Training models

Machine Learning report

Nguyen Quang Duy

Faculty of Technology, University of Transportation

April 29, 2024

Abstract

Text classification is a fundamental task in natural language processing that involves assigning predefined categories or labels to text documents.

List of Figures

List of Tables

Contents

Chapter 1

Outlier detection using Isolation Forest

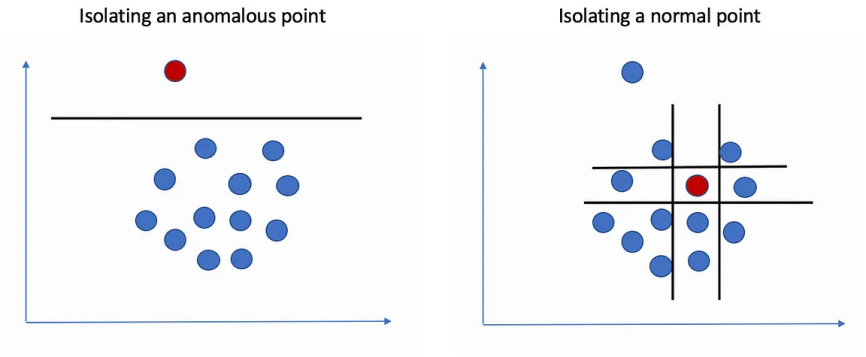

Figure 1.1: Splits to determine outlier

Isolation Forest is an unsupervised algorithm that can effectively identify anomalies or outliers in a dataset. These outliers often represent noisy or unrepresentative instances of the majority class, and their inclusion in the training data can have adverse effects on the performance of a classification model.

The Isolation Forest "isolates" observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature. Since recursive partitioning can be represented by a tree structure, the number of splittings required to isolate a sample is equivalent to the path length from the root node to the terminating node. This path length, averaged over a forest of such random trees, is a measure of normality and our decision function. Random partitioning produces noticeably shorter paths for anomalies. Hence, when a forest of random trees collectively produce shorter path lengths for particular samples, they are highly likely to be anomalies [1].

The advantages of using Isolation Forest for outlier detection include its capability to handle high-dimensional and large datasets effectively, as well as its insensitivity to the shape and distribution of normal instances.

Chapter 2

Classification using Softmax Regression

2.1. Softmax Classifier

When given an instance x, the Softmax Regression model first computes a score $s_k(x)$ for each class k, then estimates the probability of each class by applying the softmax function (also called the normalized exponential) to the scores [2]:

$$s_k(x) = x^T \theta^k$$

Each class has its own dedicated parameter vector θ^k. All these vectors are typically stored as rows in a parameter matrix Θ. Once the score of every class for the instance x is calculated, we can estimate the probability $\hat{p}_k$ that the instance belongs to class k by running the scores through the softmax function [2]:

$$\hat{p}_k = \sigma_k(s(x)) = \frac{exp(s_k(x))}{\sum_{j=1}^{K} exp(s_j(x))}$$

In the above equation:

- K is the number of classes.
- $s(x)$ is a vector containing the scores of each class for the instance x.
- $\sigma_k(s(x))$ is the estimated probability that the instance x belongs to class k, given the scores of each class for that instance.

Just like the Logistic Regression, the Softmax Regression predicts the class with the highest estimated probability (which is simply the class with the highest score), as shown in the equation below [2]:

$$\hat{y} = \arg\max_k \ \sigma_k(s(x)) = \arg\max_k \ s_k(x)$$

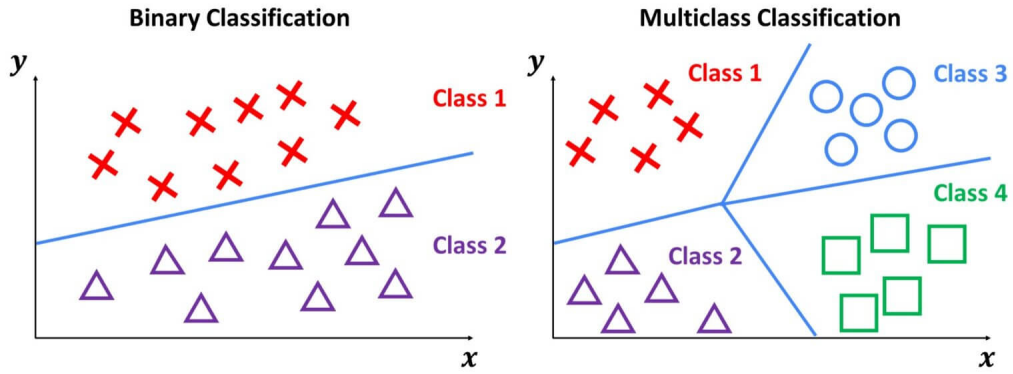

Figure 2.1: Logistic Regression vs Softmax Regression

If the dataset is two dimensional, Softmax Regression partitions the plane into multiple polygonal regions as depicted in figure 2.1.

