114 print(result)
115
116 # Function to train and evaluate TF model
117 def eval_model(dynamic_args, d, activation, loss, X_train, y_train, epochs, kf, n_folds):
118     # Unpack arguments
119     depth, hLayers, batchSize, optimizer, index = dynamic_args
120     # Clear previous models and graphs
121     tf.keras.backend.clear_session()
122
123     # Create a new graph for the current process
124     with tf.Graph().as_default():
125         # Create a session for the current graph
126         with tf.compat.v1.Session() as sess:
127             # Remind Tensorflow to keep quiet

```



```

126 tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.FATAL)
127
128 # Model creation
129 if hLayers == 1:
130     model = tf.keras.models.Sequential([
131         tf.keras.layers.Dense(depth, activation=activation, input_shape=(d,)),
132         tf.keras.layers.Dense(1, activation='sigmoid')
133     ])
134 else:
135     model = tf.keras.models.Sequential([
136         tf.keras.layers.Dense(depth, activation=activation, input_shape=(d,)),
137         tf.keras.layers.Dense(depth, activation=activation),
138         tf.keras.layers.Dense(1, activation='sigmoid')
139     ])
140
141 # Compile the model
142 model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
143
144 # Start stopwatch
145 start_time = time.time()
146
147 # Initialize averaged results, test results are to be kept aside.
148 valid_accuracy = 0
149 valid_loss = 0
150 # Train the model
151 for index_train, index_val in kf.split(X_train):
152     X_vtrain, X_val = X_train[index_train], X_train[index_val]
153     y_vtrain, y_val = y_train[index_train], y_train[index_val]
154     history = model.fit(X_vtrain, y_vtrain, epochs=epochs, batch_size=batchSize,
155                         validation_split=0.2)
156     # Test this model on the current fold's validation dataset, and the test set
157     this_valid_loss, this_valid_accuracy = model.evaluate(X_val, y_val)
158
159     # Add to average
160     valid_accuracy += this_valid_accuracy / n_folds
161     valid_loss += this_valid_loss / n_folds
162
163 # Stop stopwatch
164 end_time = time.time()
165
166 training_time = end_time - start_time
167 print(f"Done with Run {index}, t = {training_time}")
168 return index, depth, hLayers, batchSize, optimizer, valid_loss, valid_accuracy,
169     training_time
170
171 def test_model(dynamic_args, d, activation, loss, X_train, y_train, X_test, y_test, epochs):
172     # Unpack arguments
173     depth, hLayers, batchSize, optimizer, index = dynamic_args
174     # Clear previous models and graphs
175     tf.keras.backend.clear_session()
176
177     # Create a new graph for the current process
178     with tf.Graph().as_default():
179         # Create a session for the current graph
180         with tf.compat.v1.Session() as sess:
181             # Remind Tensorflow to keep quiet
182             tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.FATAL)
183
184             # Model creation
185             if hLayers == 1:
186                 model = tf.keras.models.Sequential([
187                     tf.keras.layers.Dense(depth, activation=activation, input_shape=(d,)),
188                     tf.keras.layers.Dense(1, activation='sigmoid')
189                 ])
190             else:
191                 model = tf.keras.models.Sequential([

```

```

191         tf.keras.layers.Dense(depth, activation=activation, input_shape=(d,)),
192         tf.keras.layers.Dense(depth, activation=activation),
193         tf.keras.layers.Dense(1, activation='sigmoid')
194     ])
195
196     # Compile the model
197     model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
198
199     # Start stopwatch
200     start_time = time.time()
201
202     # Train the model
203     history = model.fit(X_train, y_train, epochs=epochs, batch_size = batchSize,
204                        validation_split=0.2)
205     test_loss, test_accuracy = model.evaluate(X_test, y_test)
206     # Stop stopwatch
207     end_time = time.time()
208
209     training_time = end_time - start_time
210     print(f"Done with test, t = {training_time}")
211     return index, depth, hLayers, batchSize, optimizer, test_loss, test_accuracy,
212           training_time
213
214 if __name__ == "__main__":
215     main()

```

```

1  ## Code Execution Steps:
2  # - If you haven't already, run "pip3 install numpy matplotlib seaborn multiprocessing
   tensorflow scikit-learn ucimlrepo itertools pandas csv"
3  # - Modify individual hyperparameters to evaluate them on the test set.
4  # - Run python file using any Python >3.9 interpreter with the above libraries installed.
5  import numpy as np
6  import numpy.random as random
7  import os
8  import tensorflow as tf
9  os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3' # Suppresses noisy outputs
10 tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.FATAL)
11 from sklearn.preprocessing import StandardScaler
12 from sklearn.metrics import confusion_matrix, classification_report
13 from ucimlrepo import fetch_ucirepo
14 import pandas as pd
15 import csv
16 import time
17
18 def main():
19     ## Hyperparameters
20     PROPORTION_TRAIN = 0.8
21     layer_depth = 20
22     hidden_layers = 2
23     epochs = 100
24     batch_size = 2
25     activation_function = "relu"
26     loss_function = "binary_crossentropy"
27     optimizer = "adam"
28
29     # filename = "FNNBC Evaluation Results.csv"
30
31     ## Data Pre-Processing
32     #region
33     breast_cancer_wisconsin_diagnostic = fetch_ucirepo(id=17)
34     # data (as pandas dataframes)
35     X = breast_cancer_wisconsin_diagnostic.data.features
36     y = breast_cancer_wisconsin_diagnostic.data.targets
37
38     # metadata
39     # print(breast_cancer_wisconsin_diagnostic.metadata)

```

```

40     # variable information
41     # print(breast_cancer_wisconsin_diagnostic.variables)
42
43     # Convert data to numpy
44     X_combined = X.to_numpy()
45     y_combined = y.to_numpy()
46
47     # Labels are either 'M' or 'B', converting to 1 and 0.
48     y_combined = np.where(y_combined == 'M', 1, 0)
49
50     NUM = X_combined.shape[0]
51     DIMS = X_combined.shape[1]
52
53     # Shuffle data
54     order_all = random.permutation(NUM)
55     X_combined = X_combined[order_all,:]
56     y_combined = y_combined[order_all,:]
57
58     # Split data by train and test.
59     splitIndex = int(PROPORTION_TRAIN * NUM)
60     X_train = X_combined[:splitIndex,:] # Shape 455x30
61     y_train = y_combined[:splitIndex,:] # Shape 455x1
62     X_test = X_combined[splitIndex:,:] # Shape 114x30
63     y_test = y_combined[splitIndex:,:] # Shape 114x1
64
65     # Apply scaling to feature data, on the basis of the training dataset
66     scaler = StandardScaler()
67     X_train = scaler.fit_transform(X_train)
68     X_test = scaler.transform(X_test)
69
70     #endregion
71
72     result = test_model((layer_depth, hidden_layers, batch_size, optimizer), DIMS,
73         activation_function, loss_function, X_train, y_train, X_test, y_test, epochs)
74     depth, hLayers, batchSize, optimizer, test_loss, test_accuracy, training_time, conmat,
75     y_pred = result
76
77     print("Depth, Layers, Batch Size, Optimizer, Test Loss, Test Accuracy, Training Time")
78     print(f"{depth}, {hLayers}, {batchSize}, {optimizer}, {test_loss}, {test_accuracy},
79         {training_time}")
80
81     # Print confusion matrix
82     print("Confusion Matrix:")
83     print(conmat)
84
85     # Compute and print classification report
86     target_names = ['Class 0', 'Class 1']
87     report = classification_report(y_test, (y_pred > 0.5), target_names=target_names)
88     print("Classification Report:")
89     print(report)
90
91     def test_model(dynamic_args, d, activation, loss, X_train, y_train, X_test, y_test, epochs):
92         # Unpack arguments
93         depth, hLayers, batchSize, optimizer = dynamic_args
94         # Clear previous models and graphs
95         tf.keras.backend.clear_session()
96
97         # Create a new graph for the current process
98         with tf.Graph().as_default():
99             # Create a session for the current graph
100             with tf.compat.v1.Session() as sess:
101                 # Remind Tensorflow to keep quiet
102                 tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.FATAL)
103
104                 # Model creation
105                 if hLayers == 1:

```

```

104         model = tf.keras.models.Sequential([
105             tf.keras.layers.Dense(depth, activation=activation, input_shape=(d,)),
106             tf.keras.layers.Dense(1, activation='sigmoid')
107         ])
108     else:
109         model = tf.keras.models.Sequential([
110             tf.keras.layers.Dense(depth, activation=activation, input_shape=(d,)),
111             tf.keras.layers.Dense(depth, activation=activation),
112             tf.keras.layers.Dense(1, activation='sigmoid')
113         ])
114
115     # Compile the model
116     model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
117
118     # Start stopwatch
119     start_time = time.time()
120
121     # Train the model
122     history = model.fit(X_train, y_train, epochs=epochs, batch_size = batchSize,
123                        validation_split=0.2)
124     test_loss, test_accuracy = model.evaluate(X_test, y_test)
125
126     # Predict the test set
127     y_pred = model.predict(X_test)
128
129     # Compute confusion matrix
130     conmat = confusion_matrix(y_test, (y_pred > 0.5))
131
132     # Stop stopwatch
133     end_time = time.time()
134
135     training_time = end_time - start_time
136     print(f"Done with test, t = {training_time}")
137
138     return depth, hLayers, batchSize, optimizer, test_loss, test_accuracy,
139           training_time, conmat, y_pred
140
141 if __name__ == "__main__":
142     main()

```

Q. Code used for Dataset 2, Logistic Regression

