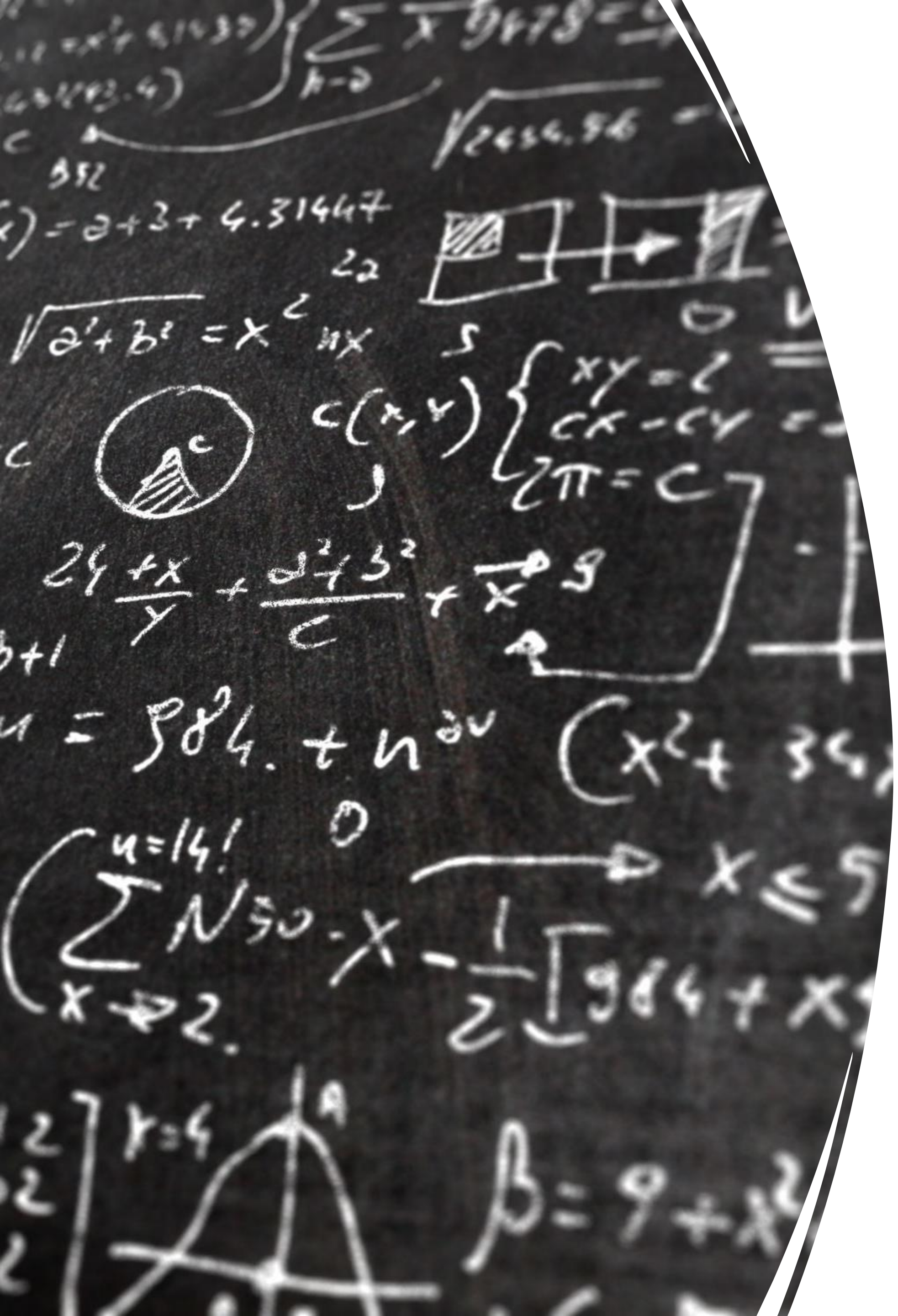


Handwritten Digit Classification

Brandon Park, Jungjae Park, Kyunghwan Min, Minjae Lee, Minseo Jang

Tools

Language

- Python
- Random Forest
- Tree Boosting
- Single Layer Neural Network
- Multilayer Neural Network