2.2. Loss function in classification

When dealing with percentages, the function $f(x) = -log(x)$ is commonly used to quantify the proximity of the current percentage to 1. The objective of a classification task is to have a model that estimates a high probability for the target class (and consequently a low probability for the other classes). Minimizing the cost function below, called the cross entropy, should lead to this objective because it penalizes the model when it estimates a low probability for a target class. Cross entropy is frequently used to measure how well a set of estimated class probabilities matches the target classes.

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(\hat{p}_k^{(i)})$$

In the above equation, $y_k^{(i)}$ is the target probability that the ith instance belongs to class k. In general, it is either equal to 1 or 0, depending on whether the instance belongs to the class or not. When there are just two classes $(K = 2)$, this cost function is equivalent to the Logistic Regression's cost function.

In practice, it is common to apply regularization techniques to the loss function in order to penalize large values of θ. This is necessary to prevent the model from becoming overly sensitive to new data. We use a technique known as the Elastic Net, the Elastic Net attempts to minimize:

$$L(\Theta) = J(\Theta) + \lambda_1 \sum_{i=1}^{K} \|\theta^{(i)}\| + \lambda_2 \sum_{i=1}^{K} \|\theta^{(i)}\|^2$$

Since $J(\Theta)$ is a convex function, $L(\Theta)$ also a convex function, and therefore has a unique minimum which make Gradient Descent (or any other optimization algorithms) guarenteed to approach the global minimum.

When dealing with a highly imbalanced class distribution, it is common to assign weights to each instance that are inversely proportional to the size of its class. This approach helps to penalize the model more severely when it misclassifies instances from the minority class.

Chapter 3

Classification using Support Vector Machines (SVM)

3.1. Support Vector Classifier (SVC)

The linear SVM classifier model predicts the class of a new instance x by simply computing the decision function $w^T x + b = w_1 x_1 + ... + w_n x_n + b$. If the result is positive, the predicted class $\hat{y}$ is the positive class, and otherwise it is the negative class.

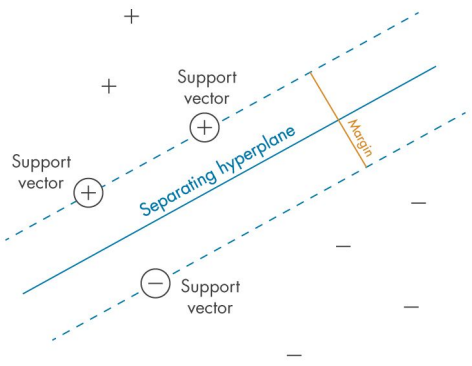

Figure 3.1: Linear SVC

In the figure above, the dashed lines represent the points where the decision function is equal to 1 or –1: they are parallel and at equal distance to the decision boundary, and they form a margin around it. Training a linear SVM classifier means finding the values of w and b that make this margin as wide as possible while avoiding margin violations (hard margin) or limiting them (soft margin). To achieve a larger margin, we aim to minimize the norm of the weight vector $w^T w$. This is because the size of the margin is directly influenced by the slope of the decision function.

To get the soft margin objective, we need to introduce a slack variable $\zeta(i) \geq 0$ for each instance, $\zeta(i)$ measures how much the ith instance is allowed to violate the margin. If we define $t^{(i)} = -1$ for negative instances and $t^{(i)} = 1$ for positive instances, we can create a constraint $t^{(i)}(w^T x^{(i)} + b) \geq 1 - \zeta(i)$. This constraint implies that smaller values of $\zeta(i)$ correspond to fewer margin violations and $\zeta(i) = 0$ indicates that there must be no margin violations (hard margin). In addition, it is necessary to introduce a hyperparameter, denoted as C to balance between minimize $\zeta(i)$ and minimize $w^T w$. This gives us the constrained optimization problem:

$$\underset{w,b,\zeta}{\text{minimize}} \ \frac{1}{2} w^T w + C \sum_{i=1}^{m} \zeta(i)$$