```
1  ## Code Execution Steps:
2  # - This is an extension of the Code from Dataset 1, as it uses the same imports and function.
3
4  ## UCI Adult Dataset
5  from ucimlrepo import fetch_ucirepo
6
7  # fetch dataset
8  adult = fetch_ucirepo(id=2)
9
10 # data (as pandas dataframes)
11 X = adult.data.features
12 y = adult.data.targets
13
14 # metadata
15 #print(adult.metadata)
16
17 # variable information
18 #print(adult.variables)
19 X_copy = X.copy()
20 X_copy.replace('?', np.nan, inplace=True)
21 nan_indices = X_copy[X_copy.isnull().any(axis=1)].index
22 X_fix = X_copy.dropna()
23 y_fix = y.drop(nan_indices)
24 X_fix = X_fix.reset_index(drop=True)
25 y_fix = y_fix.reset_index(drop=True)
26
27 remove_periods = lambda x: x.replace('.', '') if isinstance(x, str) else x
28 X_fix = X_fix.applymap(remove_periods)
29 y_fix = y_fix.applymap(remove_periods)
30
31 # Initialize LabelEncoder
32 label_encoder = LabelEncoder()
33
34 # Identify categorical columns
35 categorical_columns = X_fix.select_dtypes(include=['object']).columns
36
37 # Encode categorical columns
38 for col in categorical_columns:
39     X_fix[col] = label_encoder.fit_transform(X_fix[col])
40
41 y_fix['income'] = label_encoder.fit_transform(y_fix['income'])
42
43 # Split the data into training and testing sets
44 X_train, X_test, y_train, y_test = train_test_split(X_fix, y_fix, test_size=0.2,
45     random_state=42)
46
47 # Standardize the features using StandardScaler
48 scaler = StandardScaler()
49 X_train_scaled = scaler.fit_transform(X_train)
50 X_test_scaled = scaler.transform(X_test)
51
52 adult_search_l2_l1, adult_search_elasticnet = logreg_pipeline(X_train_scaled, X_test_scaled,
53     y_train.to_numpy().flatten(), y_test, dataset='Adult')
```

R. Code used for Dataset 2, Support Vector Machines

```
1  ## Code Execution Steps:
2  # - The dataset is already provided in "Data" directory, which is placed parallel to the code.
3  # - If the dataset is not present, download the dataset and place it in the "Data" directory.
4  # - Make sure that the relative path to the dataset is present "./data/adult/adult.data" and
   "    ./data/adult/adult.test"
5  # - Run the python file. Ex: python ./adult\_svm.py
6  import numpy as np
7  import pandas as pd
8  import matplotlib.pyplot as plt
9  from sklearn.preprocessing import StandardScaler
10 from sklearn.model_selection import train_test_split
11 from sklearn.svm import SVC
12 from sklearn.metrics import classification_report
13 from sklearn.model_selection import GridSearchCV
14 from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score,
   fl_score
15 from sklearn.preprocessing import LabelEncoder
16 from sklearn.model_selection import ValidationCurveDisplay
17 from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
18
19 # column names
20 columnNames = [
21     'age',
22     'workclass',
23     'fnlwgt',
24     'education',
25     'education-num',
26     'marital-status',
27     'occupation',
28     'relationship',
29     'race',
30     'sex',
31     'capital-gain',
32     'capital-loss',
33     'hours-per-week',
34     'native-country',
35     'income'
36 ]
37
38 # read data from the Census Income data set (./adult.data , ./adult.test)
39 adultTrainDF = pd.read_csv('./data/adult/adult.data', names=columnNames)
40 adultTestDF = pd.read_csv('./data/adult/adult.test', names=columnNames)
41 adultTestDF = adultTestDF.drop(0)
42
43 # handle missing values in the data set
44 # Strip leading/trailing whitespace
45 adultTrainDF = adultTrainDF.applymap(lambda x: x.strip() if isinstance(x, str) else x)
46 adultTrainDF.replace('?', np.NaN, inplace=True)
47 adultTestDF = adultTestDF.applymap(lambda x: x.strip() if isinstance(x, str) else x)
48 adultTestDF.replace('?', np.NaN, inplace=True)
49
50 # drop rows with NaN
51 dropIndex = adultTrainDF.isna().sum(axis=1).where(lambda x: x != 0).dropna().index
52 adultTrainDF = adultTrainDF.drop(dropIndex).reset_index().drop(columns='index')
53 dropIndex = adultTestDF.isna().sum(axis=1).where(lambda x: x != 0).dropna().index
54 adultTestDF = adultTestDF.drop(dropIndex).reset_index().drop(columns='index')
55
56 # map the adult income (>50K) = 1 and income (<=50K) = 0
57 adultTrainDF.loc[adultTrainDF['income'] == '>50K', 'income'] = 1
58 adultTrainDF.loc[adultTrainDF['income'] == '<=50K', 'income'] = 0
59 adultTrainDF['income'] = adultTrainDF['income'].astype(float)
60
61 adultTestDF.loc[adultTestDF['income'] == '>50K.', 'income'] = 1
62 adultTestDF.loc[adultTestDF['income'] == '<=50K.', 'income'] = 0
63 adultTestDF['income'] = adultTestDF['income'].astype(float)
```

```

64
65 # encode the categorical data
66 encodeColumnsList = ['workclass', 'education', 'marital-status', 'occupation', 'relationship',
67                      'race', 'sex', 'native-country']
68 encoder = LabelEncoder()
69 for column in encodeColumnsList:
70     adultTrainDF[column] = encoder.fit_transform(adultTrainDF[column])
71     adultTestDF[column] = encoder.fit_transform(adultTestDF[column])
72
73 #separate X (data) and Y (label) from the data frame
74 xTrainData = adultTrainDF.iloc[:, :-1].to_numpy()
75 yTrainData = adultTrainDF.iloc[:, -1].to_numpy()
76 xTestData = adultTestDF.iloc[:, :-1].to_numpy()
77 yTestData = adultTestDF.iloc[:, -1].to_numpy()
78
79 #use Standard scaler to normalize the data
80 scalar = StandardScaler()
81 xTrainData = scalar.fit_transform(xTrainData)
82 xTestData = scalar.fit_transform(xTestData)
83
84 # baseline/default SVM model
85 model = SVC(max_iter=10**7)
86
87 model.fit(xTrainData, yTrainData)
88
89 yPredict = model.predict(xTestData)
90
91 # Score report for the baseline/default SVM model
92 print(classification_report(yTestData, yPredict))
93
94 #Scoring Metrics
95 scoring = {'accuracy' : make_scorer(accuracy_score),
96           'precision' : make_scorer(precision_score, zero_division=0),
97           'recall' : make_scorer(recall_score, zero_division=0),
98           'f1_score' : make_scorer(f1_score, zero_division=0)}
99
100 # hyperparameter tuning and KFold cross validation
101 gammaValues = np.array([10**k for k in np.linspace(-5, 3, 17)])
102 gammaValues = gammaValues.tolist()
103 # Appending 'scale' and 'auto' to the list
104 gammaValues.append('scale')
105 gammaValues.append('auto')
106
107 polyGammaValues1 = np.array([10**k for k in np.linspace(-5, 2, 8)])
108 polyGammaValues1 = polyGammaValues1.tolist()
109 polyGammaValues1.append('scale')
110 polyGammaValues1.append('auto')
111
112 polyGammaValues2 = np.array([10**k for k in np.linspace(-5, 2, 8)])
113 polyGammaValues2 = polyGammaValues2.tolist()
114 polyGammaValues2.append('scale')
115 polyGammaValues2.append('auto')
116
117 cValues = [10**k for k in np.linspace(-3, 2, 11)]
118
119 parametersGridCV = [
120     {
121         'kernel' : ['linear'],
122         'C' : cValues
123     },
124     {
125         'kernel' : ['poly'],
126         'C' : cValues,
127         'degree' : list(range(1, 5, 1)),
128         'gamma' : polyGammaValues2
129     },

```

```

130     {
131         'kernel' : ['poly'],
132         'C' : cValues,
133         'degree' : list(range(5, 11, 1)),
134         'gamma' : polyGammaValues2
135     },
136     {
137         'kernel' : ['rbf'],
138         'C' : cValues,
139         'gamma' : gammaValues
140     }
141 ]
142
143 gridCV = GridSearchCV(estimator=model, param_grid=parametersGridCV, cv=5, scoring=scoring,
144                       refit='accuracy', verbose=3, n_jobs=-1)
145
146 # fitting the model
147 gridCV.fit(xTrainData, yTrainData)
148 yPredict = gridCV.predict(xTestData)
149
150 # Display the best parameters
151 print(f'Best Parameters : {gridCV.best_params_}')
152
153 # Score report with best parameters after hyperparameter tuning for SVM model
154 print(classification_report(yTestData, yPredict))
155
156 # refine the C and gamma parameter
157 C = gridCV.best_params_['C']
158 gamma = gridCV.best_params_['gamma']
159 refineParametersGridCV = {
160     'kernel' : [gridCV.best_params_['kernel']],
161     'C' : [10**k for k in np.linspace(np.log10(C) - 0.75, np.log10(C) + 1, 100)],
162     'gamma' : [10**k for k in np.linspace(np.log10(gamma), np.log10(gamma) + 2, 10)]
163 }
164
165 refineGridCV = GridSearchCV(estimator=model, param_grid=refineParametersGridCV, cv=5,
166                             scoring=scoring, refit='accuracy', verbose=3, n_jobs=-1)
167
168 # fitting the model
169 refineGridCV.fit(xTrainData, yTrainData)
170
171 # Display the best parameters
172 print(f'Best Parameters : {refineGridCV.best_params_}')
173
174 # predict on the test data
175 yPredict = refineGridCV.predict(xTestData)
176
177 # Score report with best parameters after hyperparameter tuning for SVM model
178 print(classification_report(yTestData, yPredict))
179
180 # Confusion Marix
181 confusionMatrix = confusion_matrix(yTestData, yPredict)
182 dispConfusionMatrix = ConfusionMatrixDisplay(confusion_matrix=confusionMatrix,
183                                               display_labels=['Income (<=50K): (Class = 0)', 'Income (>50K): (Class = 1)'])
184 dispConfusionMatrix.plot(cmap=plt.cm.Blues)
185 plt.title("Confusion Matrix for SVM Classifier on Dataset 2")
186
187 # Display validation curves for training and test
188 gammaValue = 0.1
189 validationCurve = ValidationCurveDisplay.from_estimator(
190     SVC(kernel='rbf'),
191     xTrainData,
192     yTrainData,
193     param_name="gamma",
194     param_range=np.logspace(-5, 3, 17),
195     score_type="both",
196     n_jobs=-1,

```



