CLASSIFICATION OF HANDWRITTEN DIGITS USING MACHINE LEARNING

Carlsen, J.[*]

(Dated: December 18, 2023)

We provide results from three classification models; a logistic regression (LogReg) model, a neural network (NN) model, and a decision tree (DT) model. The dataset is the *Pen-Based Recognition of Handwritten Digits* (Alpaydin and Alimoglu 1998), which consists of 10992 instances. The UC Irvine Machine Learning Repository provides benchmark model performance values in the form of accuracy and precision from different models including a logistic regression model and a neural network classification model. Missing is a decision tree model. We find that our NN model has scores comparable to the benchmarks, with an accuracy of 0.956, and a precision of 0.958. The DT model's accuracy is 0.897, and the precision is 0.899.

I. INTRODUCTION

The famous data scientist Arthur Samuel defined machine learning as the field of study that gives computers the ability to learn without being explicitly programmed (Mahesh 2020). In Samuel (2000), Samuel gives conclusive evidence to this statement, when his program learns to become a better checkers player than himself, with the use of machine learning methods. When asked *Are you smarter than humans?*, ChatGPT (OpenAI 2020) replies that it possesses access to a vast amount of information and can process data quickly. It is also "aware" of its limitations when it comes to various aspects of human intelligence such as emotions, creativity, and common sense. With the ability to aid humans in almost all aspects of life, AIs and machine-learning algorithms prove themselves valuable, especially in complex data analysis.

As a wide range of studies have found, machine-learning classification models tend to produce higher accuracy than traditional parametric classifiers (Aaron E. Maxwell and Fang 2018). In this regard, we present the results from three classification models when trained to label handwritten digits. All models are compared to the SciKit-Learn (SKL) python package (Buitinck *et al.* 2013) to test the validity of our models. We describe these models and the dataset used for training in Section II, and all results in Section IV. A discussion of our findings is found in Section V, where we also compare results with baseline model performance on the specific dataset at hand. Lastly, we conclude the paper in Section VI.

II. METHOD

In this section, we present the main ideas behind the three classification models proposed in this paper: the logistic regression (LogReg) model, the neural network (NN) model, and the decision tree (DT) model. We briefly present the LogReg and NN methods and refer the reader to our previous paper (Carlsen 2023) (Paper 1, hereafter), for a complete description. Both the LogReg and NN methods use the stochastic gradient descent (SGD) update method with a constant learning rate η, presented in Algorithm 1.

Algorithm 1 Stochastic gradient descent

```
Require: Initial parameter $\boldsymbol{\theta}$
Require: Learning rate $\eta$
  while stopping criterion not met do
      Sample a minibatch
      Compute gradient: $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{\theta}}$
      Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \boldsymbol{g}$
  end while
```

We implement the SGD method such that the size controls the number of each minibatch. In the case of LogReg, the parameter $\boldsymbol{\theta}$ is the optimal parameter $\boldsymbol{\beta}$. For the NN method, it is the weights and biases of each layer.

A. Logistic regression

The LogReg model takes the base-10 logarithm of the odds as a regression function of the predictors (LaValley 2008). The cost function is given as the cross-entropy

$$C_{\text{LogReg}}(\boldsymbol{\beta}) = -\frac{1}{n}\sum_{n}\Big[y_n \log_{10}(\tilde{y}_n + \delta)\Big], \qquad (1)$$

where we use $\delta = 10^{-9}$ to avoid singularities. The prediction is made using the Sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

on the initial prediction $\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta}$. We then pick the highest value using the argmax function.

B. Neural network

The NN method we consider is a fully connected feed-forward NN. This means that each neuron in every layer is connected to each neuron in the neighboring layer. An illustration of a fully connected NN is shown in Figure 1. We re-

[*] https://github.com/JohanCarlsen/

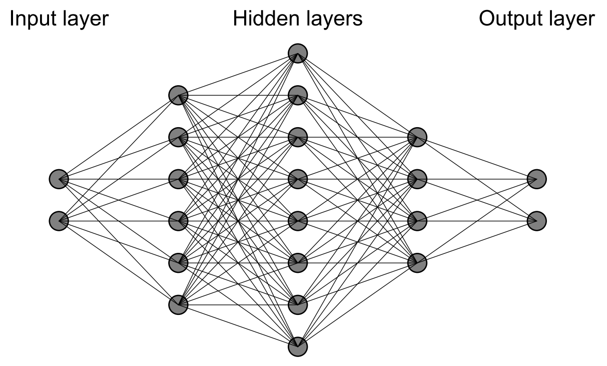


Figure 1: Illustration of a fully connected neural network with two nodes in the input and output layers and three hidden layers. Each neuron of a layer is connected to each neuron in the adjacent layers.