subject to $t^{(i)}(w^T x^{(i)} + b) \geq 1 - \zeta(i)$ for $i = 1, 2, ..., m$

3.2. Non-linear SVM with Gaussian RBF kernel

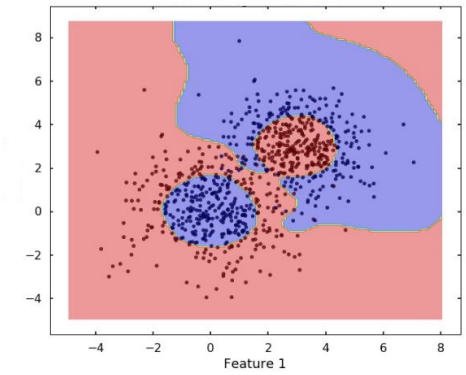

Figure 3.2: Gaussian RBF kernel in a 2D classification task

The Gaussian RBF (Radial Basis Function) kernel is a popular choice for kernel methods that are used for classification and regression tasks. It computes the similarity or distance between data points in a transformed feature space. In the case of the Gaussian RBF kernel, the similarity between two data points, denoted as x and y, is computed as $exp(-\gamma(x-y)^2)$. Here $(x-y)^2$ represents squared Euclidean distance between the two data points, and gamma is a hyperparameter that controls the spread of the kernel function. The gamma parameter determines the influence of each training example on the decision boundary. Higher values of gamma result in a narrower Gaussian curve and can lead to overfitting, while lower values of gamma result in a wider Gaussian curve and can lead to underfitting.

In practice, the Gaussian RBF kernel allows SVMs to create non-linear decision boundaries by implicitly mapping the original data into a higher-dimensional feature space. This kernel function is commonly used when the data is not easily separable in the original feature space and when there is no prior knowledge of the underlying data distribution.

3.3. SVM in multilabel classification

To perform multilabel classification using Support Vector Machines (SVM), we adopt the one-vs-one (OvO) scheme. This scheme involves creating multiple SVM models, each trained to distinguish between a pair of classes. For example, if there are n classes then the number of models required for the OvO scheme is $n(n-1)/2$. The OvO scheme also benefits SVM with Gaussian RBF kernel, since this kernel has poor scalability with respect to the dataset size.

While each binary classifier predicts a single class label, the OvO strategy determines the final prediction based on the class with the highest number of votes. If the binary classifiers provide class membership probabilities, the class with the highest sum score across the models is selected as the predicted class label.

Chapter 4

Multilayer Perceptron neural network

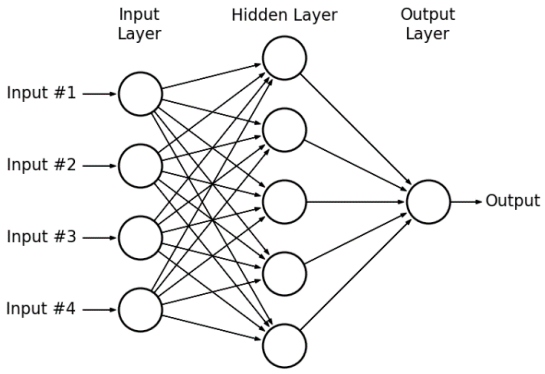

Figure 4.1: Multilayer Perceptron neural network

The Multilayer Perceptron (MLP) is regarded as a the most basic type of neural network, as well as the earliest architecture of its kind. The idea behind the MLP is to create a network of interconnected artificial neurons, organized into multiple layers. The network consists of an input layer, one or more hidden layers, and an output layer. Each layer is fully connected to the previous one. Each neuron in the network receives inputs, applies an activation function to the weighted sum of its inputs, and passes the result as output to the neurons in the subsequent layer. The activation function can either be $tanh$, $sigmoid$ or $RELU$ (rectified linear unit) based on the task at hand.

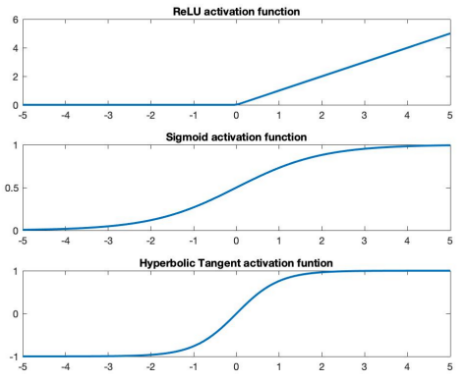

Figure 4.2: Activation functions

MLP can be considered as a composition of multiple stacked softmax regression models.

Chapter 5

Bidirectiinal long-short term memory neural network with attention mechanism

5.1. Long-short term memory neural network

Long-short term memory (LSTM) is a special kind of Recurrent Neural Network, capable of learning long term dependencies. LSTMs also have chain like structure, but the repeating module has a different structure.

