Pattern recognition by combining different classifiers

Jezidžić, Marin

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO–MATEMATIČKI FAKULTET
MATEMATIČKI ODSJEK

Marin Jezidžić

PATTERN RECOGNITION BY COMBINING DIFFERENT CLASSIFIERS

Diplomski rad

Voditelj rada:
prof. dr. sc Robert Manger

Zagreb, studeni 2023.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

To my family and friends

Contents

Abstract

In the current period where the data driven methods vastly outperform standard decision making processes, combining classifiers presents one of the key ingredients for such success. My work begins with an exhaustive analysis of all the components of the ensemble system including the popular algorithms such as Boosting and Random Forests. I explore the concept of diversity, acknowledge its role in creating a successful ensemble learner and present some of the commonly used methods for its measurement. The explored techniques are used in my experiment in the task of speech recognition. Study includes Transformers, their modification E-Branchformer as reference and the ensembles containing them which are the current state of the art models in end-to-end speech recognition problem. Experiment is held on a popular speech dataset, *Librispeech*. Thesis explores the advantages and flaws of combining several models of such type. Moreover, I discuss the computational issues in real world applications of large models. Thesis ultimately establishes that combining classifiers is an empirical process, that there is no universal, best performing algorithm for every task.

Introduction

Artificial intelligence is one of the most talked-about topics these days. Surprising results are emerging from giving a simple input prompt to a language model. Nowadays, neuroscientists use their research on brain functioning to create new components for neural networks and conversely, they use the analysis of neural net behavior to better understand the human brain. Such methodology has led to many papers in recent years. Moreover, language models are now co-authors in scientific papers, a fact that is indisputable. This trend will only continue to the point where AI gains its own consciousness. At that point, it will create its own research papers based on its research interests. The complete need for AI arose from dissatisfaction with common statistical estimators such as mean, median, variance, and so on. Scientists started searching for patterns, leading to the creation of the first machine learning models.

The entire research area is lately advancing by taking advantage of the newest hardware options, highlighting the importance of computing development for the future of the field. In the last several years, there have been hundreds of papers on scaling large models and making them suitable for real-world usage by limiting the computational time required for their training and inference. Currently, simple, small algorithms or models in the AI context simply do not possess the capability to generalize well in unseen environments. For this purpose, these algorithms or models are frequently combined to get the improved performance. As previously mentioned, that comes at the price of computing. I believe that for an AI system to be complete, functional, and not useless in a real-world scenario, it must be fast and reliable.

In this thesis, my primary goal is to emphasize the role of combining classifiers in achieving SOTA results on all of today's AI-related tasks, such as speech recognition, object detection, action recognition, and so on. There will be a discussion and overview of the advantages and shortcomings of current methods of combining classifiers and their future. The thesis is structured such that in the first chapter, some extra flavor to the discussion from above is added, but I also present some already established notation. In the second chapter, thesis explore each bit of the topic more thoroughly. Some common ways to combine classifiers are introduced and afterward, through experiment, I will attempt to prove that my ensemble outperforms each sin-

gle estimator. At the end, conclusion will be presented based on the results of the experiment.

Chapter 1

Combining Pattern Classifiers

1.1 Discussion

Let's bring our topic into a real-world context. Classification in AI systems happens on various levels and has multiple applications, from feature selection to prediction and prediction combination. Much like humans, who constantly make classification decisions in everyday life, AI systems employ similar decision-making processes. Now, consider a parliament as a political arena. Parallels can be drawn between politicians and classifiers in an AI ensemble. Each politician, much like a classifier, is chosen based on their performance, similar to how classifiers are selected based on benchmark results. In a democratic system, every politician's vote has equal weight, akin to the majority voting system in classifiers. Despite its simplicity, it's highly effective, much like the saying, "None of us is as smart as all of us" – Ken Blanchard, often applies. This principle is exemplified in Random Forests, where decisions of individual Decision Tree classifiers, developed through bootstrapping are combined.