fer to the structure of a NN as $[n_{\text{in}}, n_{\text{h1}}, n_{\text{h2}}, n_{\text{h3}}, \dots, n_{\text{out}}]$, i.e. the first and last number are the input and output layers, respectively, with the hidden layers in between them. Thus, the structure of the NN in Figure 1 is $[6, 8, 4, 2]$. Each neuron in a given layer has a bias value, and each connection has a weight value. Information is fed through the network as

$$\begin{aligned} \boldsymbol{z}^\ell &= \boldsymbol{a}^{\ell-1}\boldsymbol{W}^\ell + \boldsymbol{b}^\ell, \\ \boldsymbol{a}^\ell &= f(\boldsymbol{z}^\ell), \end{aligned}$$

where $\boldsymbol{z}^\ell$, $\boldsymbol{W}^\ell$, and $\boldsymbol{b}^\ell$ are the input, weights, and bias of layer ℓ, and $\boldsymbol{a}^\ell$ is the activation of the ℓ'th layer, with f being the activation function. Then, the error is computed by the back-propagated (backprop, hereafter) error method as

$$\begin{aligned} \boldsymbol{\delta}^L &= \tilde{\boldsymbol{y}} - \boldsymbol{y}, \\ \boldsymbol{\delta}^\ell &= \boldsymbol{\delta}^{\ell+1}\left(\boldsymbol{W}^{\ell+1}\right)^T f'(\boldsymbol{z}^{\ell+1}), \end{aligned}$$

with $\ell = L$ being the output layer. The gradient of the weights and biases then become

$$\begin{aligned} \nabla^\ell_{\boldsymbol{W}} &= \left(\boldsymbol{a}^\ell\right)^T \boldsymbol{\delta} + \boldsymbol{W}^\ell \alpha, \\ \nabla^\ell_{\boldsymbol{b}} &= \sum_i \delta_{ij}, \end{aligned}$$

which are then used by the SGD algorithm. From this, the backprop method repeatedly adjusts the weights and biases of the connections in the network to minimize the difference between the predicted vector $\tilde{\boldsymbol{y}}$ and the true vector $\boldsymbol{y}$ (Rumelhart, Hinton, and Williams 1986).

When solving classification problems using an NN model, we use the leaky rectified linear unit (LReLU) activation function for the hidden layers and the Softmax function for the output layer. The cost function is the same as for the LogReg model, given by Eq. 1. The LReLU is given by

$$\text{LReLU}(x) = \begin{cases} x & x > 0 \\ \delta x & \text{otherwise} \end{cases},$$

where we use $\delta = 0.01$. The fact that the derivative of the LReLU is non-zero can solve the problem of so-called "dead neurons" in NNs (Xu *et al.* 2020), where the LReLU unit becomes inactive. The Softmax function consists of an exponential evaluation of a vector element normalized by the summation of the exponential of all elements of that vector. In Gao and Pavel (2018), the version of the Softmax function we use is labeled the standard Softmax function and is given by

$$f(x) = \frac{\exp\{\boldsymbol{x} - \max(\boldsymbol{x})\}}{\sum_n \exp\{x_n\} + \delta},$$

where we use $\delta = 10^{-9}$.

For the initialization of the weights, we implement the LeCun method (LeCun *et al.* 1998), where for a layer ℓ with size n, the weights are obtained from the normal distribution as

$$W_i^\ell = \mathcal{N}\left(0, \sqrt{\frac{1}{n^{\ell-1}}}\right),$$

and all biases are set to 0.01.

C. Decision tree

Our new model is a classification tree, implemented using the supervised machine-learning algorithm CART (Classification And Regression Tree). The DT is constructed to partition the input data into subsets based on the variable's value. Figure 2 shows an illustration of a DT with a maximum depth of five. The tree splits the input data by evaluating the values in the design matrix. Values lower than *Split value* in column number *Feature* are sent to the left, and the remainder to the right. Then, the splitting repeats until a depth of five is reached.