```

194     score_name="Accuracy",
195 )
196 validationCurve.ax_.set_title("Validation Curve for SVM with an RBF kernel")
197 validationCurve.ax_.set_xlabel(r"gamma")
198 validationCurve.ax_.set_ylim(0.0, 1.1)
199 validationCurve.ax_.axvline(x=gammaValue, color='g', linestyle='--')
200 validationCurve.ax_.text(gammaValue, 1.05, 'gamma = {:.3f}'.format(gammaValue), color='g')
201 plt.show()
202
203 cValue = 1.906790722960592
204 validationCurve = ValidationCurveDisplay.from_estimator(
205     SVC(kernel='rbf', gamma=0.1),
206     xTrainData,
207     yTrainData,
208     param_name="C",
209     param_range=np.logspace(-3, 3, 13),
210     score_type="both",
211     n_jobs=-1,
212     score_name="Accuracy",
213 )
214 validationCurve.ax_.set_title("Validation Curve for SVM with an RBF kernel")
215 validationCurve.ax_.set_xlabel(r"C")
216 validationCurve.ax_.set_ylim(0.0, 1.1)
217 validationCurve.ax_.axvline(x=cValue, color='g', linestyle='--')
218 validationCurve.ax_.text(cValue, 1.05, 'C = {:.3f}'.format(cValue), color='g')
219 plt.show()

```

S. Code used for Dataset 2, k-Nearest Neighbors

```
1 ## Code Execution Steps:
2 # - If you already changed %pip to an appropriate syntax in previous section, then just click
  RUN.
3 # - Run the python file "PCA+k_NN.py". Each section of code will provide analysis solution
  for dataset 1, 2 and 3. Here is dataset 2.
4 %pip install ucimlrepo
5 import numpy as np
6 import matplotlib.pyplot as plt
7 import torch, torchvision
8 import pandas as pd
9 import seaborn as sns
10 import time
11 from sklearn.decomposition import PCA
12 from sklearn.impute import SimpleImputer
13 from sklearn.model_selection import train_test_split, GridSearchCV
14 from sklearn.neighbors import KNeighborsClassifier
15 from sklearn.pipeline import Pipeline
16 from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
17 from sklearn.metrics import classification_report, confusion_matrix
18 from sklearn.compose import ColumnTransformer
19 from ucimlrepo import fetch_ucirepo
20
21 # Start the timer
22 start_time = time.time()
23
24 # fetch dataset
25 adult = fetch_ucirepo(id=2)
26
27 # data (as pandas dataframes)
28 X = adult.data.features
29 y = adult.data.targets
30
31 # Handle missing values, 'None' is used for missing values, replace them with NaN
32 X.replace('None', np.nan, inplace=True)
33
34 # Drop rows with any NaN values
35 X.dropna(inplace=True)
36 y = y.loc[X.index] # Align y with the rows kept in X
37
38 # Since we've removed rows, the indices may need to be reset
39 X.reset_index(drop=True, inplace=True)
40 y.reset_index(drop=True, inplace=True)
41
42 # Remove periods from the target variable
43 remove_periods = lambda x: x.replace('.', '') if isinstance(x, str) else x
44 y = y.applymap(remove_periods)
45
46 # One-hot encode the categorical variables
47 categorical_features = X.select_dtypes(include=['object']).columns
48 one_hot_encoder = OneHotEncoder(handle_unknown='ignore')
49 X_encoded = one_hot_encoder.fit_transform(X[categorical_features]).toarray()
50
51 # Standardize the numerical variables
52 numerical_features = X.select_dtypes(include=['float64', 'int64']).columns
53 scaler = StandardScaler()
54 X_scaled = scaler.fit_transform(X[numerical_features])
55
56 # Combine the scaled and encoded features
57 X_processed = np.hstack((X_scaled, X_encoded))
58
59 # Encode the target variable
60 label_encoder = LabelEncoder()
61 y_encoded = label_encoder.fit_transform(y.squeeze())
62
63 # Split the dataset into training and testing sets
```

```

64 X_train, X_test, y_train, y_test = train_test_split(X_processed, y_encoded, test_size=0.3,
    random_state=42)
65
66 # Create the PCA + k-NN pipeline
67 pipeline = Pipeline([
68     ('scaler', StandardScaler()),
69     ('pca', PCA(n_components=0.95)),
70     ('knn', KNeighborsClassifier())
71 ])
72
73 # Hyperparameter tuning
74 param_grid = {
75     'knn__n_neighbors': [3, 5, 7, 9, 11]
76 }
77 grid = GridSearchCV(pipeline, param_grid, cv=5)
78 grid.fit(X_train, y_train)
79 end_time = time.time()
80 training_time = end_time - start_time
81
82 # Best parameters and model evaluation
83 print("Best parameters:", grid.best_params_)
84 print("Classification report:\n", classification_report(y_test, grid.predict(X_test)))
85 print("Confusion matrix:\n", confusion_matrix(y_test, grid.predict(X_test)))
86 print(f"Training time: {training_time:.2f} seconds")
87
88 plt.figure(figsize=(10, 7))
89 confusion_mat = confusion_matrix(y_test, grid.predict(X_test))
90 sns.heatmap(confusion_mat, annot=True, fmt='d', cmap='Blues')
91 plt.title('Confusion Matrix for k-NN Classifier')
92 plt.xlabel('Predicted Label')
93 plt.ylabel('True Label')
94 plt.show()
95
96 # Extract the PCA from the pipeline
97 pca = grid.best_estimator_.named_steps['pca']
98
99 # Plot the explained variance for each PCA component
100 plt.figure(figsize=(10, 7))
101 plt.bar(range(1, len(pca.explained_variance_ratio_) + 1), pca.explained_variance_ratio_)
102 plt.xlabel('Component Number')
103 plt.ylabel('Proportion of Variance Explained')
104 plt.title('PCA Component Variance')
105 plt.show()

```

T. Code used for Dataset 2, Neural Networks

```
1  ## Code Execution Steps:
2  # - Run "pip3 install numpy matplotlib seaborn multiprocessing tensorflow scikit-learn
   ucimlrepo itertools pandas csv"
3  # - Modify Hyperparameter lists to explore different grid search arrangements.
4  # - Run python file using any Python >3.9 interpreter with the above libraries installed.
5  import numpy as np
6  import numpy.random as random
7  import matplotlib.pyplot as plt
8  import seaborn as sns
9  import torch, torchvision
10 import multiprocessing
11 import os
12 import tensorflow as tf
13 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3' # Suppresses noisy outputs
14 tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.FATAL)
15 from sklearn.preprocessing import StandardScaler, OneHotEncoder
16 from sklearn.compose import ColumnTransformer
17 from sklearn.pipeline import Pipeline
18 from sklearn.model_selection import KFold
19 from ucimlrepo import fetch_ucirepo
20 import itertools
21 import pandas as pd
22 import csv
23 import time
24
25 def main():
26     ## Hyperparameters
27     PROPORTION_TRAIN = 0.8
28     LAYER Depths = [10, 20, 40, 80, 160, 320, 640]
29     HIDDEN_LAYERS = [1, 2]
30     EPOCHS = 100
31     BATCH_SIZES = [16, 32, 64, 128]
32     ACTIVATION_FUNCTIONS = "relu"
33     LOSS_FUNCTION = "binary_crossentropy"
34     OPTIMIZERS = ["SGD", "rmsprop", "adam", "adadelta", "ftrl"]
35
36     filename = "FNN Results UCI.csv"
37     n_folds = 5
38     kf = KFold(n_splits=n_folds)
39
40     ## Data Pre-Processing
41     #region
42     adult = fetch_ucirepo(id=2)
43     # data (as pandas dataframes)
44     X = adult.data.features
45     y = adult.data.targets
46
47     # metadata
48     # print(adult.metadata)
49     # variable information
50     # print(adult.variables)
51
52     # Get rid of NaNs
53     combined = pd.concat([X, y], axis=1)
54     combined = combined.dropna()
55     X = combined.iloc[:, :-1]
56     y = combined.iloc[:, -1]
57
58     NUM = X.shape[0]
59
60     # Shuffle Data
61     combined_shuffled = combined.sample(frac=1).reset_index(drop=True)
62     X_shuffled = combined_shuffled.iloc[:, :-1]
63     y_shuffled = combined_shuffled.iloc[:, -1]
64
```