It's intriguing to note how AI behaviours mirror human actions. This leads to the question: Is there a better method then majority voting for combining classifiers? Statistically, the answer is yes. For instance, the Naive Bayes Combiner (NBC) and Behavior Knowledge System often outperform majority voting, although the "No Free Lunch Theorem" still applies. Why not experiment with these methods in a parliamentary setting if they prove more effective then regular democratic votes? For example, we could track and evaluate politicians votes based on outcomes like resolving critical issues. Over a larger dataset, each politician's vote could be weighted, potentially leading to a more effective system then the unbiased original. While these analogies extend, they underscore that AI isn't a mysterious art; it's a human creation, deeply rooted in our history and now finding applications in modern AI systems. Artificial General Intelligence (AGI) will likely evolve from these ensemble

methods, possibly employing more advanced techniques then those currently in use.

1.2 Historical Overview

After the starting discussion, begginings of this field can be explored. Combining pattern classifiers fits into a topic called ensemble learning, which is considered synonymous. The definition and boundaries of the field are, to put it mildly, ambiguous. In the book 'Combining Pattern Classifiers' by Kuncheva [13], it is stated that one should use combiners as methods to combine classifiers. One step further, one may consider methods to combine combiners, and so on, to infinity.

The idea of combining classifiers was proposed more then 70 years ago by Sebestyen in the book "Decision Making Processes in Pattern Recognition", where he emphasized the importance of machine learning in pattern recognition and acknowledged possible approaches to surpass the shortcomings of regular statistics in the decision-making process. A further major leap was made by Robert E. Schapire in 1990. in his paper "The Strength of Weak Learnability", introducing a novel method of combining classifiers named Boosting. The idea can be summarized as using weak classifiers to build a strong one. The results were amazing, so much so that the best-performing algorithms on tabular data today are based on this approach, obviously with various modifications and upgrades. After this, the next major leap was made by Leo Breiman six years later, publishing a paper called "Bagging Predictors", featuring the usage of the bootstrap method on datasets and averaging predictions made by each base classifier. The concept later expanded to Random Forests, which focuses on reducing the correlation between predictors. Neural networks can also be considered as the ensemble algorithm, each neuron can be acknowledged as a simple estimator, and their stacking as a combiner. Recall that the Perceptron algorithm is nothing more then Logistic Regression in special case. Now, if neural networks are considered to belong in this field, its relevance grows exponentially.

1.3 Parallel Computing

To address the question of 'Why' concerning the relevance of the field, an understanding of parallel and distributed computing is essential, as its development shows a strong correlation with the topic of interest. Most modern computing processes involving AI systems are managed by graphics cards, primarily due to their high number of CUDA cores. In simpler terms, these are similar to CPU cores, but with differences that are typically inconsequential for AI tasks. A common analogy compares computing a massive number of operations on a GPU to a large container ship

moving slowly but carrying a vast amount of cargo, contrasted with a fast-moving boat with limited cargo space, which mirrors the function of a CPU. The larger the model, the more memory it requires for training, storage, and inference. For instance, the GPT3 model has 175 billion parameters, and GPT4 has more then a trillion. Such models are trained on clusters comprising thousands of GPU units housed in specialized facilities known as data centers.

1.4 Why?

Combining pattern classifiers nowadays have specific purpose in a specific sub-fields of AI. However, some properties are shared among all types of problems. Ensemble models yield lower error rates since the base estimators are basically 'fighting' each other for permission to grant the most influence in final prediction. Another benefit obtained is reduced overfitting with models that are prone to such behaviour. With development of multi core devices, field of Parallel computing became influential as it allows training and inferencing several algorithms in the same time.

Unstructured Data

Unstructured datasets come in several formats and usually require extensive pre-processing. For example: emails, images, videos and so on. Tasks with unstructured data are mastered by neural networks which, as it has been discussed earlier some categorize as an example of ensemble method. The biggest advantage of neural networks is their scalability, in other words, property that they can benefit from extra data in a way other methods can not. That is achieved by expanding the network architecture(more layers, more neurons). Whereas advantages are obvious in benchmark scores, the shortcomings lie in their interpretability and computational challenges. Combining classifiers may help us in both of those up to a certain point. For example:

- The larger dataset can be used and splitted so that each classifier is fed with enough data to learn specific targets and aggregate the outputs to obtain final prediction.