A DT model chooses the split by the impurity of the dataset. That is, whether the subset is homogeneous with respect to the class distribution of its examples (Laber, Molinaro, and Pereira 2018). We implement two common ways to calculate the impurity; the Gini impurity

$$\text{Gini} = 1 - \sum_{i=1}^{n} p_i^2, \tag{2}$$

and the entropy impurity

$$\text{Entropy} = -\sum_{i=1}^{n} p_i \log_2(p_i), \tag{3}$$

where n is the number of unique classes in the dataset, and p_i is the probability of an element in the dataset belonging

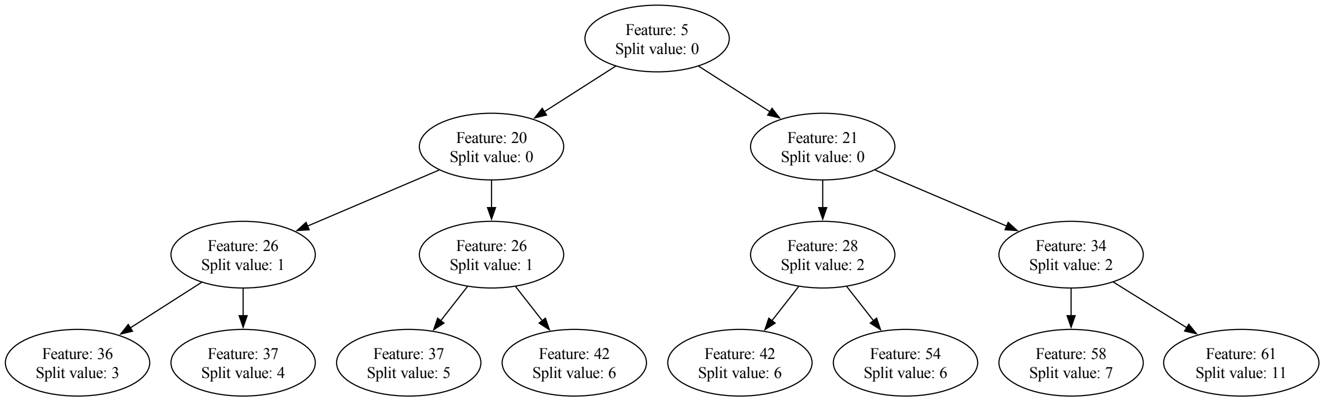


Figure 2: Illustration of a decision tree using the CART algorithm. The input data is partitioned into subsets, aiming to minimize the impurity of the subset. The feature value is a row in the design matrix, and the split value controls how to partition the data. Values lower than the split value are sent to the left, and the others to the right.

to the i'th class. The algorithm splits the nodes based on impurity reduction and recursively grows the tree until a stopping criterion is met (e.g. maximum depth). When the tree has grown to fit the data, a reduction is performed by traversing the tree, and splitting the data until a leaf node is reached, i.e. a node of the tree returning a prediction.

D. Score metrics

When evaluating the performance of a model, we consider three commonly used metrics; (i) the accuracy of the model, given as the ratio of correct classifications to the total number of classes, (ii) the precision of the model, given as the ratio of the number of correct classifications to the sum of true positive and false positive classifications, and (iii) the recall of the model, given as the ratio of correct classifications to the sum of true positive and false negative classifications (Junker, Hoch, and Dengel 1999). These metrics are summarized as

$$\text{Accuracy} = \frac{\text{Correct classifications}}{\text{All classifications}},$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}},$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}},$$

where TP, FP, and FN are true positive, false positive, and false negative, respectively.

A visualization-type metric is our version of a confusion matrix, where the rows represent the true labels, and the columns represent the predicted labels. In this way, if all predictions are correct, the main diagonal of the confusion matrix contains the frequency of the predictions while all other elements are zero. This variant is different from e.g. the definition of the confusion matrix in Davis and Goadrich (2006), but similar to the one SKL uses.

III. DATASET

Our models are trained and tested on the *Pen-Based Recognition of Handwritten Digits* (Alpaydin and Alimoglu 1998) dataset consisting of 250 samples from 44 writers. The subjects wrote digits between 0 and 9 on an LCD display, and the data was stored together with 16 features for each subject. The final dataset has 10992 instances of integer types, ranging from 0 to 100. What we aim to classify is naturally which number or numbers are written given the features. Unfortunately, the features are not disclosed, only named Attribute1, Attribute2, and so on.

The dataset is under the public domain from the UC Irvine Machine Learning Repository, where baseline model performance is also presented. For this paper, the baseline performance of logistic regression (LR) and neural network classification (NNC) is of most interest. The fact that a decision tree model is not reported is intriguing. Table I

Model	**Accuracy**	**Precision**
LR	0.905	0.910
NNC	0.969	0.969

Table I: Baseline model performance on the *Pen-Based Recognition of Handwritten Digits* (Alpaydin and Alimoglu 1998) dataset, reported by the UC Irvin Machine Learning Repository for the logistic regression (LR) and neural network classification (NNC) models.

shows the baseline model performance regarding the accuracy and precision using LR and NNC. With an accuracy of 90.480 for the LR, 96.941 for the NNC, and a precision of 90.947 for the LR, and 96.931 for the NNC, we recognize that these are relatively good models. In the next section, we present the results from our models.