Data

- MINIST Data set

Objective

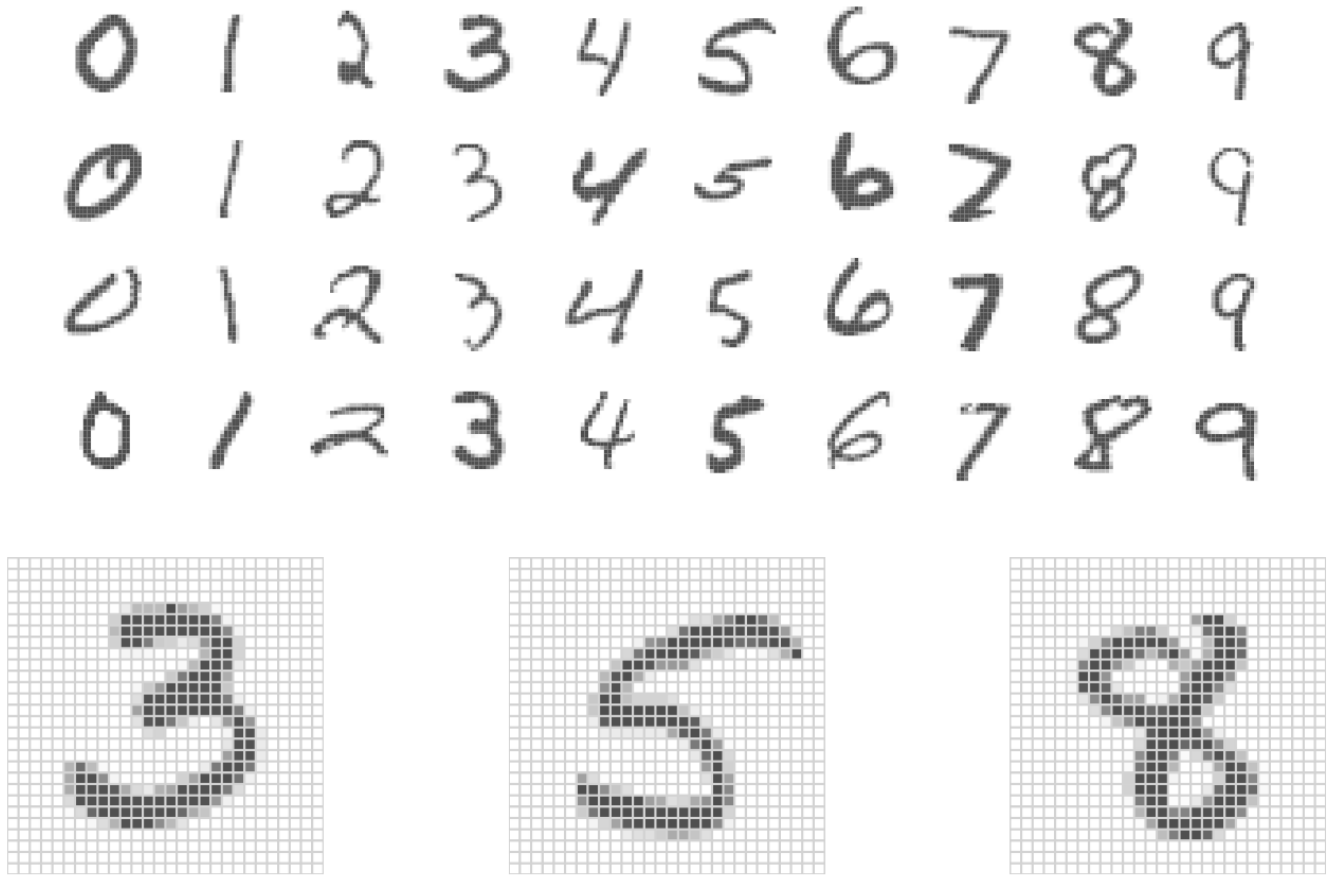
- Compare the Efficiency
- Time Cost
- Accuracy