- Use several smaller classifiers(based on number of parameters) to reach the performance of the larger one

Structured Data

Structured datasets are in a predefined format. They are easy to analyze and generally to work with. For example: financial data, customer information, product specification and so on. Algorithms such as Random Forests, XGBoost and CATBoost usually yield the best performance. Their importance in this type of tasks is not only in ensemble algorithm itself, but also in feature extraction. It is known that machine learning methods outperform standard statistical methods like ANOVA for task of feature selection. One example of such method, which is also example of ensemble method is Boruta[14]. However, this thesis will not be focusing on the feature extraction task so this example serves as a showcase of the reach of ensemble methods.

1.5 Constructing Ensemble System

As it is proposed in a paper[21] by Lior Rokach. Upon constructing ensemble method the following aspects should be defined:

Aspect	Description
Combiner	- Non-trainable: Combiners that do not include training additional parameters, for example *Majority Voting*. - Trainable: Combiners that include training parameters, for example *Naive Bayes Combiner*.
Ensemble method	- Type of Ensembling: algorithms for combining base classifiers, including training them separately (parallel) or training on top of each other (sequential/stacked).
Diversity	- How diversity is introduced?: Techniques for ensuring diversity among base classifiers, and measuring the existing one.
Ensemble Size	- How many different classifiers is it used?: The number of base classifiers that make up the ensemble. Ensemble size can vary from a small number to a large number of classifiers.
Universality	- Can it be used with any classifier?: Whether ensemble methods are compatible with any type of base classifier or whether they have specific requirements or limitations regarding the choice of base classifiers.

Table 1.1: Overview of Ensemble Learning Aspects

An additional category is acknowledged, the meta-classifiers. They are not added as they can be listed as a trainable combiners. They take output labels from models as an input and try to find a patterns in errors of classifiers. There is an interesting debate ongoing on whether Fine-Tuning procedure should be categorized as a meta-classifier.

Chapter 2

Methods

Let's go in-depth into each building block of the ensemble. In the following figure, the complete building procedure can be seen.

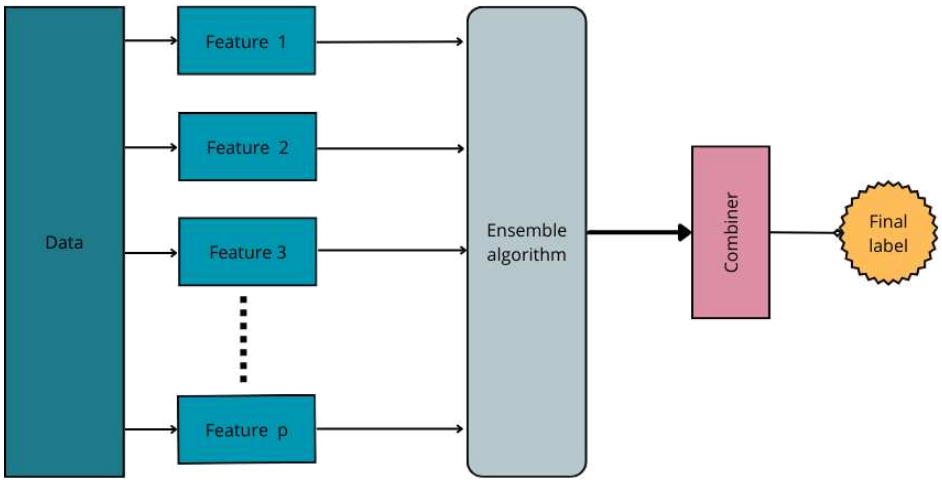


Figure 2.1: General Building Procedure

2.1 Combiner

Combiner in the ensemble build serves as the glue that binds predictions from base learners to reach final decision. There are several approaches to combiners in ensemble learning, each with its unique characteristics depending on the specifications of the system. Major distinguishing factor is whether its worked with continuous or discrete output label. In both scenarios, non trainable and trainable methods can be developed. One special group of those methods are meta-classifiers. As its been mentioned mentioned, they are categorized in trainable subgroup of combiners as they are trained on a set of output labels as the input features and output the final label. Hypothetically, meta-classifiers can be seen as a system for them selves. Typically, each base classifier is trained on the same dataset, followed by the use of a small additional dataset (preferably overlapping with the training set) to train the combiner model. To express these concepts more precisely, an introduction to the probabilistic framework is necessary. The notation is from [13].