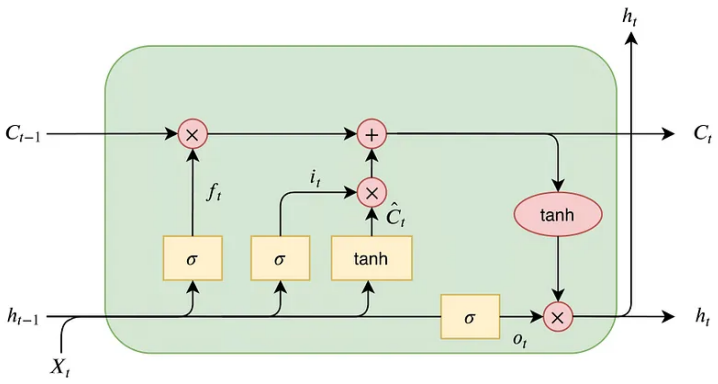

Figure 5.1: Long-Short Term Memory cell

In figure 5.1, operation $+$ and $\times$ are performed element wise on rows of one-dimensional matrix, x_t can be multi-dimensional.

The LSTM architecture is based on the following key components with $tanh$ and $sigmoid$ as activation functions [3]:

- Cell State (c_t): This represents the memory of the LSTM and can store information over long sequences. It can be updated, cleared, or read from at each time step.
- Hidden State (h_t): The hidden state serves as an intermediary between the cell state and the external world. It can selectively remember or forget information from the cell state and produce the output.
- Input Gate (i_t): The input gate controls the flow of information into the cell state. It can learn to accept or reject incoming data.
- Forget Gate (f_t): The forget gate determines what information from the previous cell state should be retained and what should be discarded. It allows the LSTM to "forget" irrelevant information.

- Output Gate (o_t): The output gate controls the information that is used to produce the output at each time step. It decides what part of the cell state should be revealed to the external world.

The typical flow of data is as follows, with W and U are parameter matrixes to be learned in model training:

$$\begin{aligned} f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\ i_t &= \sigma(W_i x_i + U_i h_{t-1} b_i) \\ o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\ \hat{c}_t &= \tanh(W_c x_t + U_c h_{t-1} + b_c) \\ c_t &= f_t \cdot c_{t-1} + i_t \cdot \hat{c}_t \\ h_t &= o_t \cdot c_t \end{aligned}$$

5.2. Bidirection LSTM neural network

Bidirectional LSTM (BiLSTM) is a variant of the traditional LSTM model that incorporates information from both past and future contexts. BiLSTM models often demonstrate faster learning capabilities, while also achieving higher accuracy levels.

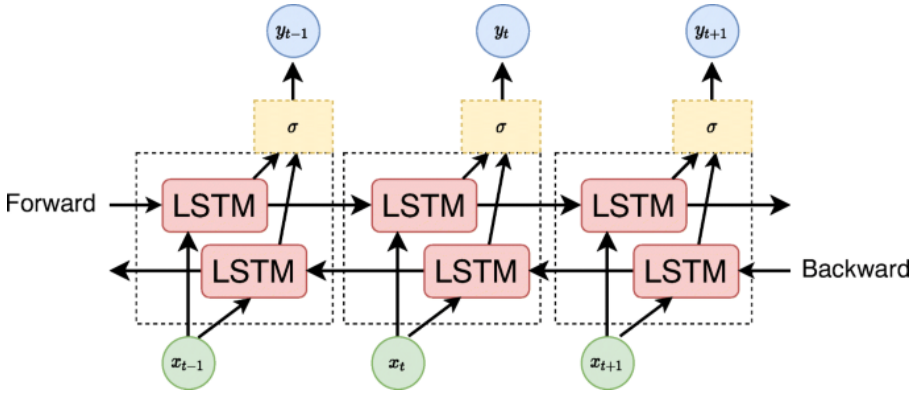

Figure 5.2: Bidirection Long-Short Term Memory neural network

It processing the input sequence in two directions: forward and backward. The input sequence is split into two parts. One part is processed in the forward direction, starting from the beginning of the sequence, while the other part is processed in the backward direction, starting from the end of the sequence. During the forward pass, the forward LSTM layer takes the input sequence and processes it step by step, updating its hidden state and cell state. The final hidden state of the forward LSTM captures the information from the past context. Simultaneously, during the backward pass, the backward LSTM layer processes the input sequence in reverse order, updating its hidden state and cell state. The final hidden state of the backward LSTM captures the information from the future context. The outputs of both the forward and backward LSTMs are then combined, usually by concatenating or summing them element-wise, to create the final output representation of each time step.

5.3. Attention mechanism

LSTM models tend to underperform when applied to datasets of medium to small size. To address this limitation, an attention layer can be introduced, which also make the model converge faster. In our specific problem, the attention weights are computed using the outputs from LSTM cells of the LSTM model, and subsequently, these weights are added to the LSTM model outputs.

