LEARNING FROM DATA ASSIGNMENT

Naïve Bayes Classification and Neural Network Analysis

Word Count: 1492

gxnj57 (001110537)

Introduction:

Naive Bayes models are fast and simple classification methods that work well with high-dimensional datasets. Due to their rapid speed and limited customizable settings, they are ideal for classification tasks. (Python Data Science Handbook *by Jake VanderPlas).*

The Naïve Bayes Classifier is a widely used probabilistic machine learning model for classification. Based on Bayes' Theorem, it manages large datasets easily and effectively (Rish, 2001).

Bayes' Theorem:

The classifier is built upon the fundamental principles of Bayes' Theorem. Bayes' theorem, a key idea in probability theory, assesses the likelihood of an event based on past knowledge. The theorem is theoretically formulated as (McCallum et al., 1998):

$$P(L \mid features) = \frac{P(features \mid L) * P(L)}{P(features)}$$

- The posterior probability P(L | features) indicates the likelihood of L based on observable characteristics. It indicates the characteristics-based likelihood of label L (the class).
- The term 'P(features | L)' represents the likelihood of detecting the characteristics given the class label L.
- The prior probability of class L is denoted as P(L).
- The marginal probability of observing the set of features is denoted by P(features).

$$\frac{P(L1|features)}{P(L2|features)} = \frac{P(features|L1) * P(L1)}{P(features|L2) * P(L2)}$$

- P(L1|features): Probability of L1 being the correct label given the feature values.
- P(L2|features): Probability of L2 being the correct label given the feature values.
- Values on the RHS of this equation can be estimated using our training data.

If this is >1, then P(L1|features) > P(L2|features) and we can make our classification choice as L1 and if this is <1, then P(L1|features) < P(L2|features) and our classification choice would be L2.

As we can see, we have no direct access to the probabilities on the right. Hence, we make assumptions about the problem to allow us to estimate the probabilities as we have data only about P(features|L1) for feature values that were actually in our training data that we know are classified as L1.

Assumptions:

- Assume that features are independent variables.
- Assume that the feature data follows some probability distribution.

The classifier's 'naïve' characteristic relies on feature independence, simplifying computation by assuming that feature presence or absence is unrelated to other features (Zhang, 2004). In the context of a target variable Y and dependent features X1, X2, ... Xn, the classifier computes the posterior probability for each class and then predicts the class with the highest probability.

Variants of Naïve Bayes Classifier:

Various variations of naive Bayes classifiers are based on distinct naive assumptions regarding the data.

- The Gaussian Naïve Bayes method assumes normally distributed features for classification. This is ideal for continuous data.
- The Multinomial Naïve Bayes method is commonly used for discrete data analysis, including text categorization (Manning et al., 2008).
- Bernoulli Naïve Bayes is designed for binary or Boolean data.

Dataset Visualization: The dataset provided to us (Group 14) comprised eight numeric features, two categorical features, and one target. Figure 1 shows that the target class, widgets 1–7, was distributed consistently across the dataset, indicating no bias. Only 'length_cm' had missing values.

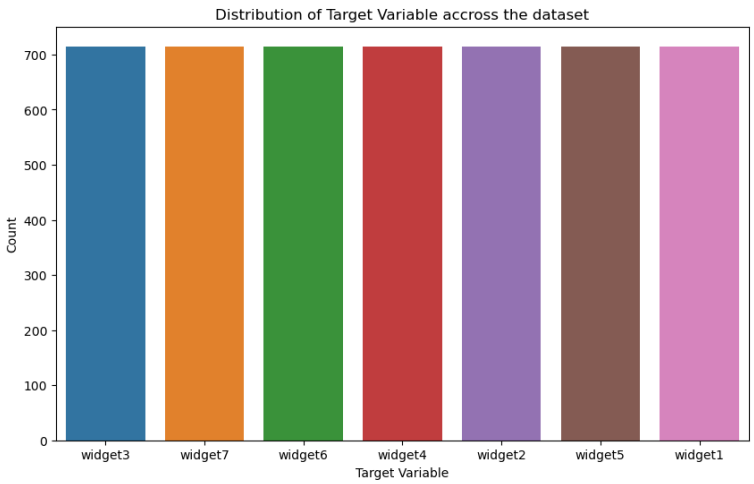

Figure 1: Distribution of Target Variables across the dataset

Dataset Cleaning: The simple imputer technique was used to fill the missing values feature by substituting null/empty values with the column mean. Next, the two categorical features were converted into numerical by using one-hot encoding. After

cleaning, the dataset was split into 24 numeric features (‘X’) and target variable (‘Y’). X and Y correspond to the datapoints and labels respectively.

