An Investigation of Classification Methods for Fashion-MNIST

Authors: Thadiel Zancoli, Bevan To, and Matthew Busby

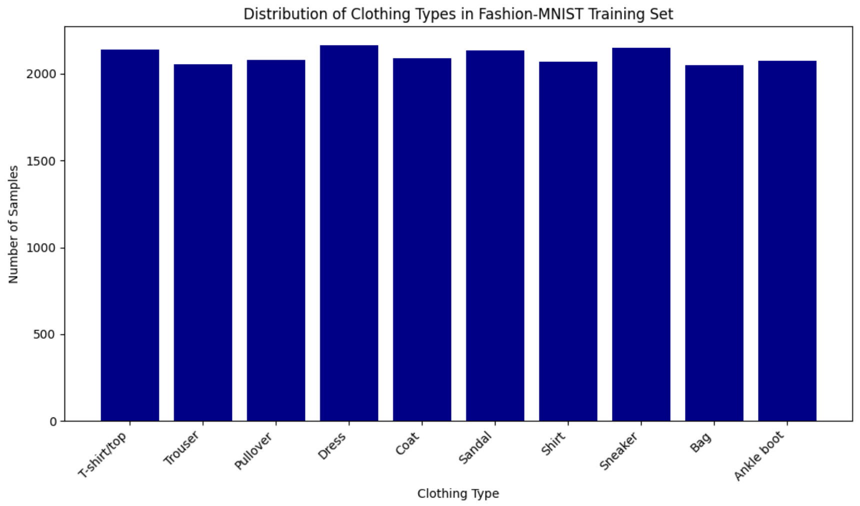
Section 1: Summary

In this project, we investigated several classification methods on the Fashion-MNIST dataset. We used K Nearest Neighbors, Logistic Regression, Feedforward Neural Networks, and Convolutional Neural Networks to conduct this investigation. We aimed to determine the most effective approach for accurately categorizing the fashion items in the dataset. We found there is no perfect algorithm to classify the dataset; however, Convolutional Neural Networks outperformed all the other methods.

Section 2: Data Description

For this project we used Fashion-MNIST, a widely used benchmark dataset in the field of machine learning. Fashion-MNIST comprises 60,000 grayscale images of fashion items, each categorized into one of 10 classes. Each image is 28x28 pixels in size, providing a total of 784 features per sample.

We visualized a subset of sample images from the Fashion-MNIST dataset to gain more of an understanding of the types of fashion items present and their visual features. This visualization helped us identify common patterns and variations within the dataset.



In “Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms” by Han Xiao, Kashif Rasul, and Roland Vollgraf, the authors introduce Fashion-MNIST as a modern replacement for the traditional MNIST dataset. Fashion-MNIST presents a more challenging and diverse collection of fashion-related images, making it a more valuable benchmark for assessing machine learning algorithms. Through comprehensive evaluation, the authors provide valuable insights into the dataset's characteristics and its suitability for benchmarking classification approaches.

Section 3: Classifiers

- K Nearest Neighbors
    - Predicts the class of a new data point based on the majority class of its k nearest neighbors in the feature space, K values tested: 1,3,5,7,11,51,111, Scikit-learn was used
- Logistic Regression
    - A linear classification algorithm that models the probability of a discrete outcome using the logistic function. It finds the best-fitting line that separates the classes in feature space. Some hyperparameters are:
        - Penalty (Regularization): Controls overfitting by penalizing large parameter values. Range: ['l1'(Lasso regularization), 'l2' (Ridge regularization), 'None']
        - C (Regularization strength): A parameter that trades off fitting the training data well against keeping the model simple. Range: [0.001, 0.01, 0.1, 1, 10, 100] Smaller values means stronger regularization.
        - Solver: Algorithm to use in the optimization problem. Range: ['liblinear', 'lbfgs', 'newton-cg', 'sag', 'saga']
    - Used scikit-learn
- Feedforward Neural Networks
    - Use matrix-vector operations in multiple layers of neurons to predict the class from input data, employing activation functions to learn complex, non-linear representations
    - We utilized ReLU activation and stochastic gradient descent for faster computation, with one hidden layer to study parameter effects
    - Hyperparameters investigated include learning rates (0.001, 0.01, 0.1), L2 regularization strength (0.001, 0.01, 0.1), hidden layer node count (32, 64, 128), and batch sizes (128, 256, 512)
    - Scikit-learn was used for training and testing
- Convolutional Neural Networks
    - Classifies new data, such as images, by decomposing them into smaller components and passing them through interconnected layers of neurons to learn patterns for prediction
    - We explored parameters like learning rate (0.001, 0.01, 0.1), layer count (3, 4), filter quantity per layer (3-4), kernel sizes (3x3, 4x4), max pooling (0, 2), dense layer count (2-3), neuron count per dense layer (2-3), and batch size (32, 64)
    - Tensorflow was used for this model