MNIST Data Set

- Training set: 60000 Images

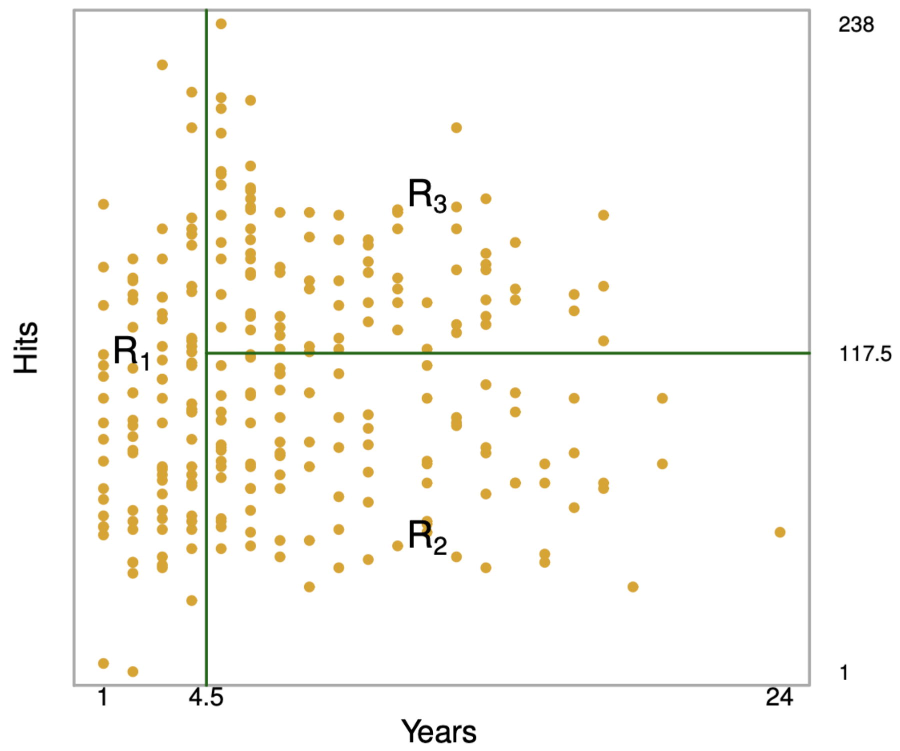
- Test set: 10000 Images

- Each images with 28 x 28 = 764 pixels
- Each pixels have value of 0 – 255



Trees

Years < 4.5
Hits < 117.5
5.11
6.00
6.74



238
117.5
1
R1
R2
R3
Hits
1
4.5
24
Years

Building a Tree

- Splitting a Tree

- Find boxes that minimize classification error rate

$$G = \sum^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}.$$

- Recursive binary splitting
  - Greedy, top-down approach

- Pruning a Tree

- Overfitting -> poor test set performance

- Option 1 : threshold

- Option 2 : pruning – cost complexity pruning

$$\sum_{m=1}^{|T|} \sum_{i:\ x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

Random Forest

- Problem 1 : **high variance**

- Bootstrap : lowering variance by averaging a set of observations

- Problem 2 : **correlation**

- Same as bagging, at each split, a random sample of m predictors are only considered

Boosting

- Does not use bootstrapping
- Grows sequentially

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
   (a) Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).
   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$
   (c) Update the residuals,
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$
3. Output the boosted model,
   $$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