In the upcoming sections, we will refer to this model as BiLSTM-ATT.

Chapter 6

Training and evaluating models

The UTC opinion dataset is divided into training set, validation set, and test set, accounting for 70%, 15%, and 15% of the data respectively while keeping the distribution of classes. Classes are weighted during training, where the weight is inversely proportional to the size of its class.

6.1. Outlier detection

During the training process of the SVM and Softmax Regression models, we apply the Isolation Forest algorithm from the Sklearn package to remove outliers in the training set. This algorithm automatically identifies and removes outliers, resulting in the removal of approximately 100 samples from the training set.

6.2. Training and evaluating Softmax Regression

To implement Softmax Regression, we will utilize the SGDClassifier class from the Sklearn package. The SGDClassifier is capable of training the model multiple epochs using the same set of batches. After several experiments, we decided that to let the algorithm run for 180 epochs with batches of 1024 samples. Additionally, we set the l1_ratio to 0.15 and the alpha (regularization strength) to 0.0002.

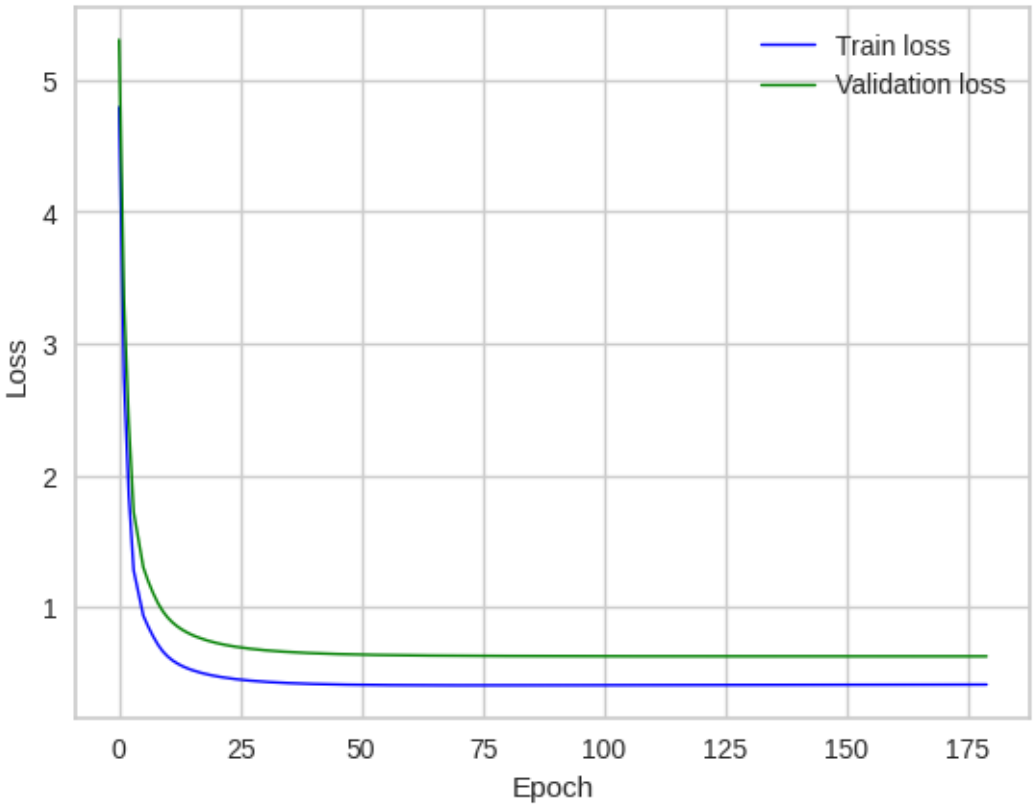

Figure 6.1: Softmax Regression training process

Class	Recall	Precision	F1
1	0.674	0.681	0.677
2	0.796	0.807	0.801
3	0.880	0.859	0.869
4	0.850	0.805	0.827
5	0.747	0.724	0.736
6	0.660	0.759	0.706
7	0.852	0.845	0.848
8	0.886	0.886	0.886
9	0.932	0.950	0.941
10	0.962	0.809	0.879
11	0.878	0.869	0.873

Table 6.1: Classification result of Softmax Regression on the validation set

Class	Recall	Precision	F1
1	0.775	0.723	0.748
2	0.782	0.785	0.784
3	0.767	0.802	0.784
4	0.810	0.832	0.821
5	0.783	0.735	0.758
6	0.623	0.707	0.662
7	0.863	0.869	0.866
8	0.873	0.825	0.848
9	0.867	0.901	0.883
10	0.897	0.926	0.911
11	0.862	0.818	0.839