Section 4: Experimental Setup

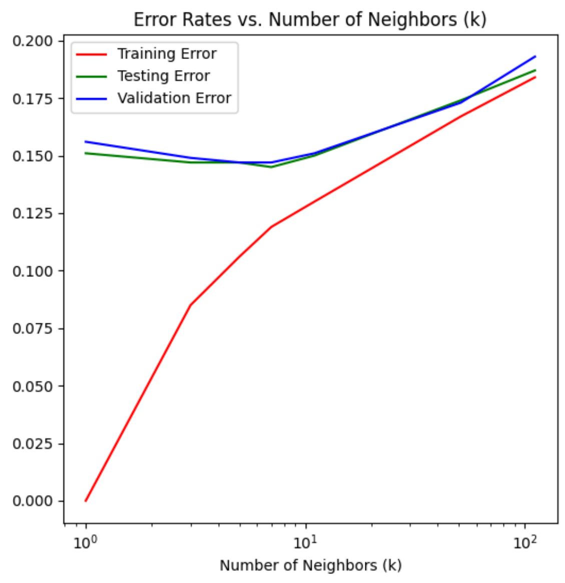
- Metrics:
    - Training Accuracy, Validation Accuracy, and Testing Accuracy
- Data Partitioning:
    - Three subsets: 70% of the data for testing, 15% for validation, and 15% for testing
- Hyperparameter Selection:
    - Hyperparameters vary depending on which classifier is being used

    - K Nearest Neighbors: K (1, 3, 5, 7, 11, 51, 111)
    - Logistic Regression: Penalty (l1, l2, None), C (0.001, 0.01, 0.1, 1, 10, 100), Solver ('liblinear', 'lbfgs', 'newton-cg', 'sag', 'saga')
    - Feedforward Neural Networks: Learning rate(.001, .01,.1), Regularization(.0001,.001, .01,.1), Nodes in the hidden layer (32,64,128,256), and the batch size for the stochastic gradient descent(128, 256, 512).
    - Convolutional Neural Networks: Learning rate (0.001, 0.01, 0.1), number of layers (3, 4), number of filters for each layer (3 - 4), kernel sizes (3x3, 4x4), max pooling (0, 2), number of dense layers (2 - 3), number of neurons in each dense layer (2 - 3), and batch size (32, 64)
  - Selected various different hyperparameters to test the effect of each on the dataset (Specific hyperparameter selection shown in experimental results tables)

Section 5: Experimental Results

K Nearest Neighbors

K	Training Accuracy	Validation Accuracy	Testing Accuracy
1	1.0	0.844	0.849
3	0.915	0.851	0.853
5	0.894	0.853	0.851
7	0.881	0.853	0.855
11	0.870	0.849	0.850
51	0.833	0.827	0.826
111	0.816	0.807	0.813



Increasing k lowers the error until a certain point.
Out of the values of k that were tested, the one
that seems best for the model is 7 because it has the lowest testing error.

Logistic Regression

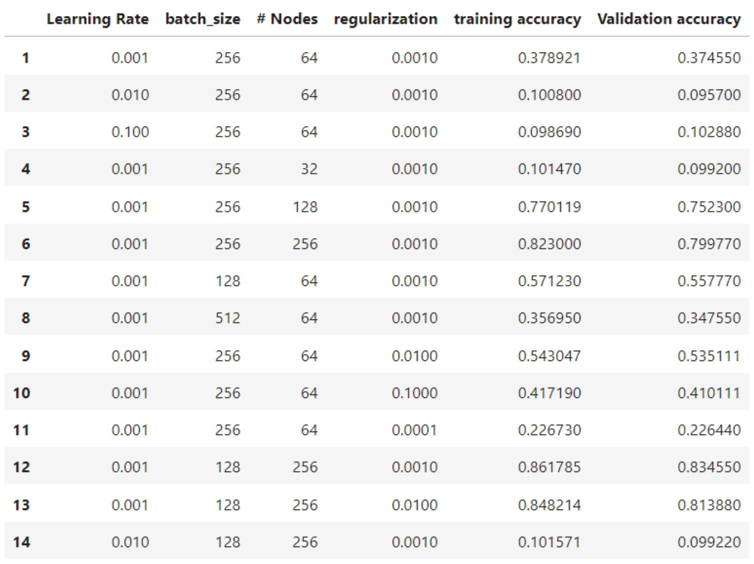
Run #	Penalty	C	Solver	Training Accuracy	Validation Accuracy	Testing Accuracy
1	None	N/A	lbfgs	0.9269	0.8031	0.7978
2	l1	0.01	saga	0.8289	0.8193	0.8134
3	None	N/A	newton-cg	0.9270	0.8007	0.7951