As seen in Figure 2, normal distribution bell-shaped curves are present in every feature.

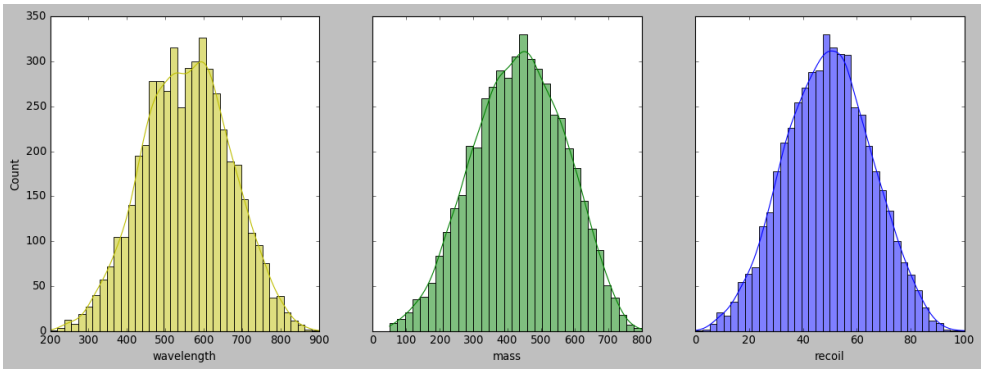

Figure 2: Distribution of features across the dataset

The correlation coefficients between dataset features are shown in Figure 3. Correlation analysis helps select features by showing their relationships.

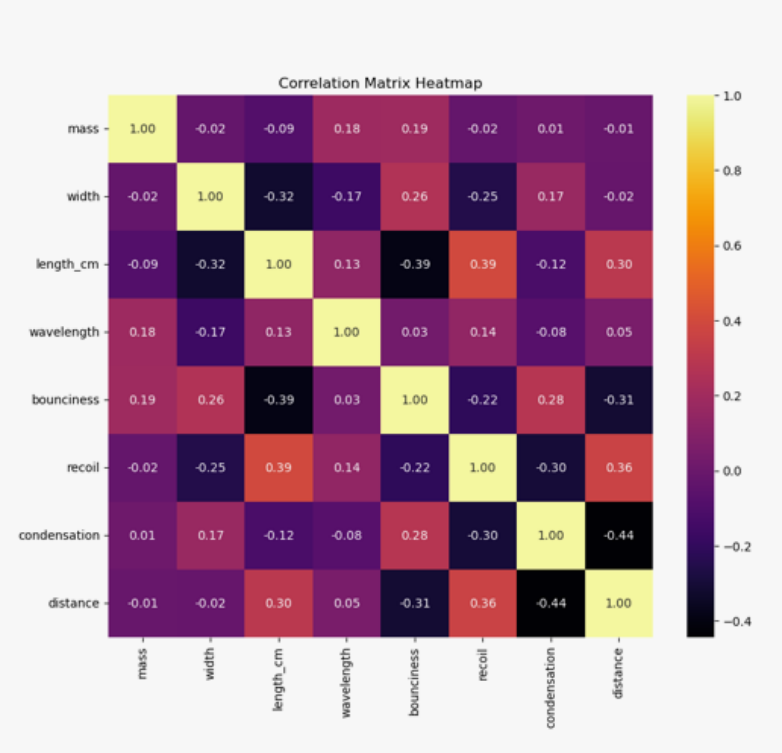

Figure 3: Correlation Matrix for all features

Guassian Naive Bayes Classifier:

Machine learning model evaluation depends on accuracy and training duration. Two GNB classifier models were developed and assessed for accuracy and training time

using 20% and 30% test sizes. The loop was run 10 times per test size to generate the model's average accuracy and training time.