Table 6.2: Classification result of Softmax Regression on the test set

Set	Accuracy	Weighted F1
Validation	0.811	0.832
Test	0.802	0.815

Table 6.3: Classification result summary for Softmax Regression

6.3. Training and evaluating SVM

When applying SVM, we employ grid search to identify the smallest value of C that maximizes both the F1 score and accuracy on the validation set. Up to three models with the highest scores on the validation set is saved, and then the model that performs best on the test set is selected. We use the SVC class from the sklearn package and keep most of its default values (SVC default kernel is Gaussian RBF and the gamma parameter is set to "auto"), except for the hyperparameter C. The sklearn SVC class is capable of handling multilabel classification automatically. Following training and evaluation, we determine that the optimal value for the hyperparameter C for our classification is around 2.5.

Class	Recall	Precision	F1
1	0.643	0.775	0.703
2	0.835	0.761	0.796
3	0.912	0.859	0.885
4	0.885	0.814	0.848
5	0.798	0.765	0.781
6	0.709	0.789	0.747
7	0.860	0.860	0.860
8	0.895	0.880	0.888
9	0.940	0.931	0.935
10	0.953	0.872	0.911
11	0.870	0.879	0.874

Table 6.4: Classification result of SVM on the validation set

Class	Recall	Precision	F1
1	0.714	0.822	0.764
2	0.809	0.775	0.791
3	0.904	0.767	0.830
4	0.843	0.858	0.851
5	0.772	0.796	0.784
6	0.692	0.744	0.717
7	0.901	0.908	0.904
8	0.915	0.885	0.900
9	0.947	0.891	0.918
10	0.946	0.926	0.935
11	0.903	0.848	0.875

Table 6.5: Classification result of SVM on the test set

Set	Accuracy	Weighted F1
Validation	0.824	0.850
Test	0.832	0.850

Table 6.6: Classification result summary for SVM

The SVM model serves as a baseline for accuracy comparison with other deep learning models.

6.4. Training and evaluating MLP

The MLP training is performed using the Keras library. The Adam solver in Keras is capable of training a model on batches but it results in a loss graph that fluctuates widely around the convergence point. Each batch has a size of 256, and instead of using a fixed number of epochs for training, we rely on alternative stopping criteria: The training process is iterated until either certain accuracy thresholds are reached, or the accuracy drops too low, or the loss becomes too high. We include a safe_epoch parameter in the Keras callback "on_epoch_end". This means that stopping rules are only applied when the number of epochs exceeds or is equal to the safe_epoch parameter. Finally, we introduce a layer dropout in the model to prevent overfitting by randomly deactivating approximately 60% of the neurons in the previous layer.

Layer type	Output shape	No. parameters
Dense	(None, 128)	98432
Dropout	(None, 128)	0
Dense	(None, 11)	1419