4	None	N/A	sag	0.9068	0.8367	0.8267
5	l1	10	liblinear	0.9095	0.8184	0.8071
6	l2	0.001	saga	0.8551	0.8394	0.8298
7	l2	0.01	lbfgs	0.8827	0.8521	0.8395
8	l2	0.1	sag	0.9025	0.8425	0.8302
9	l2	1	liblinear	0.9000	0.8334	0.8217
10	l2	100	newton-cg	0.9275	0.8001	0.7943

Overall, the results vary dramatically depending on the different hyperparameters being used. This ultimately shows there are multiple ways of achieving a desirable result from logistic regression models. There is no one true best set of hyperparameters, especially when considering your own individual goal from the model.

FeedForward Neural Networks

From these results you can see that increasing the learning rate drastically increased the error from both training and validation error no matter the other parameters, which means that feedforward or 1 hidden layer neural networks are very sensitive to overshooting convergence with learning rate. While regularization did not have much effect on the error, batch size and number of nodes did successfully decrease the error, and so when selecting the smallest batch size with the highest # of nodes, as well as some regularization, we get the parameters that give us the smallest validation error. When putting this set of parameters with the testing data, we received a testing accuracy of : .81422 or 81.4%



	Learning Rate	batch_size	# Nodes	regularization	training accuracy	Validation accuracy
1	0.001	256	64	0.0010	0.378921	0.374550
2	0.010	256	64	0.0010	0.100800	0.095700
3	0.100	256	64	0.0010	0.098690	0.102880
4	0.001	256	32	0.0010	0.101470	0.099200
5	0.001	256	128	0.0010	0.770119	0.752300
6	0.001	256	256	0.0010	0.823000	0.799770
7	0.001	128	64	0.0010	0.571230	0.557770
8	0.001	512	64	0.0010	0.356950	0.347550
9	0.001	256	64	0.0100	0.543047	0.535111
10	0.001	256	64	0.1000	0.417190	0.410111
11	0.001	256	64	0.0001	0.226730	0.226440
12	0.001	128	256	0.0010	0.861785	0.834550
13	0.001	128	256	0.0100	0.848214	0.813880
14	0.010	128	256	0.0010	0.101571	0.099220

Convolutional Neural Networks

Learning Rate	# of Layers	# of Filters	Kernel sizes	Max Pooling	# of Dense Layers	# of Neurons in each Dense Layer	Batch Size	Training Accuracy	Validation Accuracy	Testing Accuracy

0.001	3	(32,64,64)	3x3 (all)	2 (2x2)	2	(64,10)	32	0.845	0.827	0.837
0.001	3	(64,64,128)	3x3 (all)	2 (2x2)	2	(64,10)	32	0.869	0.856	0.858
0.01	3	(64,64,128)	3x3 (all)	2 (2x2)	2	(64,10)	32	0.848	0.838	0.832
0.1	3	(64,64,128)	3x3 (all)	2 (2x2)	2	(64,10)	32	0.099	0.098	0.106
0.001	4	(64,128,128,256)	3x3 (all)	2 (2x2)	2	(64,10)	32	0.884	0.876	0.872
0.001	4	(64,128,128,256)	3x3 (all)	2 (2x2)	3	(64,32,10)	32	0.883	0.873	0.876
0.001	4	(64,128,128,256)	3x3 (all)	2 (2x2)	3	(64,32,10)	64	0.882	0.872	0.874
0.001	4	(64,128,128,256)	4x4 (all)	2 (2x2)	3	(64,32,10)	64	0.885	0.873	0.873
0.001	4	(64,128,128,256)	4x4 (all)	0	3	(64,32,10)	64	0.869	0.856	0.862

Increasing the learning rate seems to lower the testing accuracy. This could be because the model overshot the convergence point. Adding another convolutional layer increased the testing accuracy. Adding another dense layer and increasing the batch size had minimal effects on the testing accuracy.

Section 6: Insights

While conducting these experiments and this project we have come to the conclusion that there is no one perfect algorithm or way to classify the fashion-mnist dataset, and that it is much more difficult to fine tune then the regular MNIST dataset which was able to achieve up to 95% accuracy while we achieved at most 85%. The highest accuracy came from the Convolutional Neural Networks which makes the most sense logically since it is oftentimes practically used for image classification and analysis while the other methods were not. The lowest best testing accuracy ends up coming from the Feed Forward Neural Network closely followed by Logistic Regression which shows that those methodologies are better suited for concrete numerative data as opposed to qualitative images, while KNN showcases itself to be more flexible in its uses and applications.