```

65 # Split data by train and test.
66 splitIndex = int(PROPORTION_TRAIN * NUM)
67 X_train_pre = X_shuffled.iloc[:splitIndex, :] # Shape 38096x14 (108 for onehot)
68 y_train_pre = y_shuffled.iloc[:splitIndex] # Shape 38096
69 X_test_pre = X_shuffled.iloc[splitIndex:, :] # Shape 9525x14 (108 for onehot)
70 y_test_pre = y_shuffled.iloc[splitIndex:] # Shape 9525
71
72 # Convert categorical features to numerical
73 categorical_cols = X.select_dtypes(include=['object', 'category']).columns
74 numerical_cols = X.select_dtypes(include=['number']).columns
75 # Preprocessing for numerical data
76 numerical_transformer = StandardScaler()
77
78 # Preprocessing for categorical data
79 categorical_transformer = OneHotEncoder(handle_unknown='ignore')
80
81 # Bundle preprocessing for numerical and categorical data
82 preprocessor = ColumnTransformer(
83     transformers=[
84         ('num', numerical_transformer, numerical_cols),
85         ('cat', categorical_transformer, categorical_cols)
86     ])
87
88 # Create a preprocessing and modeling pipeline
89 clf = Pipeline(steps=[('preprocessor', preprocessor)])
90
91 # Preprocess the features. Xs are in compressed sparse matrix format (sample, feature),
92 # value
93 X_train = clf.fit_transform(X_train_pre)
94 X_test = clf.transform(X_test_pre)
95
96 # Labels are either '<=50K' or '>50K', converting to 1 and 0.
97 y_train = y_train_pre.str.contains('>').astype(int)
98 y_test = y_test_pre.str.contains('>').astype(int)
99
100 DIMS = X_train.shape[1]
101
102 #endregion
103
104 #region
105 # Prepare grid search over all combinations of hyperparameters
106 num_param_states = len(LAYER_DEPTHS) * len(HIDDEN_LAYERS) * len(BATCH_SIZES) *
107     len(OPTIMIZERS)
108 combinations = list(itertools.product(LAYER_DEPTHS, HIDDEN_LAYERS, BATCH_SIZES,
109     OPTIMIZERS))
110 # Append indices to this large list
111 combinations = [(x + (i,)) for i, x in enumerate(combinations)]
112
113 # Prepare gigantic list of complete parameters.
114 args_full = [(combo, DIMS, ACTIVATION_FUNCTIONS, LOSS_FUNCTION, X_train, y_train, EPOCHS,
115     kf, n_folds) for combo in combinations]
116
117 print(f"ready to iterate over {num_param_states} combinations")
118
119 # Start running parallel trainings over all combinations
120 outputData = []
121 with multiprocessing.Pool() as pool:
122     results = pool.starmap(eval_model, args_full)
123
124 # Open the CSV file once and write all results
125 with open(filename, 'w', newline='') as csvfile:
126     writer = csv.writer(csvfile, delimiter=',')
127     header = [
128         ["Index"], ["Depth"], ["Layers"], ["Batch Size"], ["Optimizer"], ["Valid
129         Loss"], ["Valid Acc"], ["Time"]]
130     ]
131     writer.writerow(header)
132     for result in results:
133         outputData.append(result)

```

```

127         writer.writerow(result)
128
129     #endregion
130
131     # Pick top 1 in terms of validation loss and get test error
132     sorted_outputData = sorted(outputData, key=lambda x: x[5])
133     top_1 = sorted_outputData[0]
134     index, depth, hLayers, batchSize, optimizer, _, _, _ = top_1
135     result = test_model((depth, hLayers, batchSize, optimizer, index), DIMS,
136                        ACTIVATION_FUNCTIONS, LOSS_FUNCTION, X_train, y_train, X_test, y_test, EPOCHS)
137     print("Index, Depth, Layers, Batch Size, Optimizer, Test Loss, Test Accuracy, Training
138           Time")
139     print(result)
140
141 # Function to train and evaluate TF model
142 def eval_model(dynamic_args, d, activation, loss, X_train, y_train, epochs, kf, n_folds):
143     # Unpack arguments
144     depth, hLayers, batchSize, optimizer, index = dynamic_args
145     # Clear previous models and graphs
146     tf.keras.backend.clear_session()
147
148     # Create a new graph for the current process
149     with tf.Graph().as_default():
150         # Create a session for the current graph
151         with tf.compat.v1.Session() as sess:
152             # Remind Tensorflow to keep quiet
153             tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.FATAL)
154
155             # Model creation
156             if hLayers == 1:
157                 model = tf.keras.models.Sequential([
158                     tf.keras.layers.Dense(depth, activation=activation, input_shape=(d,)),
159                     tf.keras.layers.Dense(1, activation='sigmoid')
160                 ])
161             else:
162                 model = tf.keras.models.Sequential([
163                     tf.keras.layers.Dense(depth, activation=activation, input_shape=(d,)),
164                     tf.keras.layers.Dense(depth, activation=activation),
165                     tf.keras.layers.Dense(1, activation='sigmoid')
166                 ])
167
168             # Compile the model
169             model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
170
171             # Start stopwatch
172             start_time = time.time()
173
174             # Initialize averaged results, test results are to be kept aside.
175             valid_accuracy = 0
176             valid_loss = 0
177             # Train the model
178             for index_train, index_val in kf.split(X_train):
179                 X_vtrain, X_val = X_train[index_train], X_train[index_val]
180                 y_vtrain, y_val = y_train[index_train], y_train[index_val]
181                 history = model.fit(X_vtrain, y_vtrain, epochs=epochs, batch_size=batchSize,
182                                   validation_split=0.2)
183                 # Test this model on the current fold's validation dataset, and the test set
184                 this_valid_loss, this_valid_accuracy = model.evaluate(X_val, y_val)
185
186                 # Add to average
187                 valid_accuracy += this_valid_accuracy / n_folds
188                 valid_loss += this_valid_loss / n_folds
189
190             # Stop stopwatch
191             end_time = time.time()

```

```

191         training_time = end_time - start_time
192         print(f"Done with Run {index}, t = {training_time}")
193         return index, depth, hLayers, batchSize, optimizer, valid_loss, valid_accuracy,
            training_time
194
195 def test_model(dynamic_args, d, activation, loss, X_train, y_train, X_test, y_test, epochs):
196     # Unpack arguments
197     depth, hLayers, batchSize, optimizer, index = dynamic_args
198     # Clear previous models and graphs
199     tf.keras.backend.clear_session()
200
201     # Create a new graph for the current process
202     with tf.Graph().as_default():
203         # Create a session for the current graph
204         with tf.compat.v1.Session() as sess:
205             # Remind Tensorflow to keep quiet
206             tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.FATAL)
207
208             # Model creation
209             if hLayers == 1:
210                 model = tf.keras.models.Sequential([
211                     tf.keras.layers.Dense(depth, activation=activation, input_shape=(d,)),
212                     tf.keras.layers.Dense(1, activation='sigmoid')
213                 ])
214             else:
215                 model = tf.keras.models.Sequential([
216                     tf.keras.layers.Dense(depth, activation=activation, input_shape=(d,)),
217                     tf.keras.layers.Dense(depth, activation=activation),
218                     tf.keras.layers.Dense(1, activation='sigmoid')
219                 ])
220
221             # Compile the model
222             model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
223
224             # Start stopwatch
225             start_time = time.time()
226
227             # Train the model
228             history = model.fit(X_train, y_train, epochs=epochs, batch_size = batchSize,
                validation_split=0.2)
229             test_loss, test_accuracy = model.evaluate(X_test, y_test)
230             # Stop stopwatch
231             end_time = time.time()
232
233             training_time = end_time - start_time
234             print(f"Done with test, t = {training_time}")
235             return index, depth, hLayers, batchSize, optimizer, test_loss, test_accuracy,
                training_time
236
237 if __name__ == "__main__":
238     main()

```

```

1  ## Code Execution Steps:
2  # - If you haven't already, run "pip3 install numpy matplotlib seaborn multiprocessing
    tensorflow scikit-learn ucimlrepo itertools pandas csv"
3  # - Modify individual hyperparameters to evaluate them on the test set.
4  # - Run python file using any Python >3.9 interpreter with the above libraries installed.
5  import os
6  import tensorflow as tf
7  os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3' # Suppresses noisy outputs
8  tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.FATAL)
9  from sklearn.preprocessing import StandardScaler, OneHotEncoder
10 from sklearn.compose import ColumnTransformer
11 from sklearn.pipeline import Pipeline
12 from sklearn.metrics import confusion_matrix, classification_report
13 from ucimlrepo import fetch_ucirepo

```