Table 6.7: MLP model

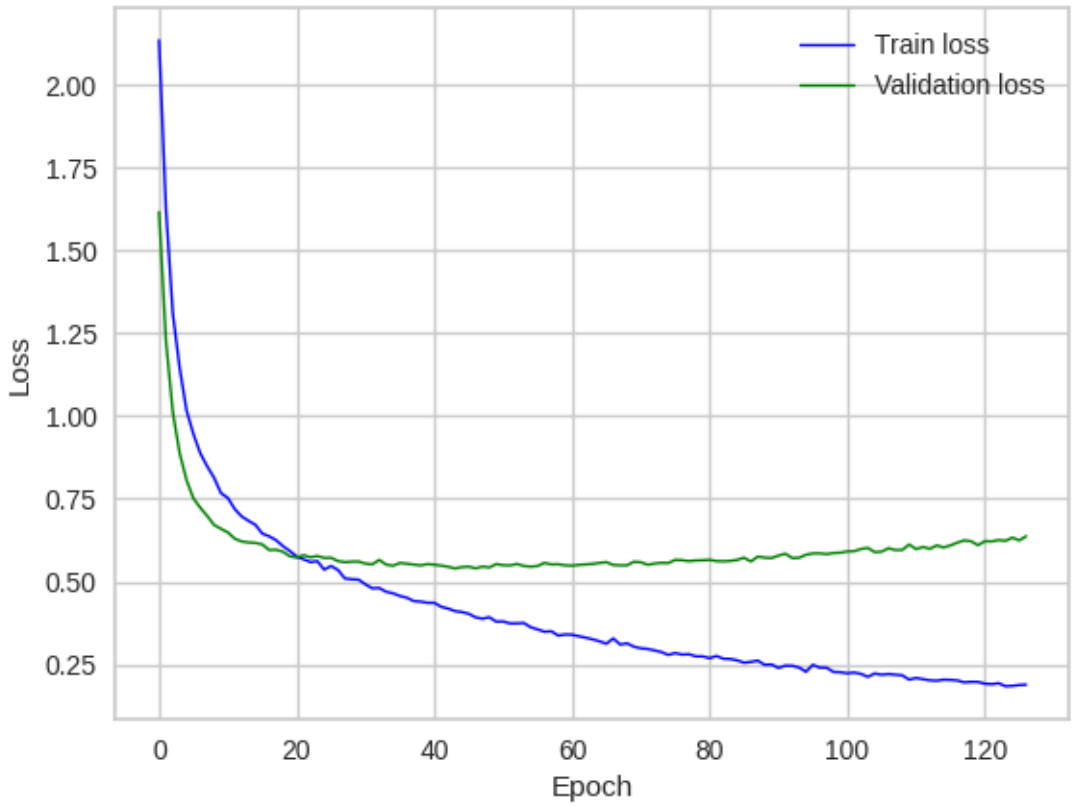

Figure 6.2: MLP training process

Class	Recall	Precision	F1
1	0.675	0.738	0.705
2	0.800	0.814	0.807
3	0.925	0.871	0.897
4	0.902	0.814	0.856
5	0.728	0.765	0.746
6	0.769	0.752	0.760
7	0.880	0.853	0.866
8	0.901	0.891	0.896
9	0.925	0.980	0.952
10	0.913	0.894	0.903
11	0.924	0.859	0.890

Table 6.8: Classification result of MLP on the validation set

Class	Recall	Precision	F1
1	0.738	0.780	0.758
2	0.825	0.813	0.819
3	0.882	0.779	0.827
4	0.846	0.876	0.861
5	0.740	0.755	0.747
6	0.754	0.737	0.745
7	0.881	0.908	0.894
8	0.902	0.852	0.876
9	0.857	0.891	0.874
10	0.880	0.936	0.907
11	0.874	0.838	0.856

Table 6.9: Classification result of MLP on the test set

Set	Accuracy	Weighted F1
Validation	0.831	0.853
Test	0.829	0.837

Table 6.10: Classification result summary for MLP

6.5. Training and evaluating BiLSTM-ATT

Each sample is reshaped to become a three dimensional object of size $(None, 8, 96)$. Then, the BiLSTM-ATT model is trained using a similar technique to the MLP model, with some remarkable layers as follows:

1. Bidirectional LSTM Layer: This layer utilizes a bidirectional LSTM network with an output of 64 units for each LSTM cell. The dropout parameter is set to 0.2, which means that during training, 20% of the LSTM units are randomly disabled.

2. Concatenate Layer: This layer concatenates the output of the LSTM layer with the attention vector.

3. Lambda Layer: This layer applies a lambda function to sum the concatenated tensor along the second-to-last axis, resulting in a fixed-size representation of the input sequence.

4. Dropout layer: Randomly deactivates 40% of the elements before the next layers.

Layer type	Output shape	No. parameters
Input	[(None, 8, 96)]	0
Bidirectional	(None, 8, 128)	82432
Dense	(None, 8, 1)	129
Flatten	(None, 8)	0
Activation	(None, 8)	0
Repeat vector	(None, 64, 8)	0
Permute	(None, 8, 64)	0
Concatenate	(None, 8, 192)	0
Lambda	(None, 192)	0
Dropout	(None, 192)	0
Dense	(None, 11)	2123