An ensemble of classifiers is considered, comprising p base classifiers in the set $G = \{g_1, g_2, ..., g_p\}$ and a set of classes $\Omega = \{w_1, ..., w_c\}$. Four types of classifier outputs are considered:

(a) **Class Labels**
Each classifier g_i produces a class label $w_i \in \Omega$, $i = 1, ..., c$. Thus, for any input x, the p classifier outputs define a vector $\mathbf{l} = [l_1, ..., l_p]^T \in \Omega^p$. In this scenario, the measure of certainty in the predicted label is not considered, and there is no information about the ordering of labels. Additionally, the object x could be either a vector or a matrix. Certain algorithms, like Convolutional Neural Networks support matrix inputs. For ones that don't, input is typically reshaped into a "line."

(b) **Ranked Class Labels**
The output of each g_i classifier is a subset of the class labels Ω, ranked by a certain property. These labels are called Ordinal labels. In this type of problems, standard classifiers such as Logistic Regression perform sub-optimally as they do not take into account the measure of distance from the predicted label to the true label(Does it matter how wrong is something?). For this reason, classifiers with different loss functions and target outputs were developed [4],[22]. Another approach for taking advantage of this type of class labels is utilized in

field of deep learning with Embedding vectors. For each label, high dimensional feature representation is learned and a positional information is added to it[16].

(c) **Numerical Support For Classes**
Each classifier g_i gives a c-dimensional vector $[d_{i,1}, ..., d_{i,c}]^T$ as its output. Value $d_{i,j}$ represents a probability that element x comes from class w_j. Without loss of generality, it can be assumed that the outputs contain values between 0 and 1, spanning the space $[0, 1]^c$. Such normalization can be made with application of softmax, among others.

(d) **Oracle Output**
With these outputs, information as to which class label has the input data point been assigned is disregarded. For a given dataset X, and its element x_j, classifier g_i produces an output element l_i such that $d_{i,j} \in \{0, 1\}$ depending on whether classifier g_i classified object x_j correctly or not. Mathematically speaking, following is found: $\mathbb{1}_{l_i=w_j}$. This type of output is useful in problems with many class labels.

With previously introduced notation for class label outputs $\mathbf{l} = [l_1, ..., l_p]^T \in \Omega^p$, following methods model the probability $P(w_k \text{ is correct } |\mathbf{l})$ for $k = 1, ..., c$.

Discrete Case

➤ **Majority Voting Combiner**
Most common combiner when working with the large number of classifiers is majority voting combiner. Final decision is reached by simply counting 'votes' for each class. If there are p classifiers, and the Oracle outputs are being observed, it is sufficient to have $\lfloor \frac{p}{2} \rfloor + 1$ correct votes. If p_i, $i = 1, ..., p$ represents the probability of each classifier hitting correct label '$p_i = accuracy_i/100$', then in order to reach the maximum potential of combiner, exactly $\lfloor \frac{p}{2} \rfloor + 1$ correct votes are required. Any additional correct vote will negatively impact the overall performance of combiner since the classifier that made it will, on average make 1 correct vote less in the future. On the other hand, if there are $\lfloor \frac{p}{2} \rfloor$ correct votes and $\lfloor \frac{p}{2} \rfloor + 1$ incorrect votes, then, following the same analogy as before, combiner will have the worst possible performance. These special cases are called *Pattern of success/Pattern of failure*. Algorithm goes as follows:

Algorithm 1 Majority Voting Combiner

1: **INFERENCE**
2: **Input:** Base classifiers g_1,..,g_p
3: **Input:** Data point x
4: Find class predictions $l_1, ...l_p$ for each of p classifiers.
5: Calculate $d_{i,j}$ for each class w_j, $j = 1, ..., c$ by each classifier g_i, $i = 1, ..., p$
$$d_{i,j}(x) = \mathbb{1}_{l_i = w_j}$$
6: For x, new element, choose $\hat{y}$ output class s.t. $\hat{y} = \text{argmax}_{j=1}^{c} \sum_{i=1}^{p} d_{i,j}(x)$

In case of a tie, common solution is to randomly select from the top-voted labels.