```

14 import itertools
15 import pandas as pd
16 import csv
17 import time
18
19 def main():
20     ## Hyperparameters
21     PROPORTION_TRAIN = 0.8
22     layer_depth = 640
23     hidden_layers = 2
24     epochs = 100
25     batch_size = 128
26     activation_function = "relu"
27     loss_function = "binary_crossentropy"
28     optimizer = "adam"
29
30     # filename = "FNN Results UCI.csv"
31
32     ## Data Pre-Processing
33     #region
34     adult = fetch_ucirepo(id=2)
35     # data (as pandas dataframes)
36     X = adult.data.features
37     y = adult.data.targets
38
39     # metadata
40     # print(adult.metadata)
41     # variable information
42     # print(adult.variables)
43
44     # Get rid of NaNs
45     combined = pd.concat([X, y], axis=1)
46     combined = combined.dropna()
47     X = combined.iloc[:, :-1]
48     y = combined.iloc[:, -1]
49
50     NUM = X.shape[0]
51
52     # Shuffle Data
53     combined_shuffled = combined.sample(frac=1).reset_index(drop=True)
54     X_shuffled = combined_shuffled.iloc[:, :-1]
55     y_shuffled = combined_shuffled.iloc[:, -1]
56
57     # Split data by train and test.
58     splitIndex = int(PROPORTION_TRAIN * NUM)
59     X_train_pre = X_shuffled.iloc[:splitIndex, :] # Shape 38096x14 (108 for onehot)
60     y_train_pre = y_shuffled.iloc[:splitIndex] # Shape 38096
61     X_test_pre = X_shuffled.iloc[splitIndex:, :] # Shape 9525x14 (108 for onehot)
62     y_test_pre = y_shuffled.iloc[splitIndex:] # Shape 9525
63
64     # Convert categorical features to numerical
65     categorical_cols = X.select_dtypes(include=['object', 'category']).columns
66     numerical_cols = X.select_dtypes(include=['number']).columns
67     # Preprocessing for numerical data
68     numerical_transformer = StandardScaler()
69
70     # Preprocessing for categorical data
71     categorical_transformer = OneHotEncoder(handle_unknown='ignore')
72
73     # Bundle preprocessing for numerical and categorical data
74     preprocessor = ColumnTransformer(
75         transformers=[
76             ('num', numerical_transformer, numerical_cols),
77             ('cat', categorical_transformer, categorical_cols)
78         ])
79
80     # Create a preprocessing and modeling pipeline

```



```

81     clf = Pipeline(steps=[('preprocessor', preprocessor)])
82
83     # Preprocess the features. Xs are in compressed sparse matrix format (sample, feature),
84     # value
85     X_train = clf.fit_transform(X_train_pre)
86     X_test = clf.transform(X_test_pre)
87
88     # Labels are either '<=50K' or '>50K', converting to 1 and 0.
89     y_train = y_train_pre.str.contains('>').astype(int)
90     y_test = y_test_pre.str.contains('>').astype(int)
91
92     DIMS = X_train.shape[1]
93
94     #endregion
95
96     result = test_model((layer_depth, hidden_layers, batch_size, optimizer), DIMS,
97                         activation_function, loss_function, X_train, y_train, X_test, y_test, epochs)
98     depth, hLayers, batchSize, optimizer, test_loss, test_accuracy, training_time, conmat,
99     y_pred = result
100
101     print("Depth, Layers, Batch Size, Optimizer, Test Loss, Test Accuracy, Training Time")
102     print(f"{depth}, {hLayers}, {batchSize}, {optimizer}, {test_loss}, {test_accuracy},
103           {training_time}")
104
105     # Print confusion matrix
106     print("Confusion Matrix:")
107     print(conmat)
108
109     # Compute and print classification report
110     target_names = ['Class 0', 'Class 1']
111     report = classification_report(y_test, (y_pred > 0.5), target_names=target_names)
112     print("Classification Report:")
113     print(report)
114
115 def test_model(dynamic_args, d, activation, loss, X_train, y_train, X_test, y_test, epochs):
116     # Unpack arguments
117     depth, hLayers, batchSize, optimizer = dynamic_args
118     # Clear previous models and graphs
119     tf.keras.backend.clear_session()
120
121     # Create a new graph for the current process
122     with tf.Graph().as_default():
123         # Create a session for the current graph
124         with tf.compat.v1.Session() as sess:
125             # Remind Tensorflow to keep quiet
126             tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.FATAL)
127
128             # Model creation
129             if hLayers == 1:
130                 model = tf.keras.models.Sequential([
131                     tf.keras.layers.Dense(depth, activation=activation, input_shape=(d,)),
132                     tf.keras.layers.Dense(1, activation='sigmoid')
133                 ])
134             else:
135                 model = tf.keras.models.Sequential([
136                     tf.keras.layers.Dense(depth, activation=activation, input_shape=(d,)),
137                     tf.keras.layers.Dense(depth, activation=activation),
138                     tf.keras.layers.Dense(1, activation='sigmoid')
139                 ])
140
141             # Compile the model
142             model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
143
144             # Start stopwatch
145             start_time = time.time()

```

```
144     # Train the model
145     history = model.fit(X_train, y_train, epochs=epochs, batch_size = batchSize,
146                         validation_split=0.2)
147     test_loss, test_accuracy = model.evaluate(X_test, y_test)
148
149     # Predict the test set
150     y_pred = model.predict(X_test)
151
152     # Compute confusion matrix
153     conmat = confusion_matrix(y_test, (y_pred > 0.5))
154
155     # Stop stopwatch
156     end_time = time.time()
157
158     training_time = end_time - start_time
159     print(f"Done with test, t = {training_time}")
160
161     return depth, hLayers, batchSize, optimizer, test_loss, test_accuracy,
162           training_time, conmat, y_pred
163
164 if __name__ == "__main__":
165     main()
```

U. Code used for Dataset 3, Logistic Regression

```
1  ## Code Execution Steps:
2  # - This code can be run independently from the code from the other datasets.
3
4  from sklearn.linear_model import LogisticRegression
5  from sklearn.model_selection import GridSearchCV, cross_val_score
6  import matplotlib.pyplot as plt
7  from sklearn.metrics import classification_report, confusion_matrix
8  import seaborn as sns
9  def logreg_fashion_pipeline(X_train, X_test, y_train, y_test, dataset='Fashion'):
10     # First Grid Search: L2 and L1
11     param_grid_l2_l1 = {
12         'C': [0.01, 0.1, 1],
13         'penalty': ['l2', 'l1']
14     }
15
16     # Create Logistic Regression model
17     model = LogisticRegression(solver='saga', max_iter=100)
18
19     # Perform Grid Search for L2 and L1
20     grid_search_l2_l1 = GridSearchCV(model, param_grid_l2_l1, cv=5, scoring='accuracy')
21     grid_search_l2_l1.fit(X_train, y_train)
22     print('finished L2, L1 grid search CV')
23
24     # Second Grid Search: ElasticNet and l1_ratio
25     param_grid_elasticnet = {
26         'C': [0.01, 0.1, 1],
27         'penalty': ['elasticnet'],
28         'l1_ratio': [0.2, 0.4, 0.6, 0.8]
29     }
30
31     model = LogisticRegression(solver='saga', max_iter=100)
32
33     # Perform Grid Search for ElasticNet and l1_ratio
34     grid_search_elasticnet = GridSearchCV(model, param_grid_elasticnet, cv=5,
35         scoring='accuracy')
36     grid_search_elasticnet.fit(X_train, y_train)
37     print('finished elastic-net grid search CV')
38
39     # Extract the results
40     results_l2_l1 = grid_search_l2_l1.cv_results_
41     l2_means = results_l2_l1['mean_test_score'][results_l2_l1['param_penalty'] == 'l2']
42     l1_means = results_l2_l1['mean_test_score'][results_l2_l1['param_penalty'] == 'l1']
43
44     results_elasticnet = grid_search_elasticnet.cv_results_
45     best_ratio = grid_search_elasticnet.best_params_['l1_ratio']
46     elastic_means = results_elasticnet['mean_test_score'][results_elasticnet['param_l1_ratio']
47         == best_ratio]
48
49     # Logistic Regression with no regularization
50     model_none = LogisticRegression(max_iter=100, penalty=None)
51     none_accuracies = cross_val_score(model_none, X_train, y_train, cv=5, scoring='accuracy')
52     print('finished None CV')
53     none_accuracy = np.mean(none_accuracies)
54
55     # Plot the curves
56     plt.figure(figsize=(12, 8))
57
58     # L2 Average Accuracy Curve
59     plt.plot(param_grid_l2_l1['C'], l2_means, label='L2', marker='o')
60
61     # L1 Average Accuracy Curve
62     plt.plot(param_grid_l2_l1['C'], l1_means, label='L1', marker='o')
```

```

63     plt.plot(param_grid_elasticnet['C'], elastic_means, label=f'Elastic-Net (best l1_ratio:
        {best_ratio:.2f})', linestyle='--', color='red', marker='o')
64
65     # Logistic Regression with no regularization (constant line)
66     plt.axhline(y=none_accuracy, color='green', linestyle='--', label='None', linewidth=2)
67
68     # Set plot properties
69     plt.title(f'{dataset} - Logistic Regression Grid Search')
70     plt.xlabel('C', fontsize=14)
71     plt.ylabel('Average Validation Accuracy', fontsize=14)
72     plt.xscale('log')
73     plt.legend(fontsize=14)
74     plt.grid(True)
75     plt.show()
76
77     # Extract the best parameters and scores for each penalty
78     best_params_l2_l1 = grid_search_l2_l1.best_params_
79     best_score_l2_l1 = grid_search_l2_l1.best_score_
80
81     best_params_elasticnet = grid_search_elasticnet.best_params_
82     best_score_elasticnet = grid_search_elasticnet.best_score_
83
84     # Choose the best parameters based on the highest mean test score
85     best_params = max([(best_params_l2_l1, best_score_l2_l1),
86                       (best_params_elasticnet, best_score_elasticnet),
87                       ({'penalty':None}, none_accuracy)],
88                       key=lambda x: x[1])[0]
89
90     print("Best Parameters L2/L1:", best_params_l2_l1)
91     print("Best Score L2/L1:", best_score_l2_l1)
92     print("Best Parameters ElasticNet:", best_params_elasticnet)
93     print("Best Score ElasticNet:", best_score_elasticnet)
94     print("Score None:", none_accuracy)
95     print("Best Parameters Overall:", best_params)
96
97     # Fit the best model with the full training set
98     best_model = LogisticRegression(solver='saga', max_iter=100, **best_params)
99     best_model.fit(X_train, y_train)
100    print('finished best model fit')
101
102    # Evaluate the best model on the test set
103    y_pred = best_model.predict(X_test)
104
105    # Display the classification report and confusion matrix
106    print("Test Classification report:\n", classification_report(y_test, y_pred))
107    print("Test Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
108
109    # Plot the confusion matrix
110    confusion_mat = confusion_matrix(y_test, y_pred)
111    sns.heatmap(confusion_mat, annot=True, fmt='d', cmap='Blues')
112    plt.title(f'{dataset} - Logistic Regression Test Confusion Matrix')
113    plt.xlabel('Predicted Label')
114    plt.ylabel('True Label')
115    plt.show()
116
117    return grid_search_l2_l1, grid_search_elasticnet
118
119    ## import Fashion MNIST data
120    import numpy as np
121    import torch, torchvision
122    train_set = torchvision.datasets.FashionMNIST("./data", download=True)
123    test_set = torchvision.datasets.FashionMNIST("./data", download=True, train=False)
124    X_train = train_set.data.numpy()
125    labels_train = train_set.targets.numpy()
126    X_test = test_set.data.numpy()
127    labels_test = test_set.targets.numpy()
128    X_train = X_train.reshape((X_train.shape[0], X_train.shape[1]*X_train.shape[2]))

```