Test size: 0.3

Training data accuracy: 86.40%
Test data accuracy: 86.81%
Average training time: 0.0125s

Test size 0.2

Training data accuracy: 86.34%
Test data accuracy: 86.60%
Average training time: 0.0141s

Multinomial Naive Bayes Classifier:

For the MNB classifier, 1 model was trained and tested:

Test size 0.3

Training data accuracy: 72.46%
Test data accuracy: 73.60%
Average training time: 0.0364s

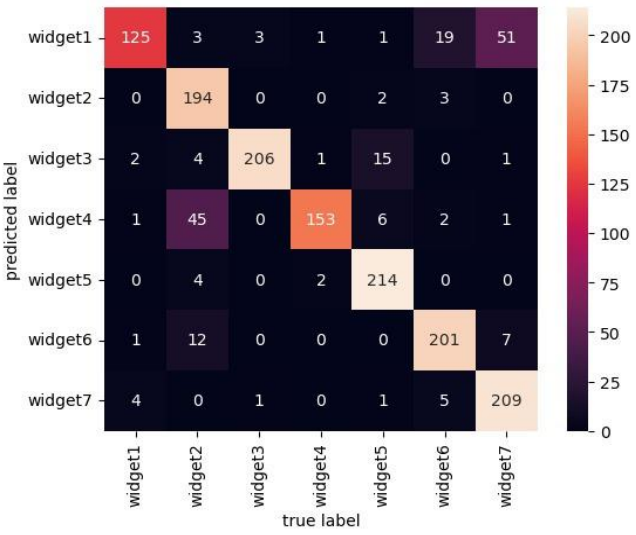

Figure 4: GNB Correlation Matrix

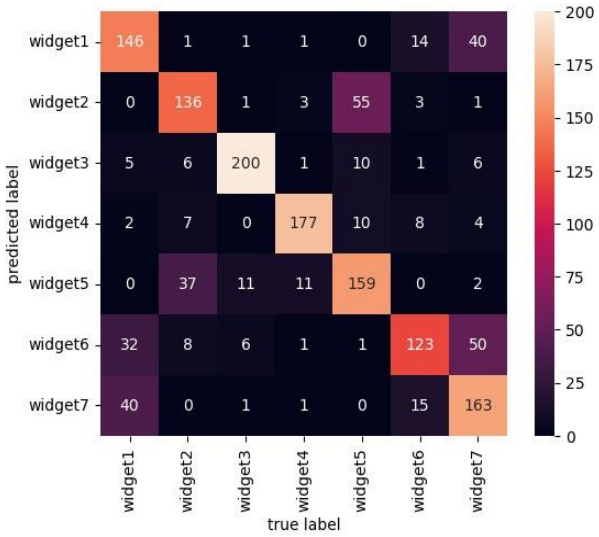

Figure 5: MNB Correlation Matrix

Figure 4 shows GNB accurately predicted 1302/1500 samples, whereas Figure 5 indicates MNB correctly predicted 1104/1500. There is no significant difference for the training time between the two. Figure 6 compares GNB and MNB model

accuracy and training times. GNB outperforms MNB on both training and testing data.

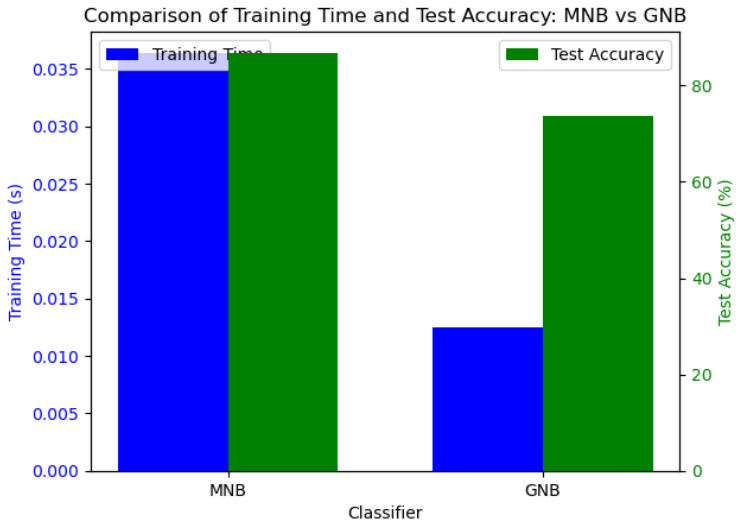

Figure 6: MNB vs GNB model

Neural Network Classifier Development:

A feedforward neural network is designed for this classification task having multiple classes (widget1-widget7). In its construction, sequential models are utilized. In order to compile this report, I have manually tuned the hyperparameters of fifteen distinct models in an effort to identify the one with the highest accuracy with respect to the specified hyperparameters. Two configurations of hidden layers (2 and 3) were selected to explore the effect of model depth on performance. The network architecture consists of an input layer that matches the dimensionality of the feature space (24 features).