IV. RESULTS

In this section, we present the results and how we obtained them using the LogReg, NN, and DT models. Table II shows the results from each model after the parameters have been selected to optimize the outcome. As seen, our NN model performs the best with an accuracy of 0.956, precision of 0.958, and recall of 0.957. The DT and LogReg models have accuracy, precision, and recall of 0.897 and 0.675, 0.899 and 0.734, and 0.897 and 0.673, respectively. In the following sub-sections, we also include the scores

Model	Accuracy	Precision	Recall
LogReg	0.675	0.734	0.673
NN	0.956	0.958	0.957
DT	0.897	0.899	0.897

Table II: Results from the logistic regression (LogReg), neural network classification (NN), and decision tree (DT) models using their respective optimal parameters.

using the SKL library. Note that the structures of the SKL models may differ from ours, so the scores may not be completely analogous.

A. Logistic regression

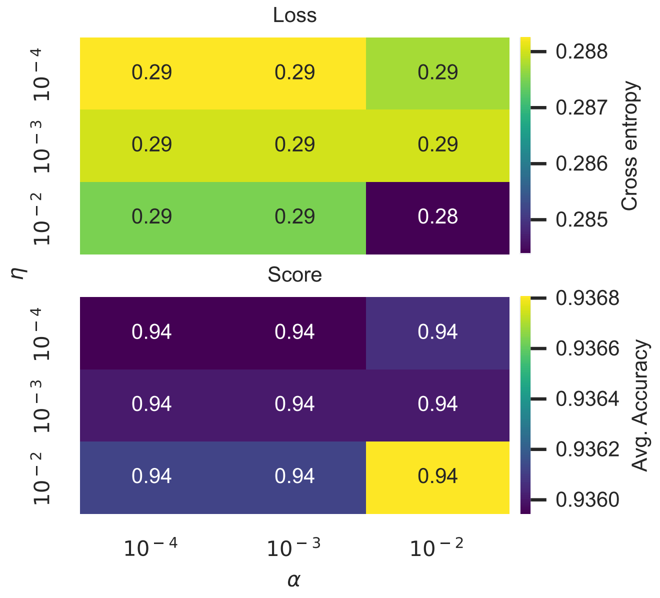


Figure 3: Heat maps showing the losses (top) and scores (bottom) for different α and η values with the LogReg model.

Our LogReg model is initialized with the minibatch size set to 1000, and a maximum of 10^5 epochs to perform. However, an early stopping clause is implemented, forcing the

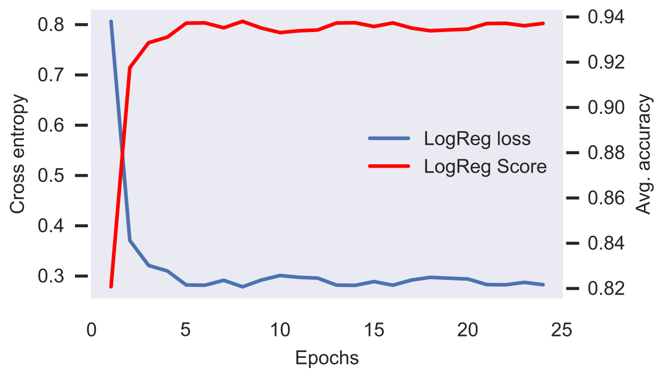
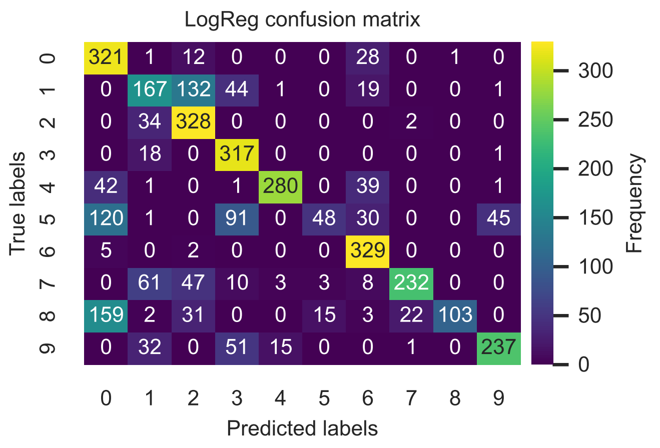


Figure 4: Evolution curves of the average loss (blue) and score (red) for the LogReg model.