```
129 X_test = X_test.reshape((X_test.shape[0], X_test.shape[1]*X_test.shape[2]))
130 X_train = X_train/255.0
131 X_test = X_test/255.0
```

V. Code used for Dataset 3, Support Vector Machines

```
1  ## Code Execution Steps:
2  # - Run the python file. Ex: python ./fashionMNIST\_svm.py
3  import numpy as np
4  import pandas as pd
5  import matplotlib.pyplot as plt
6  import torch, torchvision
7  from sklearn.model_selection import train_test_split
8  from sklearn.svm import SVC
9  from sklearn.metrics import classification_report
10 from sklearn.model_selection import GridSearchCV
11 from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score,
    fl_score
12 from sklearn.model_selection import ValidationCurveDisplay
13 from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
14
15
16 # Load the Fashion MNIST Data
17 trainData = torchvision.datasets.FashionMNIST("./data", download=True)
18 testData = torchvision.datasets.FashionMNIST("./data", download=True, train=False)
19
20 # reshape the images from (28x28) to (14x14)
21 resize = torchvision.transforms.Resize((14, 14))
22 trainDataResize = resize(trainData.data)
23 testDataResize = resize(testData.data)
24
25 xTrainData = trainDataResize.data.numpy()
26 yTrainData = trainData.targets.numpy()
27 xTestData = testDataResize.data.numpy()
28 yTestData = testData.targets.numpy()
29
30 # Normalize the data
31 xTrainData = xTrainData.reshape((xTrainData.shape[0], xTrainData.shape[1]*xTrainData.shape[2]))
32 xTestData = xTestData.reshape((xTestData.shape[0], xTestData.shape[1]*xTestData.shape[2]))
33 xTrainData = xTrainData/255.0
34 xTestData = xTestData/255.0
35
36 # baseline/default SVM model
37 model = SVC(max_iter=10**5)
38
39 model.fit(xTrainData, yTrainData)
40
41 yPredict = model.predict(xTestData)
42
43 # Score report for the baseline/default SVM model
44 print(classification_report(yTestData,yPredict))
45
46 #Scoring Metrics
47 scoring = {'accuracy' : make_scorer(accuracy_score),
48           'precision' : make_scorer(precision_score, average='macro', zero_division=0),
49           'recall' : make_scorer(recall_score, average='macro', zero_division=0),
50           'fl_score' : make_scorer(fl_score, average='macro', zero_division=0)
51 }
52
53 # hyperparameter tuning and KFold cross validation
54 gammaValues = np.array([10**k for k in np.linspace(-4, 2, 7)])
55 gammaValues = gammaValues.tolist()
56 # Appending 'scale' and 'auto' to the list
57 gammaValues.append('scale')
58 gammaValues.append('auto')
59
60 parametersGridCV = [
61     {
62         'kernel' : ['linear'],
63         'C' : [10**k for k in np.linspace(-4, 1, 6)]
64     },
```

```

65     {
66         'kernel' : ['poly'],
67         'C' : [10**k for k in np.linspace(-3, 3, 7)],
68         'degree' : list(range(1, 6, 1)),
69         'gamma' : gammaValues
70     },
71     {
72         'kernel' : ['rbf'],
73         'C' : [10**k for k in np.linspace(-3, 3, 7)],
74         'gamma' : gammaValues
75     }
76 ]
77
78 gridCV = GridSearchCV(estimator=model, param_grid=parametersGridCV, cv=5, scoring=scoring,
79                       refit='accuracy', verbose=3, n_jobs=-1)
80
81 # fitting the model
82 gridCV.fit(xTrainData, yTrainData)
83 yPredict = gridCV.predict(xTestData)
84
85 print(f'Best Parameters : {gridCV.best_params_}')
86
87 # Score report with best parameters after hyperparameter tuning for SVM model
88 print(classification_report(yTestData, yPredict))
89
90 # refine the C and gamma parameter
91 C = gridCV.best_params_['C']
92 gamma = gridCV.best_params_['gamma']
93 refineParametersGridCV = {
94     'kernel' : [gridCV.best_params_['kernel']],
95     'C' : [10**k for k in np.linspace(np.log10(C) - 0.75, np.log10(C) + 1, 10)],
96     'gamma' : [10**k for k in np.linspace(np.log10(gamma), np.log10(gamma) + 1, 10)]
97 }
98
99 refineGridCV = GridSearchCV(estimator=model, param_grid=refineParametersGridCV, cv=5,
100                             scoring=scoring, refit='accuracy', verbose=3, n_jobs=-1)
101
102 # fitting the model
103 refineGridCV.fit(xTrainData, yTrainData)
104
105 # Display the best parameters
106 print(f'Best Parameters : {refineGridCV.best_params_}')
107
108 # predict on the test data
109 yPredict = refineGridCV.predict(xTestData)
110
111 # Score report with best parameters after hyperparameter tuning for SVM model
112 print(classification_report(yTestData, yPredict))
113
114 # Confusion Marix
115 confusionMatrix = confusion_matrix(yTestData, yPredict)
116 dispConfusionMatrix = ConfusionMatrixDisplay(confusion_matrix=confusionMatrix)
117 dispConfusionMatrix.plot(cmap=plt.cm.Blues)
118 plt.title("Confusion Matrix for SVM Classifier on FashionMNIST Dataset")
119
120 # Display validation curves for training and test
121 gammaValue = 0.16681005372000587
122 validationCurve = ValidationCurveDisplay.from_estimator(
123     SVC(kernel='rbf'),
124     xTrainData,
125     yTrainData,
126     param_name="gamma",
127     param_range=np.logspace(-4, 3, 8),
128     score_type="both",
129     n_jobs=-1,
130     score_name="Accuracy",
131 )

```

```

130 validationCurve.ax_.set_title("Validation Curve for SVM with an RBF kernel")
131 validationCurve.ax_.set_xlabel(r"gamma")
132 validationCurve.ax_.set_ylim(0.0, 1.1)
133 validationCurve.ax_.axvline(x=gammaValue, color='g', linestyle='--')
134 validationCurve.ax_.text(gammaValue, 1.05, 'gamma = {:.4f}'.format(gammaValue), color='g')
135 plt.show()
136
137 cValue = 4.354004653656649
138 validationCurve = ValidationCurveDisplay.from_estimator(
139     SVC(kernel='rbf', gamma=0.16681005372000587),
140     xTrainData,
141     yTrainData,
142     param_name="C",
143     param_range=np.logspace(-3, 3, 7),
144     score_type="both",
145     n_jobs=-1,
146     score_name="Accuracy",
147 )
148 validationCurve.ax_.set_title("Validation Curve for SVM with an RBF kernel")
149 validationCurve.ax_.set_xlabel(r"C")
150 validationCurve.ax_.set_ylim(0.0, 1.1)
151 validationCurve.ax_.axvline(x=cValue, color='g', linestyle='--')
152 validationCurve.ax_.text(cValue, 1.05, 'C = {:.4f}'.format(cValue), color='g')
153 plt.show()

```


W. Code used for Dataset 3, k-Nearest Neighbors