Hidden layers (2 & 3) with ‘relu’ activation functions. Simpler mathematical approaches make Relu more computationally efficient than sigmoid or tanh. Regularization was achieved by incorporating dropout layers to prevent overfitting. An output layer employing a 'softmax' activation function generates a probability distribution across all seven classes, making it well-suited for multi-class classification problems. The optimizer ‘Adam’ is known for its efficiency, effectiveness, and adaptive learning, which makes it suitable for processing any data.

Hyperparamer Tuning:

- The hidden layer activation function is ‘relu’ and the output layer is ‘softmax’.
- The loss function used is ‘sparse_categorical_crossentropy’, ideal for multi-class classification tasks.

- Two hidden layers utilized (128, 64) and (64, 32) neuron combinations, whereas three hidden layers used (128, 64, 32) and (64, 32, 16).
- Dropout rate has been maintained constant throughout at 0.5. Because different neuron values were used, dropout was kept constant, allowing for a calibrated experiment.
- Adjusted learning rate (0.1, 0.01, 0.001) to assess accuracy impact.
- Used batch sizes of 20, 30, 40, and 70.
- Epochs varied between 10, 20, 25, 50, and 100 between models.

The decision was made to modify the learning rate, epochs, and batch sizes as hyperparameters. This was done to find the model's optimal learning rate, which preserves performance without overfitting to the training data. Their influence on accuracy and training time is the greatest among other hyperparameters.

The following table lists all optimization hyperparameters, model accuracy, and training times:

Model	No. of Hidden Layers	No. of neurons	Learning Rate	Epoch	Batch Size	Test Accuracy	Training Time
Model1	2	128, 64	0.1	10	40	15.27%	6.083s
Model2		128, 64	0.01	10	40	92.60%	6.39s
Model3		128, 64	0.001	10	40	92.93%	5.579s
Model4		64, 32	0.01	100	40	91.87%	36.399s
Model5		64, 32	0.01	25	40	92.27%	13.054s
Model6		64, 32	0.01	10	40	92.73%	6.702s
Model7		64, 32	0.01	20	70	92.86%	7.142s
Model8		64, 32	0.01	20	40	92.80%	9.961s
Model9		64, 32	0.01	20	20	91.53%	15.71s
Model10	3	128, 64, 32	0.01	10	40	91.53%	12.0534s
Model11		128, 64, 32	0.1	10	40	13.27%	11.321s
Model12		128, 64, 32	0.001	10	40	92.87%	11.477s
Model13		64, 32, 16	0.01	100	40	91.66%	55.96s
Model14		64, 32, 16	0.01	50	30	92.00%	36.288s
Model15		64, 32, 16	0.01	20	20	89.60%	21.66s

Figure 7: Table showing hyperparameters, training time, and accuracy.

Analysis and Comparison of Results:

Neural Network Model: Figure 7 shows that model 3 has the highest accuracy, 92.93%, and 5.579s training time. Models 6, 7, 8, and 12 also have high accuracies. The high learning rate of 0.1 causes Models 1 and 11 to have the lowest accuracies. The top 5 models' training loss vs. epochs is shown in Figure 8.