➤ **Weighed Majority Voting**
Method that fits into category of trainable combiners. It comes as a modification from majority voting combiner in a way that more importance is given to the more relevant classifiers. Relevance of the classifier is determined by its performance on testing set. The lower the error, the greater the relevance. Final prediction is determined as a sum of weighted votes, where class with highest overall support is being selected as the output. Algorithm goes as follows:

Algorithm 2 Weighted Majority Voting Combiner

1: **TRAINING**
2: **Input:** Base classifiers g_1,..,g_p
3: **Input:** Training set for combiner $\{(x_1, y_1), ..., (x_m, y_m)\}$
4: Calculate accuracy $\hat{p}_i$, $i = 1, ..., p$ on Training set
5: Find $\nu_i = \log(\frac{\hat{p}_i}{1-\hat{p}_i}), i = 1, ..., p$
6: **INFERENCE**
7: **Input:** Data point x
8: Find class predictions $l_1, ...l_p$ for each of p classifiers.
9: Calculate $d_{i,j}$ for each class w_j, $j = 1, ..., c$ by each classifier g_i, $i = 1, ..., p$
$$d_{i,j}(x) = \mathbb{1}_{l_i = w_j}$$
10: For x, new element, choose $\hat{y}$ output class s.t. $\hat{y} = \text{argmax}_{j=1}^{c} \sum_{i=1}^{p} \nu_i d_{i,j}(x)$

➤ **Naive Bayes Combiner**
This method falls into category of trainable combiners and differs from weighted majority voting in its use of Bayes' rule. Its simplicity may be deceiving, as it often yields better performance then some more complicated algorithms in specific tasks. To train it, confusion matrix for each classifier is first computed

based on a dataset (overlap with the training set is desirable). Then, the usual practice is to add some correction coefficient ($\epsilon > 0$) to avoid zeroes in confusion matrices. During the inference phase, prior support for each label is found as the quotient of the number of class elements and all elements. Posterior probability of support for each class is then reached as a product of the corresponding elements in confusion matrices of each classifier. At the end, final output label is assigned to class with the largest support. Algorithm proceeds as follows:

Algorithm 3 Naive Bayes Combiner

TRAINING
Input: Base classifiers g_1,..,g_p
Input: Training set for combiner $\{(x_1, y_1, ..., (x_m, y_m)\}$
Find number of elements for each class w_k, m_k
Find confusion matrices $K_i \in \mathbb{R}^{c \times c}$ for $i = 1, ..., p$
Apply correction for zeroes in confusion matrices
INFERENCE
Input: Data point x
Find class predictions l_i for $i = 1, ..., p$.
Set prior support for each class: $\mu_k(x) = \frac{m_k}{m}$
Update support for posterior: $\mu_j(x) \leftarrow \mu_j(x) \prod_{i=1}^{p} K_i(j, l_i)$ for $j = 1, ..., c$
For x, new point, choose $\hat{y}$ output class s.t. $\hat{y} = \text{argmax}_{j=1}^{c} \mu_j(x)$

➤ Behaviour Knowledge System Combiner

BKS is a method that fits into the category of trainable combiners. This combiner works with multinomial combinations of classifier outputs, meaning that it tries to find a pattern in the combination of all classifier outputs rather then observing each separately. During the training phase, a matrix with all combinations of outputs of p classifiers is found and later used during inference to identify the most probable class. Essentially, it counts how many times a received combination of votes occurred and what the output was, assigning a label to the output that occurred most times. The problem with this method is that it requires a large dataset for training, and even when that is fulfilled, it can not fully capture the posterior probabilities of each combination. The algorithm proceeds as follows:

Algorithm 4 Behaviour Knowledge System Combiner

1: **TRAINING**
2: **Input:** Base classifiers g_1,..,g_p
3: **Input:** Training set for combiner $\{(x_1, y_1), ..., (x_m, y_m)\}$
4: Find a pattern matrix on Training set, $E \in \mathbb{R}^{m \times p}$
5: **INFERENCE**
6: **Input:** Data point x
7: Find the class predictions vector $\mathbf{l} = [l_1, ...l_p]$
8: Compare rows in matrix E with vector $\mathbf{l}$ and select one that occurs most frequently

In case of a tie, majority voting combiner can be utilized.