Three parameters

B : Number of Trees

λ : Shrinkage parameter

d : Number of Splits

Result : Random Forest

- Compared m values :
- 350 (p/2) vs 26 (√p)
- 26 was better : less correlation

- Confusion matrix : strong diagonal

```
Number of predictors = 700
mtry1 (0.5p): 350
mtry2 (sqrt(p)): 26

--- Timing Random Forest (mtry1) ---
Time: 1396.43798494339 seconds

--- Timing Random Forest (mtry2) ---
Time: 432.87593960762024 seconds
OOB Error (0.5p): 0.03166666666666662
OOB Error (sqrt(p)): 0.028433333333333331
Best mtry selected = 26

--- Timing Final Random Forest ---
Time: 432.5059537887573 seconds
Test Misclassification Rate: 0.0277

Confusion Matrix:
[[ 970    0    0    0    0    3    2    1    3    1]
 [    0 1124    2    3    0    2    2    0    1    1]
 [    6    0 1000    5    3    0    4    8    6    0]
 [    0    0    9  976    0    5    0    9    9    2]
 [    1    0    2    0  958    0    4    0    2   15]
 [    3    0    0   10    3  862    6    1    5    2]
 [    6    3    0    0    3    3  939    0    4    0]
 [    1    2   17    1    1    0    0  994    1   11]
 [    3    0    5    7    2    5    3    3  935   11]
 [    5    5    2    9   10    2    1    4    6  965]]
```

Result : Tree Boosting

- Increasing depth led to lower error
- Increasing number of trees led to lower error

- **Cannot see the effect of overfitting**

20000	3	0.0512	1926.30
20000	4	0.0372	2571.23
20000	5	0.0305	3405.74
20000	6	0.0282	4352.93
30000	1	0.1211	1341.37
30000	2	0.0651	2081.97
30000	3	0.0426	2870.53
30000	4	0.0314	3870.28
30000	5	0.0272	5155.69
30000	6	0.0261	6375.44

```
Best Parameters → Trees: 30000 | Depth: 6 | Validation Error: 0.0261
```

Overfitting

- Original task: **10-class digit classification (0–9)**
- Modified task: **Binary classification (digit 3 vs digit 8)**
- Reduces problem complexity and allows the model to reach high capacity more quickly

```
##################### result ##############################
#Trees: 10000 | Depth: 4 | Validation Error Rate: 0.0129
#Trees: 10000 | Depth: 5 | Validation Error Rate: 0.0109
#Trees: 10000 | Depth: 6 | Validation Error Rate: 0.0121
#Trees: 20000 | Depth: 4 | Validation Error Rate: 0.01
#Trees: 20000 | Depth: 5 | Validation Error Rate: 0.0083
#Trees: 20000 | Depth: 6 | Validation Error Rate: 0.0104
#Trees: 30000 | Depth: 4 | Validation Error Rate: 0.0092
#Trees: 30000 | Depth: 5 | Validation Error Rate: 0.0083
#Trees: 30000 | Depth: 6 | Validation Error Rate: 0.0092
#Best Parameters → Trees: 20000 | Depth: 5 | Validation Error: 0.0083
#Test Misclassification Rate (Boosted Tree): 0.0055
###################################################################
```

Single Layer Neural Network

- Hidden layer function : RELU
- Regularization : Dropout
- Searched over :
- Neutrons : 50, 100, 150, 200
- Dropout : 0.2, 0.3, 0.4, 0.5
- Batch size : 100, 200, 300
- Learning rate : 0.01 (Adam)
- Epoch : 5000

- **Best Parameters :**

Neurons: 150 | Dropout: 0.2 | Batch Size: 300

Validation Error: 3.19%

Test Error : 3.85%

Result : Single NN

- Increasing neurons generally improved validation error up to a point (capacity helps).
- Too much dropout (0.5) often worsened performance (too much regularization).
- Best NN achieved **3.85% test error**, but required **~31 minutes of grid search**, so tuning cost is significant.
- Compared to your best tree ensembles, this simple 1-hidden-layer network is competitive but not the top performer.

Multilayer Neural Network

- Learning rate: 0.001 (Adam)
- Epoch: 5000
- Searched for :