Table 6.11: BiLSTM-ATT model

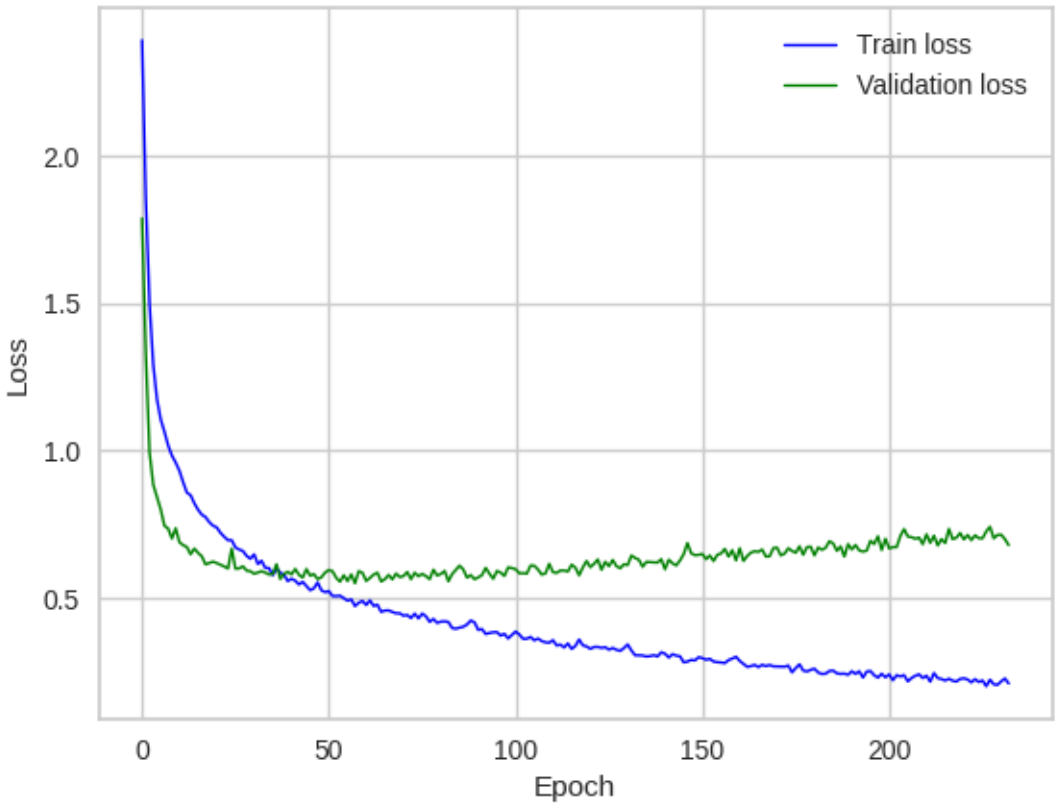

Figure 6.3: BiLSTM-ATT training process

Class	Recall	Precision	F1
1	0.665	0.738	0.700
2	0.831	0.775	0.802
3	0.869	0.859	0.864
4	0.920	0.814	0.864
5	0.818	0.735	0.774
6	0.684	0.782	0.730
7	0.803	0.884	0.841
8	0.885	0.875	0.880
9	0.948	0.901	0.924
10	0.933	0.894	0.913
11	0.880	0.889	0.884

Table 6.12: Classification result of BiLSTM-ATT on the validation set

Class	Recall	Precision	F1
1	0.763	0.775	0.769
2	0.826	0.803	0.814
3	0.843	0.814	0.828
4	0.786	0.876	0.828
5	0.824	0.714	0.765
6	0.730	0.752	0.741
7	0.815	0.915	0.862
8	0.896	0.847	0.871
9	0.935	0.861	0.897
10	0.879	0.926	0.902
11	0.860	0.869	0.864

Table 6.13: Classification result of BiLSTM-ATT on the test set

Set	Accuracy	Weighted F1
Validation	0.821	0.844
Test	0.826	0.836

Table 6.14: Classification result summary for BiLSTM-ATT

It's worth noting that although the BiLSTM-ATT model has the lowest score overall, as the dataset grows the SVC model becomes impractical due to its poor scalability and complications in incremental learning, and MLP tends to underperform when the size of the dataset becomes excessively large.

6.6. Bagging models

To further enhance our predictions, we combine the outputs of multiple models through an ensemble approach. The predicted class is determined by voting, and in the event of a tie, an order is set to select the model with the highest priority. We employ ensemble bagging with previously trained models, including SVM, MLP, and BiLSTM-ATT.

Class	Recall	Precision	F1
1	0.674	0.759	0.714
2	0.833	0.821	0.827
3	0.892	0.871	0.881
4	0.886	0.823	0.853
5	0.793	0.745	0.768
6	0.748	0.782	0.765
7	0.866	0.853	0.859
8	0.905	0.880	0.893
9	0.932	0.950	0.941
10	0.933	0.883	0.907
11	0.879	0.879	0.879

Table 6.15: Classification result on the validation set

Class	Recall	Precision	F1
1	0.751	0.822	0.785
2	0.810	0.824	0.817
3	0.893	0.779	0.832
4	0.847	0.885	0.866
5	0.815	0.765	0.789
6	0.758	0.729	0.743
7	0.881	0.908	0.894
8	0.909	0.874	0.891
9	0.891	0.891	0.891
10	0.936	0.936	0.936
11	0.885	0.859	0.872

Table 6.16: Classification result on the test set

Set	Accuracy	Weighted F1
Validation	0.834	0.853
Test	0.841	0.852

Table 6.17: Classification result summary

References

[1] scikit-learn developers, "sklearn.ensemble.isolationforest," https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html.

[2] A. G. Sager, *Hands-On Machine Learning with Python and TensorFlow*. O'Reilly Media.

[3] S. Hesaraki, "Long short-term memory (lstm)," *Medium*. [Online]. Available: https://medium.com/@saba99/long-short-term-memory-lstm-fffc5eaebfdc