Continuous Case

In continuous case, classifiers with continuous outputs are considered. It is assumed that the classifier returns for each class a degree of support, i.e., a measure proportional to the probability that a data point belongs to a certain class. Thus, each classifier can be redefined as $g_i : \mathscr{R} \rightarrow [0,1]^c$. This notation is possible since softmax transformation can be applied to the outputs, i.e. $Softmax : \mathbb{R}^n \rightarrow [0,1]^n$ where the sum of output vector elements is 1. Decision profile, in notation $DP(x)$ for input data point x is a matrix whose elements $d_{i,j}$ are functions of x and are defined as support that classifier g_i gives to the hypothesis that the output label comes from class w_j.

➤ **Average Combiner**
The most basic combiner in group of continuous labeled output combiners is average combiner. Final label is obtained by averaging supports that each classifier gives to the certain class. So, for a new data point, decision profile is found and after that the supports are calculated. Finally, element is assigned to label with maximum support. Algorithm goes as follows:

Algorithm 5 Average Combiner

1: **INFERENCE**
2: **Input:** Base classifiers g_1,..,g_p
3: **Input:** Data point x
4: Compute decision profile for input data point
5: Find support for each class by computing $\mu_j = \frac{1}{p} \sum_{i=1}^{p} d_{i,j}(x)$ j=1,...,c
6: For x, new point, choose $\hat{y}$ output class s.t. $\hat{y} = \text{argmax}_{j=1}^{c} \mu_j(x)$

Further non-trainable methods, like minima/maxima or median combiners, are also viable options. Additionally, introducing a tunable parameter $\alpha \in \mathbb{R}$ allows for the calculation of support for each label on an input sample x as follows:

$$\mu_j(x) = \frac{1}{p}\left(\sum_{i=1}^{p} d_{i,j}(x)^{\alpha}\right)^{1/\alpha}$$

Other modifications are analogously defined.

➤ **Borda Count Combiner**
Algorithm firstly devised in 1770. by Jean Charles de Borda. It fits into a list of non trainable combiners. Using this combiner, supports for classes that are not most likely are not excluded. In some literature, Borda count can be found grouped within the discrete case label outputs. General condition is that each classifier provides a complete ranking of all possible alternative choices. Therefore, it is necessary for each classifier to give c degrees of support. The most probable class gets $c-1$ points of support, and so on down to the least likely which gets 0 points of support. Final label belongs to a class with the largest score. Algorithm goes as follows:

Algorithm 6 Borda Count Combiner

INFERENCE
Input: Base classifiers g_1,..,g_p
Input: Data point x
Compute decision profile DP(x)
Initialize support for each class: $\mu_j(x) = 0$, for all $j = 1, \ldots, c$
for each classifier g_i in G **do**
    Update score: $\mu_{:}(x) += \mathbb{R}\texttt{ank}([DP(x)]_{:,i})$
end for
For x choose $\hat{y}$ output class s.t. $\hat{y} = \text{argmax}_{j=1}^{c}\mu_j(x)$

Where $\mathbb{R}\texttt{ank}$ function returns ranks across whole column.

➤ **Decision Template Combiner**
Combiner firstly proposed by Kuncheva[13] where the concept of decision profile is used to create decision templates for each class. Decision template captures central tendencies in the continuous outputs of base classifiers. Using certain training set, decision profiles are firstly computed, and then averaged per class to create specific decision templates. New data point is assigned to a class

whose decision template is the most similar to its decision profile. Algorithm goes as follows:

Algorithm 7 Decision Template Combiner

1: **TRAINING**
2: **Input:** Base classifiers g_1,..,g_p
3: **Input:** Data set for training combiner $\{(x_1, y_1), ..., (x_m, y_m)\}$
4: Find decision profiles $DP(x_i)$ for i=1,...,m
5: Compute decision templates for each class as: $DT_j = \frac{1}{m_k} \sum_{x_k : w_j = y_k} DP(x_k)$
6: **INFERENCE**
7: **Input:** Data point x
8: Compute decision profile DP(x)
9: Find similarity score between DP(x) and each DT_j where $j = 1, ..., c$
$$\mu_j(x) = \mathbb{S}(DP(x), DT_j)$$
10: For x, new element, choose $\hat{y}$ output class s.t. $\hat{y} = \text{argmax}_{j=1}^{c} \mu_j(x)$