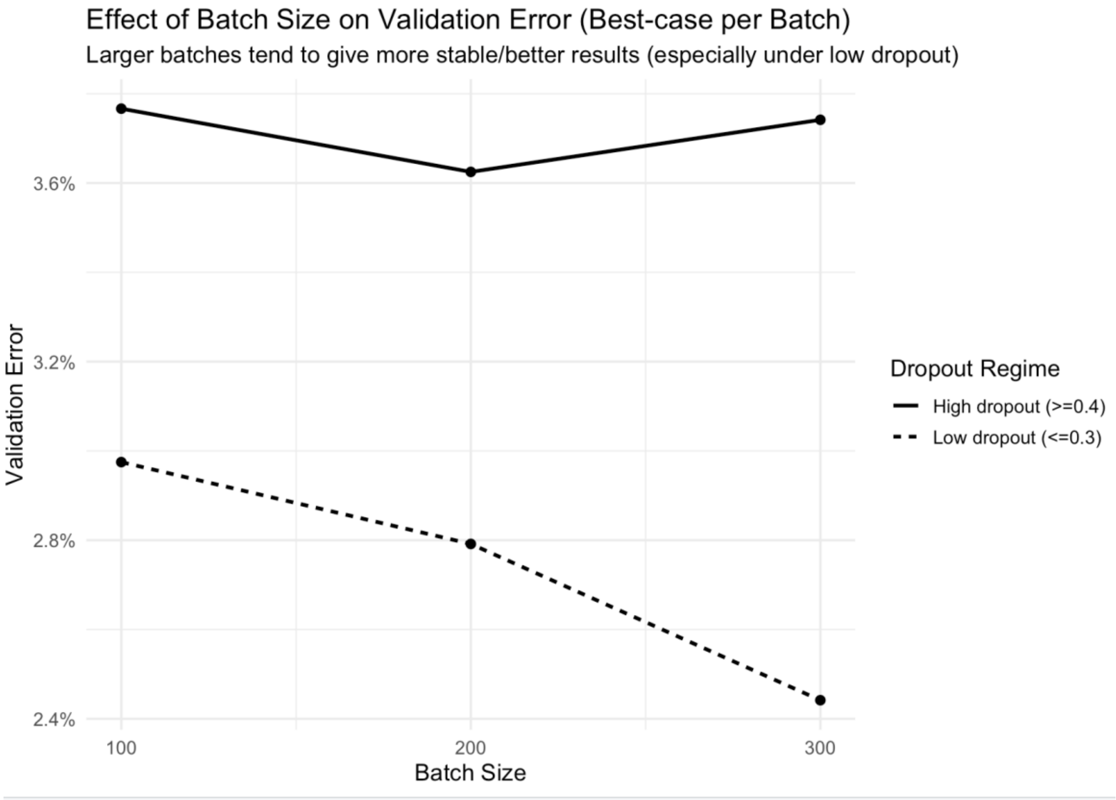
- L1 (1st layer): 50, 100, 150, 200
- L2 (2nd layer): 50, 100, 150, 200
- Drop1: 0.2, 0.3, 0.4, 0.5
- Drop2: 0.2, 0.3, 0.4, 0.5
- Batch size: 100, 200, 300