iteration to stop if the loss does not improve after 10 iterations within each epoch. During training, the model evaluates the loss of a separate validation set, which is where the early stopping criterion is obtained. The tunable hyperparameters are the learning rate η and the L2 penalty term α. Figure 3 shows a heat map of the loss and score (score given as the average accuracy) of the model in the hyperparameter space of $\eta, \alpha \in [10^{-2}, 10^{-3}, 10^{-4}]$. The top panel shows the loss (cross-entropy) being lowest for $\eta = \alpha = 10^{-2}$, while the bottom panel shows the score is highest for $\eta = 10^{-4}$ and $\alpha = 10^{-2}$. Note that though the numeric values are the same for all losses and scores, the truncation to two decimals hides the real value. Refer to the color bar for a more precise value. From these results, we chose $\eta = 10^{-4}$, and $\alpha = 10^{-2}$ as the optimal hyperparameters. With these values, we plot the evolution of the loss and score as functions of the number of epochs before the early stopping clause is activated. This is shown in Figure 4. We see that the model reaches a loss of ~ 0.82, and a score of ~ 0.94 after 24 epochs. The



True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	321	1	12	0	0	0	28	0	1	0
1	0	167	132	44	1	0	19	0	0	1
2	0	34	328	0	0	0	0	2	0	0
3	0	18	0	317	0	0	0	0	0	1
4	42	1	0	1	280	0	39	0	0	1
5	120	1	0	91	0	48	30	0	0	45
6	5	0	2	0	0	0	329	0	0	0
7	0	61	47	10	3	3	8	232	0	0
8	159	2	31	0	0	15	3	22	103	0
9	0	32	0	51	15	0	0	1	0	237

Figure 5: Confusion matrix for the LogReg model.

confusion matrix shown in Figure 5 shows that labels 0, 2, 3, and 6 are correctly predicted the most, while labels 4, 7, and 9 are more often correctly classified than not. The remaining predictions are categorized as ambiguous, given the number of incorrect predictions relative to the number of correct ones. We obtain a final accuracy of 0.675, a precision of 0.734, and a recall of 0.673. The fact that these values are low compared to the score evolution curve described earlier is that the calculation is different. During training, the average accuracy is calculated as the sum of correct classifications divided by the total number of classifications to save computation time. However, the final scores are calculated using the definitions of the metrics in Section II D, and are thus the ones related to the actual performance. A model using the SKL library yields an accuracy of 0.901, a precision of 0.902, and a recall of 0.901. Remember that different architectures can affect the model performance.

B. Neural network

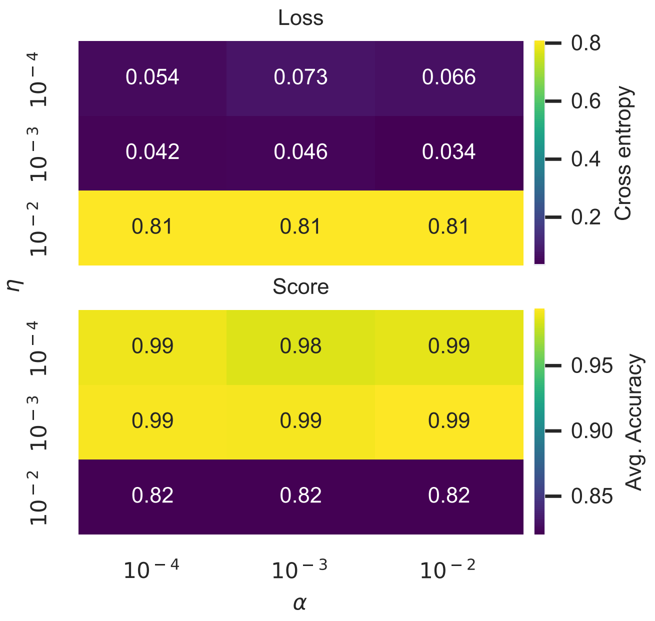


Figure 6: Heat maps showing the losses (top) and scores (bottom) for different α and η values with the neural network model.