```
1 ## Code Execution Steps:
2 # - If you already changed %pip to an appropriate syntax in previous section, then just click
  RUN.
3 # - Run the python file "PCA+k\NN.py". Each section of code will provide analysis solution
  for dataset 1, 2 and 3. Here is dataset 3.
4 %pip install ucimlrepo
5 import numpy as np
6 import matplotlib.pyplot as plt
7 import torch, torchvision
8 import pandas as pd
9 import seaborn as sns
10 import time
11 from sklearn.decomposition import PCA
12 from sklearn.impute import SimpleImputer
13 from sklearn.model_selection import train_test_split, GridSearchCV
14 from sklearn.neighbors import KNeighborsClassifier
15 from sklearn.pipeline import Pipeline
16 from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
17 from sklearn.metrics import classification_report, confusion_matrix
18 from sklearn.compose import ColumnTransformer
19 from ucimlrepo import fetch_ucirepo
20
21 # Start the timer
22 start_time = time.time()
23
24 # Load Fashion MNIST dataset
25 train_set = torchvision.datasets.FashionMNIST("./data", download = True)
26 test_set = torchvision.datasets.FashionMNIST("./data", download = True, train = False)
27
28 # Prepare training and testing data
29 X_train = train_set.data.numpy().reshape((train_set.data.size(0), -1)) / 255.0
30 labels_train = train_set.targets.numpy()
31 X_test = test_set.data.numpy().reshape((test_set.data.size(0), -1)) / 255.0
32 labels_test = test_set.targets.numpy()
33
34 # Create the PCA + k-NN pipeline
35 pipeline = Pipeline([
36     ('scaler', StandardScaler()),
37     ('pca', PCA(n_components=0.95)), # Retain 95% of the variance
38     ('knn', KNeighborsClassifier(n_neighbors=5)) # Start with k=5
39 ])
40
41 # Fit the pipeline to the training data
42 pipeline.fit(X_train, labels_train)
43
44 # Predict the labels for the test set
45 labels_pred = pipeline.predict(X_test)
46
47 # Classification report
48 print(classification_report(labels_test, labels_pred))
49
50 # Confusion matrix
51 conf_mat = confusion_matrix(labels_test, labels_pred)
52
53 # Plot the confusion matrix
54 print(f"Training time: {training_time:.2f} seconds")
55 plt.figure(figsize=(10, 8))
56 sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues')
57 plt.xlabel('Predicted Label')
58 plt.ylabel('True Label')
59 plt.title('Confusion Matrix for k-NN Classifier')
60 plt.show()
61
62 # Extract the PCA from the pipeline
63 pca = pipeline.named_steps['pca']
```

```
64 end_time = time.time()
65 training_time = end_time - start_time
66
67 # Plot the cumulative explained variance to decide on the number of components to retain
68 plt.figure(figsize=(10, 8))
69 plt.plot(np.cumsum(pca.explained_variance_ratio_))
70 plt.xlabel('Number of components')
71 plt.ylabel('Cumulative explained variance')
72 plt.title('Explained Variance by PCA Components')
73 plt.axhline(y=0.95, color='r', linestyle='--', label='95% Explained Variance')
74 plt.legend(loc='best')
75 plt.show()
```

X. Code used for Dataset 3, Neural Networks

```
1  ## Code Execution Steps:
2  # - Run "pip3 install numpy matplotlib seaborn torch torchvision multiprocessing tensorflow
   scikit-learn ucimlrepo itertools pandas csv"
3  # - Modify Hyperparameter lists to explore different grid search arrangements.
4  # - Run python file using any Python >3.9 interpreter with the above libraries installed.
5  import numpy as np
6  import numpy.random as random
7  import matplotlib.pyplot as plt
8  import seaborn as sns
9  import torch, torchvision
10 import multiprocessing
11 import os
12 import tensorflow as tf
13 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3' # Suppresses noisy outputs
14 tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.FATAL)
15 from sklearn.preprocessing import StandardScaler
16 from sklearn.model_selection import KFold
17 import itertools
18 import pandas as pd
19 import csv
20 import time
21
22 def main():
23     ## Hyperparameters
24     FILTERS = [16, 32, 64]
25     LAYER_DEPTHS = [10, 20, 40, 80]
26     HIDDEN_LAYERS = [1, 2]
27     EPOCHS = 30
28     BATCH_SIZES = [32, 64]
29     ACTIVATION_FUNCTIONS = "relu"
30     LOSS_FUNCTION = "sparse_categorical_crossentropy"
31     OPTIMIZERS = ["SGD", "rmsprop", "adam", "adadelta", "ftrl"]
32
33     filename = "CNN Results.csv"
34     n_folds = 5
35     kf = KFold(n_splits=n_folds)
36
37     ## Data Pre-Processing
38     #region
39     train_set = torchvision.datasets.FashionMNIST("./data", download=True)
40     test_set = torchvision.datasets.FashionMNIST("./data", download=True, train=False)
41     X_train = train_set.data.numpy()
42     labels_train = train_set.targets.numpy()
43     X_test = test_set.data.numpy()
44     labels_test = test_set.targets.numpy()
45     # Add a fourth dimension to satisfy conv2D
46     X_train = X_train.reshape(-1, 28, 28, 1)
47     X_test = X_test.reshape(-1, 28, 28, 1)
48     X_train = X_train/255.0
49     X_test = X_test/255.0
50     N_TRAIN = X_train.shape[0]
51     N_TEST = X_test.shape[0]
52     DIMS = [X_train.shape[1], X_train.shape[2]]
53     # Shuffle data
54     order_train = random.permutation(N_TRAIN)
55     order_test = random.permutation(N_TEST)
56     X_TRAIN = X_train[order_train,:] # Shape 60000x28x28
57     Y_TRAIN = labels_train[order_train] # Shape 60000x1
58     X_TEST = X_test[order_test,:] # Shape Nx28x28
59     Y_TEST = labels_test[order_test] # Shape Nx1
60
61     #endregion
62
63     # Prepare grid search over all combinations of hyperparameters
```

```

64 num_param_states = len(FILTERS) * len(LAYER_DEPTHS) * len(HIDDEN_LAYERS) *
    len(BATCH_SIZES) * len(OPTIMIZERS)
65 combinations = list(itertools.product(FILTERS, LAYER_DEPTHS, HIDDEN_LAYERS, BATCH_SIZES,
    OPTIMIZERS))
66 # Append indices to this large list
67 combinations = [(x + (i,)) for i, x in enumerate(combinations)]
68
69 # Prepare gigantic list of complete parameters.
70 args_full = [(combo, DIMS, ACTIVATION_FUNCTIONS, LOSS_FUNCTION, X_TRAIN, Y_TRAIN, EPOCHS,
    kf, n_folds) for combo in combinations]
71
72 print(f"ready to iterate over {num_param_states} combinations")
73
74 # Start running parallel trainings over all combinations
75 outputData = []
76 with multiprocessing.Pool() as pool:
77     results = pool.starmap(eval_model, args_full)
78
79 # Open the CSV file once and write all results
80 with open(filename, 'w', newline='') as csvfile:
81     writer = csv.writer(csvfile, delimiter=',')
82     header = [
83         ["Index"], ["Filters"], ["Depth"], ["Layers"], ["Batch Size"],
84         ["Optimizer"], ["Valid Loss"], ["Valid Acc"], ["Time"]]
85     ]
86     writer.writerow(header)
87     for result in results:
88         outputData.append(result)
89         writer.writerow(result)
90
91 # Pick top 1 in terms of validation loss and get test error
92 sorted_outputData = sorted(outputData, key=lambda x: x[5])
93 top_1 = sorted_outputData[0]
94 filters, index, depth, hLayers, batchSize, optimizer, _, _, _ = top_1
95 result = test_model((filters, depth, hLayers, batchSize, optimizer, index), DIMS,
96     ACTIVATION_FUNCTIONS, LOSS_FUNCTION, X_train, y_train, X_test, y_test, EPOCHS)
97 print("Filters, Index, Depth, Layers, Batch Size, Optimizer, Test Loss, Test Accuracy,
98     Training Time")
99 print(result)
100
101 # Function to train and evaluate TF model
102 def eval_model(dynamic_args, d, activation, loss, X_train, y_train, epochs, kf, n_folds):
103     # Unpack arguments
104     filters, depth, hLayers, batchSize, optimizer, index = dynamic_args
105     # Clear previous models and graphs
106     tf.keras.backend.clear_session()
107     # Create a new graph for the current process
108     with tf.Graph().as_default():
109         # Create a session for the current graph
110         with tf.compat.v1.Session() as sess:
111             # Remind Tensorflow to keep quiet
112             tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.FATAL)
113
114             # Model creation
115             if hLayers == 1:
116                 model = tf.keras.models.Sequential([
117                     # Convolutional layer with N filters, kernel size of 3x3, ReLU activation,
118                     # and input shape for MNIST
119                     tf.keras.layers.Conv2D(filters, kernel_size=(3, 3), activation=activation,
120                         input_shape=(d[0], d[1], 1)),
121                     # Pooling layer to reduce dimensionality
122                     tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
123                     # Flatten the output to feed into the dense layer
124                     tf.keras.layers.Flatten(),
125                     tf.keras.layers.Dense(depth, activation=activation),
126                     tf.keras.layers.Dense(10, activation='softmax')
127                 ])
128             else:
129                 model = tf.keras.models.Sequential([

```

```

123         # Convolutional layer with N filters, kernel size of 3x3, ReLU activation,
124         and input shape for MNIST
125         tf.keras.layers.Conv2D(filters, kernel_size=(3, 3), activation=activation,
126         input_shape=(d[0], d[1],1)),
127         # Pooling layer to reduce dimensionality
128         tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
129         # Flatten the output to feed into the dense layer
130         tf.keras.layers.Flatten(),
131         tf.keras.layers.Dense(depth, activation=activation),
132         tf.keras.layers.Dense(depth, activation=activation),
133         tf.keras.layers.Dense(10, activation='softmax')
134     ])
135
136     # Compile the model
137     model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
138
139     # Start stopwatch
140     start_time = time.time()
141
142     # Initialize averaged results, test results are to be kept aside.
143     valid_accuracy = 0
144     valid_loss = 0
145     # Train the model
146     for index_train, index_val in kf.split(X_train):
147         X_vtrain, X_val = X_train[index_train], X_train[index_val]
148         y_vtrain, y_val = y_train[index_train], y_train[index_val]
149         history = model.fit(X_vtrain, y_vtrain, epochs=epochs, batch_size=batchSize,
150         validation_split=0.2)
151         # Test this model on the current fold's validation dataset, and the test set
152         this_valid_loss, this_valid_accuracy = model.evaluate(X_val, y_val)
153
154         # Add to average
155         valid_accuracy += this_valid_accuracy / n_folds
156         valid_loss += this_valid_loss / n_folds
157
158     # Stop stopwatch
159     end_time = time.time()
160
161     training_time = end_time - start_time
162     print(f"Done with Run {index}, t = {training_time}")
163     return index, filters, depth, hLayers, batchSize, optimizer, valid_loss,
164     valid_accuracy, training_time
165
166 if __name__ == "__main__":
167     main()

```