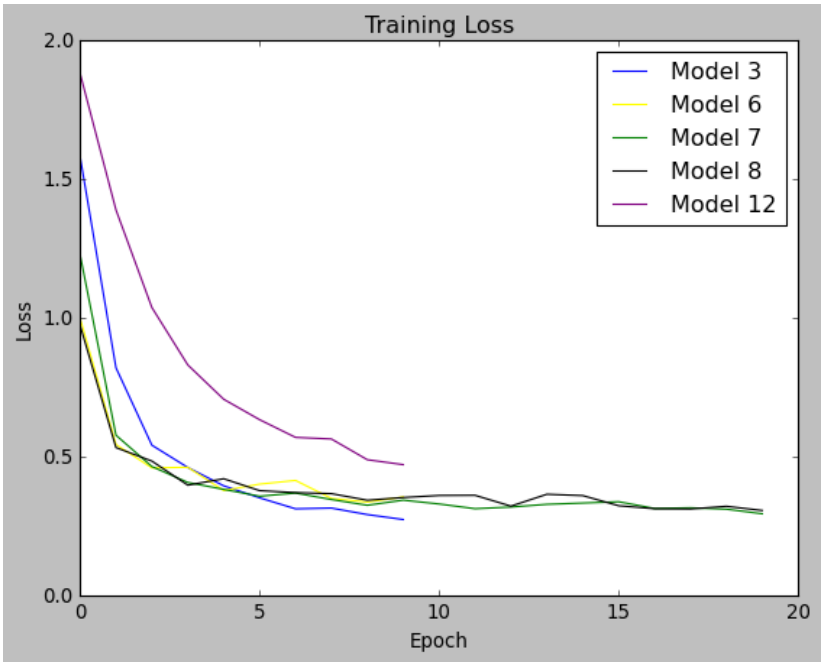

Figure 8: Training loss vs Epochs

Figure 9 compares the accuracies for these best performing models against the number of epochs. Higher accuracies have been obtained while using batch size of 10 and learning rate of 0.01. More accurate models were developed with epochs 10 or 20 than with 50 or 100. It can also be noted that the training time increases significantly when the number of epochs is increased.

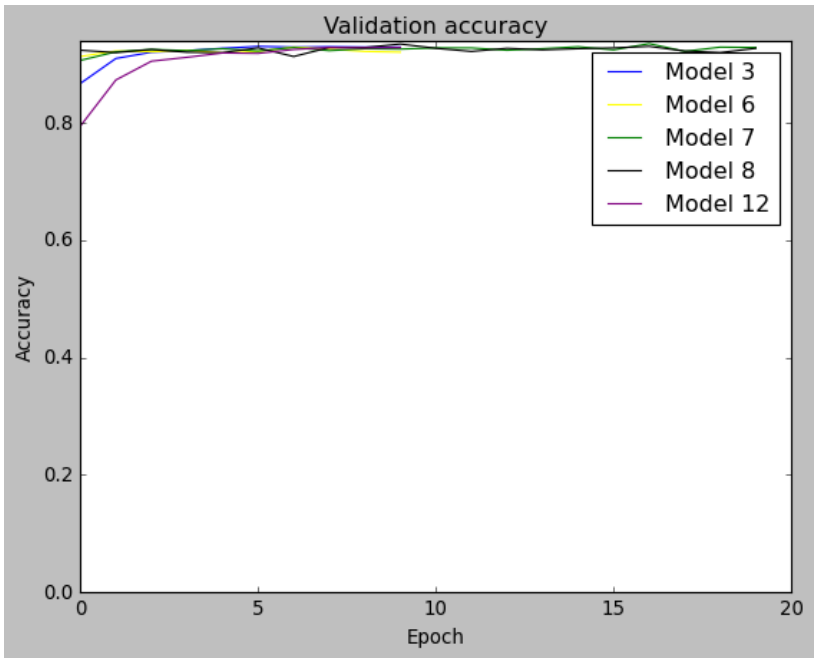

Figure 9: Validation Accuracy vs Epochs

Naive Bayes vs Neural Network:

The figures below compare the training durations and accuracies of three models: GNB, MNB, and Neural Network Model (model 3). Neural network model has the best accuracy of the 3 models, whereas MNB model has the lowest. Neural networks possess the ability to capture complex, non-linear relationships in data. This is achieved by employing non-linear activation functions and multiple layers of neurons, which enable the representation of complex patterns that simpler models might fail to capture. In practical data, Naive Bayes' feature independence assumption is often false. Neural networks can acquire interdependent representations without this limitation.