We implement the NN model with a layer structure of [16, 50, 10], i.e. 16 neurons in the input layer, 50 neurons in the hidden layer, and 10 neurons in the output layer. Similar to the LogReg model, we set the size of each minibatch to 1000, and the maximum number of epochs to 10^5, with the same early stopping clause of 10 iterations without loss-improvement. Also similar are the values for the learning rate and L2 penalty, namely 10^{-2}, 10^{-3}, and 10^{-4}. The heat map of Figure 6 shows the loss in the top panel, and the score in the bottom panel. Different from the LogReg model is the fact that the two maps are correlated in the sense that high losses correspond to low scores. We can then easily choose $\eta = \alpha = 10^{-3}$ as the optimal hyper-parameters for the NN model. The evolution of the loss

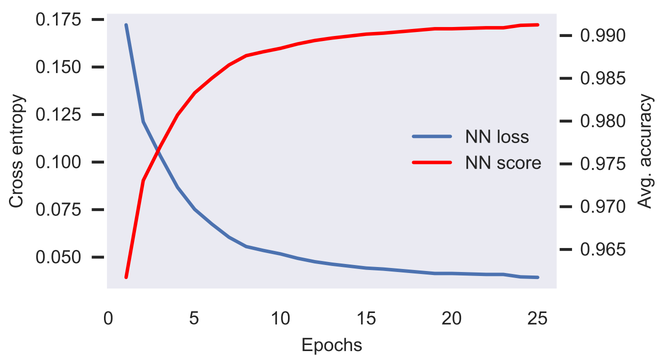
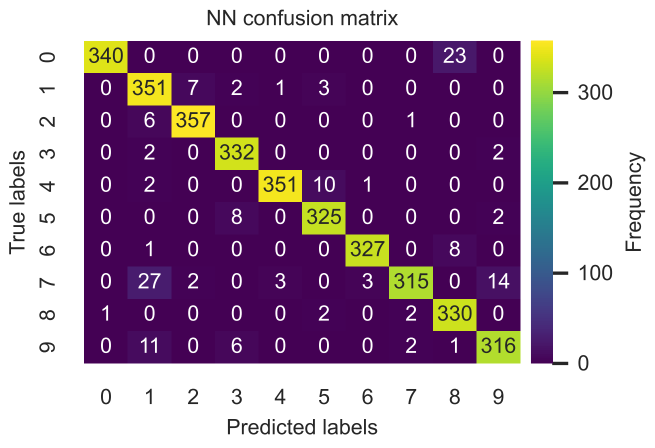


Figure 7: Evolution curves of the average loss (blue) and score (red) for the neural network model.

and score from training is shown in Figure 7, which reveals that the NN model reaches a loss of $\sim$ 0.960, and a score of $\sim$ 0.990 after 25 epochs. Analyzing the confusion matrix shown in Figure 8 indicates that the prediction is mostly correct, with the main diagonal holding nearly all classifications. Our NN model results in an accuracy of 0.956, a precision of 0.958, and a recall of 0.957. This time, the SKL model results in an accuracy of 0.873, a precision of 0.875, and a recall of 0.875. It is interesting to note that the SKL model has to use 2000 epochs to reach these values.



True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	340	0	0	0	0	0	0	0	23	0
1	0	351	7	2	1	3	0	0	0	0
2	0	6	357	0	0	0	0	1	0	0
3	0	2	0	332	0	0	0	0	0	2
4	0	2	0	0	351	10	1	0	0	0
5	0	0	0	8	0	325	0	0	0	2
6	0	1	0	0	0	0	327	0	8	0
7	0	27	2	0	3	0	3	315	0	14
8	1	0	0	0	0	2	0	2	330	0
9	0	11	0	6	0	0	0	2	1	316

Figure 8: Confusion matrix for the neural network model.

C. Decision tree

Our analysis of the DT model consists of testing different tree depths for both the Gini and entropy impurity (Eqs. 2 & 3). In Figure 9, we show the accuracy of the final model as a function of depth, for both impurities. We observe that a model using entropy as the impurity function has an accuracy of $\sim$ 90 with a depth of 10. With the use of Gini, a tree depth of 20 is required. From this, we proceed with a DT model with 10 as the maximum depth, and entropy as the impurity function. This choice is mostly based on computation time, as well as a risk of over-fitting the model