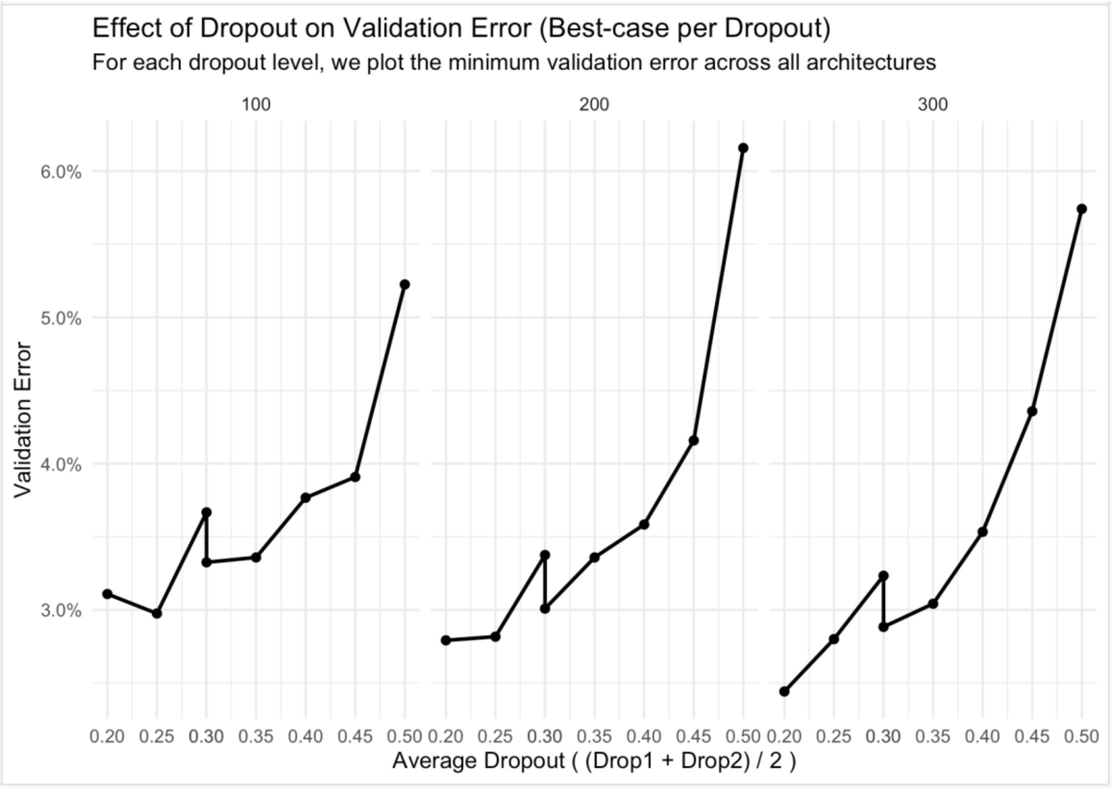
- **Best Parameters:**
- **L1: 150 | L2: 50 | Drop1: 0.2 | Drop2: 0.2 | Batch Size: 300**

- **Validation Error: 2.44%**
- **Test Error: 3.39%**

Multilayer Neural Network

- The second hidden layer mainly refines these features, so fewer units are sufficient.
- Low dropout provides mild regularization without removing too much information.
- Higher dropout values led to underfitting and worse validation performance.
- Larger batches produce more stable gradient updates when using Adam with a small learning rate.


Effect of Batch Size on Validation Error (Best-case per Batch)
Larger batches tend to give more stable/better results (especially under low dropout)
Validation Error
3.6%
3.2%
2.8%
2.4%
100
200
300
Batch Size
Dropout Regime
High dropout (>=0.4)
Low dropout (<=0.3)


Effect of Dropout on Validation Error (Best-case per Dropout)
For each dropout level, we plot the minimum validation error across all architectures
100
200
300
Validation Error
6.0%
5.0%
4.0%
3.0%
0.20 0.25 0.30 0.35 0.40 0.45 0.50
Average Dropout ((Drop1 + Drop2) / 2)

Comparison with Single NN

- **Hidden unit allocation differs**
- Single-layer NN requires many units (150) in one layer because it performs feature extraction and classification together.
- Multi-layer NN distributes capacity across layers, reducing bias more efficiently.
- **Regularization strength is unchanged**
- Both models selected dropout = 0.2, showing that stronger regularization is unnecessary for MNIST.
- Over-regularization leads to underfitting in both cases.
- **Batch size remains the same**
- Batch size = 300 is optimal for training stability in both architectures.
- Optimization behavior does not change with depth.
- **Performance improvement source**
- The multi-layer NN improves accuracy by learning hierarchical representations.
- Gains are moderate, reflecting diminishing returns on this well-structured dataset.

References

- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An introduction to statistical learning: With applications in R* (2nd ed.). Springer.
- LeCun, Y., Cortes, C., & Burges, C. J. C. (1998). **The MNIST database of handwritten digits**. http://yann.lecun.com/exdb/mnist/
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, É. (2011). *Scikit-learn: Machine learning in Python*. *Journal of Machine Learning Research, 12*, 2825–2830.
- Chen, T., & Guestrin, C. (2016). *XGBoost: A scalable tree boosting system*. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785–794). ACM.
- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2016). *TensorFlow: Large-scale machine learning on heterogeneous systems*. https://www.tensorflow.org/
- Chollet, F., et al. (2015). *Keras*. https://keras.io