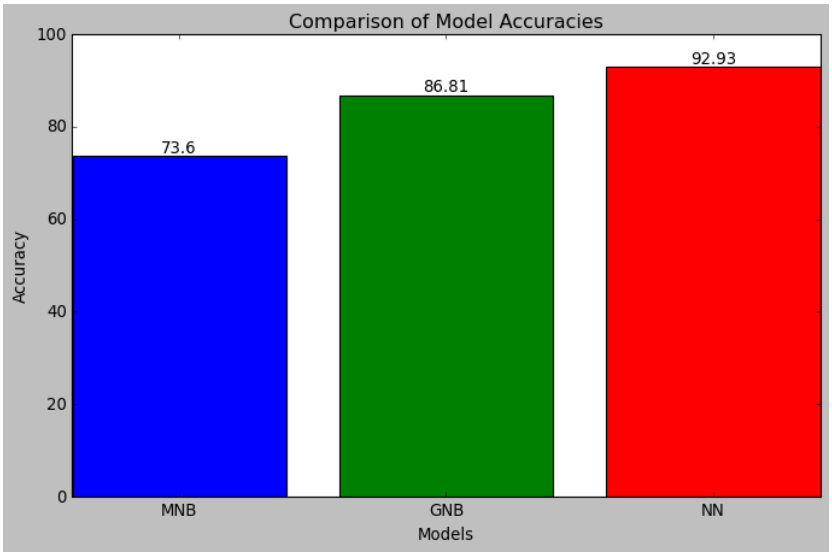

Figure 10: Test data accuracy across models

Figure 11 below shows the training times across the 3 models. Neural Network models take longer to train than NB classifiers because they have a more complicated structure and require more iterations (epochs) to converge. Complexity and training time increase with numerous hidden layers and hyperparameters.

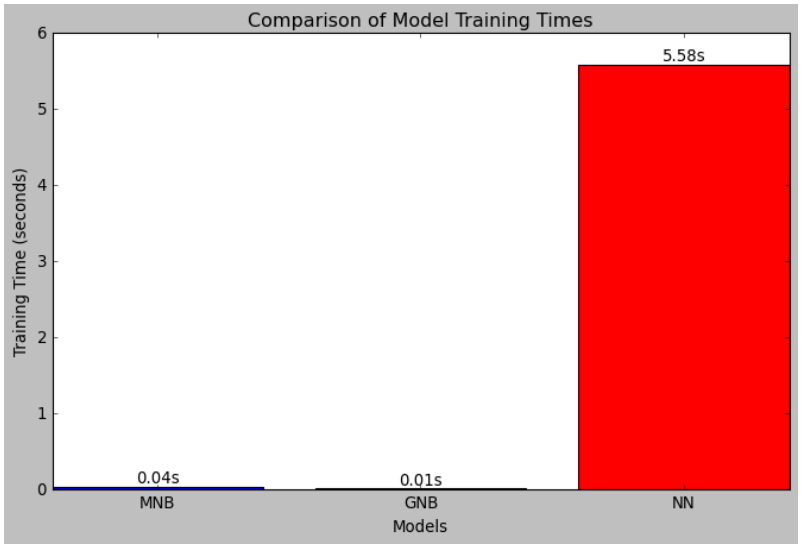

Figure 11: Training time across models

Conclusion:

In conclusion, crucial knowledge regarding the performance metrics and operational mechanisms of the Guassian Naive Bayes (GNB), Multinomial Naive Bayes (MNB), and neural network (NN) models has been amassed. Based on probabilistic theory, the NB classifier can train rapidly and efficiently, making it a good choice for fast-paced and independent data feature situations (Rish, 2001). NN models are more accurate due to strategic network construction and hyperparameter optimization, thereby achieving the highest level of accuracy among the three models.

References:

1] Rish, I. (2001). An empirical study of the naive Bayes classifier. In IJCAI 2001 workshop on empirical methods in artificial intelligence.

2] McCallum, A., Nigam, K., et al. (1998). A comparison of event models for naive Bayes text classification. AAAI-98 workshop on learning for text categorization.

3] Zhang, H. (2004). The optimality of naive Bayes. FLAIRS Conference.

4] Manning, C., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press.

5] Hand, D. J., & Yu, K. (2001). Idiot's Bayes—Not so stupid after all? International statistical review, 69(3), 385-398.

6] Lewis, D. D. (1998). Naive (Bayes) at forty: The independence assumption in information retrieval. In ECML '98 Proceedings of the Tenth European Conference on Machine Learning.

7] Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning. Springer.

8] Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.

9] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.