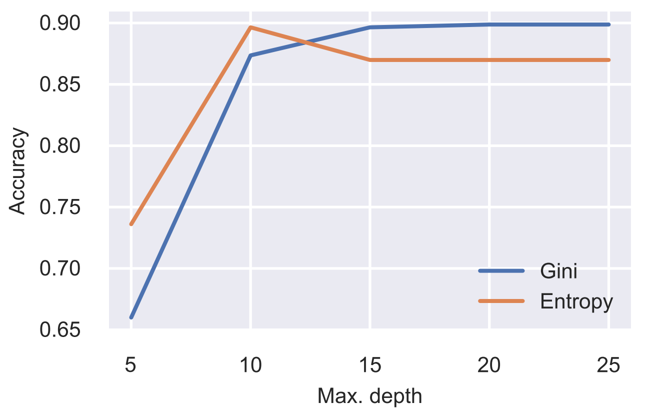
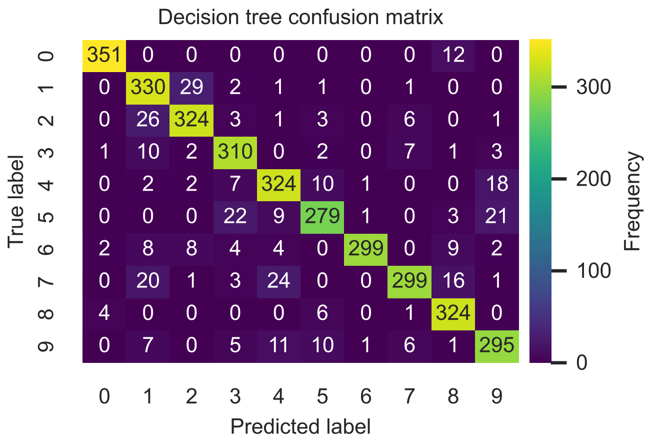


Figure 9: Accuracy on test-set using Gini (blue) and entropy (orange) as functions of maximum tree depth.

with increasing tree depth. As seen in Figure 10 which shows the confusion matrix for the DT model, most of the predictions are correct, with the major diagonal holding the most occurrences for each label. The DT model results in an accuracy of 0.897, a precision of 0.899, and a recall of 0.897. This time, the SKL decision tree yields an accuracy of 0.901, a precision of 0.905, and a recall of 0.901.



	0	1	2	3	4	5	6	7	8	9
0	351	0	0	0	0	0	0	0	12	0
1	0	330	29	2	1	1	0	1	0	0
2	0	26	324	3	1	3	0	6	0	1
3	1	10	2	310	0	2	0	7	1	3
4	0	2	2	7	324	10	1	0	0	18
5	0	0	0	22	9	279	1	0	3	21
6	2	8	8	4	4	0	299	0	9	2
7	0	20	1	3	24	0	0	299	16	1
8	4	0	0	0	0	6	0	1	324	0
9	0	7	0	5	11	10	1	6	1	295

Figure 10: Confusion matrix for the decision tree model with entropy as the loss function.

V. DISCUSSION

Our LogReg model's scores being the poorest among the three models is not unexpected, but the low scores of < 0.735 are surprising. According to LaValley (2008), unordered categories as we analyze here should not impose major difficulties for a logistic regression model. Obviously, there is a potential pitfall in our implementation of the in-training computation of the accuracy, where we suspect that the final scores can be affected by the way the model trains. What we mean by this is that since the model checks the loss and score after each iteration, an incorrect score calculation can cause the early stopping clause to activate too soon. That being said, we do not have great confidence in this being the (only) cause, as the early stopping only checks for improvement in the loss, and not the score. We acknowledge the correlation between the confusion matrix of Figure 3 and the resulting scores, which leads us to believe our model is valid, meaning that it does not contain major flaws which affect the end results. The SKL model we compare with outperforms our model, with all scores > 0.901. We again emphasize the fact that creating an identical model using SKL is hard or impossible, but the general algorithms used in both models should not be too different. Also, SKL is of course a highly sophisticated package, so any errors in our source code will naturally make a difference when compared to SKL, however small they might be.

We find our NN model's performance exceeding our expectations with regard to the consistently high scores of > 0.95 for accuracy, precision, and recall. In Féraud and Clérot (2002), neural networks are labeled *frustration tools*, while still exhibiting excellent modeling performance. We experience with our model that choosing the structure of the hidden layers is the only uncertain element of building the network. As seen in the heat maps of Figure 6, one can with a high level of confidence choose the values for the hyperparameters η and α with ease. The confusion matrix in Figure 8 also speaks for itself, with overall excellent predictions. This time, when we compare it to the SKL model, we experience the true difficulty when trying to make a comparable model. The SKL model scores below 0.876 for all metrics, when we let it run for 2000 epochs. Obviously, this is not a good comparison to our model, which reaches its optimal configuration after 25 epochs. It is still our view that the way we write the source code for our model is easy to use and understand, and that the generality of the SKL library may be affecting its ability to be tuned to the user's specific needs.