```

1  ## Code Execution Steps:
2  # - If you haven't already, run "pip3 install numpy matplotlib seaborn torch torchvision
3  multiprocessing tensorflow scikit-learn ucimlrepo itertools pandas csv"
4  # - Modify individual hyperparameters to evaluate them on the test set.
5  # - Run python file using any Python >3.9 interpreter with the above libraries installed.
6  import numpy as np
7  import numpy.random as random
8  import matplotlib.pyplot as plt
9  import seaborn as sns
10 import torch, torchvision
11 import os
12 import tensorflow as tf
13 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3' # Suppresses noisy outputs
14 tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.FATAL)
15 from sklearn.metrics import confusion_matrix, classification_report
16 import pandas as pd
17 import csv
18 import time

```

```

19 def main():
20     ## Hyperparameters
21     filters = 64
22     layer_depth = 80
23     hidden_layers = 1
24     epochs = 30
25     batch_size = 64
26     activation_function = "relu"
27     loss_function = "sparse_categorical_crossentropy"
28     optimizer = "adam"
29
30     filename = "CNN Final Results.csv"
31
32     ## Data Pre-Processing
33     #region
34     train_set = torchvision.datasets.FashionMNIST("./data", download=True)
35     test_set = torchvision.datasets.FashionMNIST("./data", download=True, train=False)
36     X_train = train_set.data.numpy()
37     labels_train = train_set.targets.numpy()
38     X_test = test_set.data.numpy()
39     labels_test = test_set.targets.numpy()
40     # Add a fourth dimension to satisfy conv2D
41     X_train = X_train.reshape(-1, 28, 28, 1)
42     X_test = X_test.reshape(-1, 28, 28, 1)
43     X_train = X_train/255.0
44     X_test = X_test/255.0
45     N_TRAIN = X_train.shape[0]
46     N_TEST = X_test.shape[0]
47     DIMS = [X_train.shape[1], X_train.shape[2]]
48     # Shuffle data
49     order_train = random.permutation(N_TRAIN)
50     order_test = random.permutation(N_TEST)
51     X_train = X_train[order_train,:] # Shape 60000x28x28
52     y_train = labels_train[order_train] # Shape 60000x1
53     X_test = X_test[order_test,:] # Shape Nx28x28
54     y_test = labels_test[order_test] # Shape Nx1
55
56     #endregion
57
58     result = test_model((filters, layer_depth, hidden_layers, batch_size, optimizer), DIMS,
59         activation_function, loss_function, X_train, y_train, X_test, y_test, epochs)
60
61     filters, depth, hLayers, batchSize, optimizer, test_loss, test_accuracy, training_time,
62     conmat, y_pred = result
63
64     print("Filters, Depth, Layers, Batch Size, Optimizer, Test Loss, Test Accuracy, Training
65         Time")
66     print(f"{filters}, {depth}, {hLayers}, {batchSize}, {optimizer}, {test_loss},
67         {test_accuracy}, {training_time}")
68
69     # Print confusion matrix
70     # print("Confusion Matrix:")
71     # print(conmat)
72
73     plt.figure(figsize=(10,8))
74     sns.heatmap(conmat, annot=True, fmt='d', cmap='Blues')
75     plt.xlabel('Predicted Label')
76     plt.ylabel('True Label')
77     plt.title('Confusion Matrix for CNN Classifier')
78     plt.show()
79
80     # Compute and print classification report
81     # target_names = ['Class 0', 'Class 1']
82     # report = classification_report(y_test, (y_pred > 0.5), target_names=target_names)
83     # print("Classification Report:")
84     # print(report)

```

```

82 # Function to train and evaluate TF model
83 def test_model(dynamic_args, d, activation, loss, X_train, y_train, X_test, y_test, epochs):
84     # Unpack arguments
85     filters, depth, hLayers, batchSize, optimizer = dynamic_args
86     # Clear previous models and graphs
87     tf.keras.backend.clear_session()
88     # Create a new graph for the current process
89     with tf.Graph().as_default():
90         # Create a session for the current graph
91         with tf.compat.v1.Session() as sess:
92             # Remind Tensorflow to keep quiet
93             tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.FATAL)
94
95             # Model creation
96             if hLayers == 1:
97                 model = tf.keras.models.Sequential([
98                     # Convolutional layer with N filters, kernel size of 3x3, ReLU activation,
99                     # and input shape for MNIST
100                     tf.keras.layers.Conv2D(filters, kernel_size=(3, 3), activation=activation,
101                                             input_shape=(d[0], d[1], 1)),
102                     # Pooling layer to reduce dimensionality
103                     tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
104                     # Flatten the output to feed into the dense layer
105                     tf.keras.layers.Flatten(),
106                     tf.keras.layers.Dense(depth, activation=activation),
107                     tf.keras.layers.Dense(10, activation='softmax')
108                 ])
109             else:
110                 model = tf.keras.models.Sequential([
111                     # Convolutional layer with N filters, kernel size of 3x3, ReLU activation,
112                     # and input shape for MNIST
113                     tf.keras.layers.Conv2D(filters, kernel_size=(3, 3), activation=activation,
114                                             input_shape=(d[0], d[1], 1)),
115                     # Pooling layer to reduce dimensionality
116                     tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
117                     # Flatten the output to feed into the dense layer
118                     tf.keras.layers.Flatten(),
119                     tf.keras.layers.Dense(depth, activation=activation),
120                     tf.keras.layers.Dense(depth, activation=activation),
121                     tf.keras.layers.Dense(10, activation='softmax')
122                 ])
123
124             # Compile the model
125             model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
126
127             # Start stopwatch
128             start_time = time.time()
129
130             # Train the model
131             history = model.fit(X_train, y_train, epochs=epochs, batch_size=batchSize,
132                               validation_split=0.2)
133             test_loss, test_accuracy = model.evaluate(X_test, y_test)
134
135             # Predict the test set
136             y_pred = model.predict(X_test)
137             y_pred_labels = np.argmax(y_pred, axis=1)
138
139             # Compute confusion matrix
140             conmat = confusion_matrix(y_test, y_pred_labels)
141
142             # Stop stopwatch
143             end_time = time.time()
144
145             training_time = end_time - start_time
146             print(f"Done with test, t = {training_time}")
147             return filters, depth, hLayers, batchSize, optimizer, test_loss, test_accuracy,
148                 training_time, conmat, y_pred

```

```
143
144 if __name__ == "__main__":
145     main()
```


REFERENCES

- [1] E. Alpaydin, "Introduction to Machine Learning," The MIT Press. Cambridge, Second Edition, pp. 1–20, 2010.
- [2] W. Wolberg, O. Mangasarian, N. Street, W. Street, "Breast Cancer Wisconsin (Diagnostic)," UC Irvine Machine Learning Repository, Available: <https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic> [Accessed: Dec. 02, 2023]
- [3] B. Becker, R. Kohavi, "Adult," UC Irvine Machine Learning Repository, Available: <https://archive.ics.uci.edu/dataset/2/adult> [Accessed: Dec. 02, 2023]
- [4] H. Xiao, K. Rasul, R. Vollgraf, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms," CoRR, Vol. 1708.07747. Available: <http://arxiv.org/abs/1708.07747> [Accessed: Dec. 02, 2023]
- [5] S. Abdulhamit, "Machine Learning Techniques," Practical Machine Learning for Data Analysis Using Python, Academic Press, 2020, pp. 91-202.
- [6] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," JMLR 12, pp. 2825-2830, 2011.
- [7] C.R. Harris, K.J. Millman, S.J. van der Walt, et al. "Array programming with NumPy," Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2. Available: <https://www.nature.com/articles/s41586-020-2649-2> [Accessed: Dec. 05, 2023]
- [8] "itertools — Functions creating iterators for efficient looping," Python 3.12.0 Documentation, Available: <https://docs.python.org/3/library/itertools.html> [Accessed: Dec. 05, 2023]
- [9] The Pandas Development Team, "pandas-dev/pandas: Pandas," Zenodo, feb 2020, 10.5281/zenodo.3509134, Available: <https://doi.org/10.5281/zenodo.3509134> [Accessed: Dec. 05, 2023]
- [10] J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.
- [11] M. L. Waskom, "seaborn: statistical data visualization," Journal of Open Source Software, 2021, 6(60), 3021, <https://doi.org/10.21105/joss.03021> [Accessed: Dec. 05, 2023]
- [12] OpenAI, "ChatGPT: Language Model for Natural Language Processing," [Online], Available: <https://www.openai.com/> [Accessed: Dec. 05, 2023]
- [13] M. Abadi, A. Agarwal, P. Barham, et al., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015, Available: <https://www.tensorflow.org/> [Accessed: Dec. 05, 2023]
- [14] S. Marcel, Y. Rodriguez, "Torchvision the Machine-Vision Package of Torch," in Proceedings of the 18th ACM International Conference on Multimedia, Association for Computing Machinery, Firenze, Italy, 2010, pp. 1485-1488. Available: <https://doi.org/10.1145/1873951.1874254> [Accessed: Dec. 05, 2023]