When it comes to our DT model, the way to optimize it really only consists of adjusting two parameters; the impurity function and the maximum depth. The CART algorithm is most often said to use the Gini impurity (e.g. Rutkowski *et al.* 2014; Gulati, Sharma, and Gupta 2016). However, we find that the entropy impurity yields similar results with a lower tree depth. It is our belief that this can lower the risk of over-fitting. This is in contrast to Gulati, Sharma, and Gupta (2016), where they find that using the Gini index both reduces over-fitting and is (a little) faster to compute. From our analysis shown in Figure 9, we see the peak of accuracy for the entropy impurity at a depth of 10. The fact that our DT model converges to an accuracy value as the depth gets higher is expected, as the CART algorithm only splits the data in two each time. The comparison to a decision tree model using the SKL library shows that the scores from our DT model using the entropy are quite similar to the SKL model, where accuracy, precision, and recall are 0.897, 0.899, and 0.897 for the DT model and 0.901, 0.905, and 0.901 for the SKL model.

Comparing the LogReg and NN models to the baseline model performances in Table I reveals that our LogReg model's accuracy of 0.675 and precision of 0.734 is poor compared to the LR baseline accuracy of 0.905 and precision of 0.910. The opposite is the case for our NN model, where the accuracy of 0.956 and precision of 0.958 are arguably comparable with the NNC baseline accuracy and precision, both of 0.969. As all baseline models score higher than 0.9 on both accuracy and precision, our DT model may not (yet) have what it takes to compete with these models. However, we urge the scientific community to provide their results on as many datasets as possible.

VI. CONCLUSION

In this paper, we have presented the results from classifying handwritten digits using three models; a logistic regression (LogReg) model, a neural network (NN) model, and a decision tree (DT) model. We also compared results to the closest model we managed to create using the SciKit-Learn (SKL) library for each model. For both the LogReg and NN models, we searched for the optimal hyperparameters for the learning rate and L2 penalty term, while for the DT model, we analyzed the results using both the Gini and entropy impurity functions, and the maximum depth of the tree.

We found that our NN model performed the best, and gave results comparable to the baseline model performance for neural network classification reported by the UC Irwin Machine Learning Repository. In particular, we found that the LogReg model's accuracy and precision of 0.675 and 0.734 were poor compared to the baseline, while the NN's accuracy of 0.956 and precision of 0.958 are strong scores in this comparison.

Future research could include using different metrics to evaluate the models as well as running cross-validation and bootstrap routines to gain higher confidence in the results. We also want to emphasize that all of our results are public and free to use to update baselines or similar catalogs. Anyone reading this is also welcome to use and/or modify the source codes available at `https://github.com/JohanCarlsen/fys-stk4155`, as well as providing updates, fixes, and improvements.

REFERENCES

Aaron E. Maxwell, T. A. W.and Fang, F., International Journal of Remote Sensing **39**, 2784 (2018).

Alpaydin, E.and Alimoglu, F., "Pen-Based Recognition of Handwritten Digits," UCI Machine Learning Repository (1998).

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., and Varoquaux, G., in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning* (2013) pp. 108–122.

Carlsen, J., "Using deep learning for regression and classification," (2023), unpublished.

Davis, J.and Goadrich, M., in *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06 (Association for Computing Machinery, New York, NY, USA, 2006) p. 233–240.

Féraud, R.and Clérot, F., Neural Networks **15**, 237 (2002).

Gao, B.and Pavel, L., "On the properties of the softmax function with application in game theory and reinforcement learning," (2018), arXiv:1704.00805 [math.OC].

Gulati, P., Sharma, A., and Gupta, M., Int. J. Comput. Appl. **141** (2016), 10.5120/ijca2016909926.

Junker, M., Hoch, R., and Dengel, A., in *ICDAR* (1999) pp. 713–716.

Laber, E., Molinaro, M., and Pereira, F. M., in *Proceedings of the 35th International Conference on Machine Learning*, Proceedings of Machine Learning Research, Vol. 80, edited by J. Dy and A. Krause (PMLR, 2018) pp. 2854–2862.

LaValley, M. P., Circulation **117**, 2395 (2008).

LeCun, Y., Bottou, L., Orr, G. B., and Müller, K. R., "Efficient backprop," in *Neural Networks: Tricks of the Trade*, edited by G. B. Orr and K.-R. Müller (Springer Berlin Heidelberg, Berlin, Heidelberg, 1998) pp. 9–50.

Mahesh, B., IJSR **9** (2020), 10.21275/ART20203995.

OpenAI,, "GPT-3 (generative pre-trained transformer 3)," AI language model (2020).

Rumelhart, D. E., Hinton, G. E., and Williams, R. J., Nature **323**, 533 (1986).

Rutkowski, L., Jaworski, M., Pietruczuk, L., and Duda, P., Information Sciences **266**, 1 (2014).

Samuel, A. L., IBM Journal of Research and Development **44**, 206 (2000).

Xu, J., Li, Z., Du, B., Zhang, M., and Liu, J., in *ISCC* (2020) pp. 1–7.