Two similarity scores were presented in this thesis, including the Mahalanobis distance, which can be defined as follows:

$$\mathbb{S}(DP(x), DT_j) = \left(\sum_{i,k=1}^{p,c} (DP(x)_{i,k} - [DT_j]_{i,k}) \right)^T \Sigma^{-1} \left(\sum_{i,k=1}^{p,c} (DP(x)_{i,k} - [DT_j]_{i,k}) \right) \tag{2.1}$$

Where the Σ is covariance matrix. The second one presented is $Swain\&Ballard$ similarity which can be defined as:

$$\mathbb{S}(DP(x), DT_j) = \frac{\sum_{i,k=1}^{p,c} \min(DP(x)_{i,k}, [DT_j]_{i,k})}{\sum_{i,k=1}^{p,c} [DT_j]_{i,k}} \tag{2.2}$$

In some scenarios it is better to have both similarity scores calculated so to get more robust information on how strong is the combiner(testing accuracy should be similar).

➤ **Dempster-Schafer Combiner**
Dempster-Shafer theory of evidence, originally presented by Barnett and Shafer during the 1970s and 1980s, offers an unique approach to handling uncertainty, differing from Bayesian reasoning by not necessitating prior probabilities for evidence and effectively dealing with incomplete knowledge. Despite its early popularity, it was rejected due to various exposed shortcomings by critics such

as Cheeseman [5] in his paper, interestingly named *In defense of probability*. Nowadays, its applications have been mostly limited to data fusion problems. The Dempster-Shafer combiner operates by first calculating a proximity function using previously defined decision templates, from which belief is derived, and then by employing Dempster's rule of combination[3][20], degree of support for each class is determined. Algorithm goes as follows:

Algorithm 8 Dempster-Schafer Combiner

1: **TRAINING**
2: **Input:** Base classifiers g_1,..,g_p
3: **Input:** Data set for training combiner $\{(x_1, y_1), ..., (x_m, y_m)\}$
4: Find decision profiles $DP(x_i)$ for i=1,...,m
5: Compute decision templates for each class as: $DT_j = \frac{1}{m_k} \sum_{k:w_j=y_k} DP(x_k)$
6: **INFERENCE**
7: **Input:** Data point x
8: Compute decision profile DP(x)
9: Compute proximity score for each $j = 1, ..., c$: $\beta_{j,i}(x) = \frac{(1+||[DT_j]_i - g_i(x)||^2)^{-1}}{\sum_{k=1}^{c}(1+||[DT_k]_i - g_i(x)||^2)^{-1}}$
10: Compute beliefs $b_j(g_i(x)) = \frac{\beta_{j,i}(x)\prod_{k\neq j}(1-\beta_{k,i}(x))}{1-\beta_{j,i}(x)[1-\prod_{k\neq j}(1-\beta_{k,i}(x)]}$ for i=1,...,p
11: Apply Dempster rule: $\mu_j = K\prod_{i=1}^{p} b_j(g_i(x))$
12: For x, new element, choose $\hat{y}$ output class s.t. $\hat{y} = \text{argmax}_{j=1}^{c}\mu_j(x)$

K is here normalization constant ensuring sum of all support values is 1.

➤ **Stacked Generalization Combiner**
This combiner represent a general class of combiners that make use of standard machine learning algorithms such as Logistic Regression, Decision Trees, ANN and so on. It was first considered by Wolpert [25] in 1992. In its most general form, for a new point x, input feature for the combiner is its entire decision profile $DP(x)$ and the output is final class label. This can obviously be simplified by optimizing weights only for a certain base classifier(weighted combiner) or including also weights for each class. General algorithm goes as follows:

Algorithm 9 Stacked Generalization Combiner

1: **TRAINING**
2: **Input:** Base classifiers g_1,..,g_p
3: **Input:** Data set for training combiner $\{(x_1, y_1), ..., (x_m, y_m)\}$
4: **Input:** Combiner classifier f
5: Train combiner classifier f on $\{(D(x_1), y_1), ..., (D(x_m), y_m)\}$ dataset
6: **INFERENCE**
7: **Input:** Data point x
8: Compute predictions for each base classifier g_i, i=1,...,p
9: Predict the final output label $\hat{y} = f \circ D(x)$

Here, D prediction represents abstract input for f combiner which can be decision profile, but it can also be a vector.

2.2 Ensemble methods

Ensemble algorithms boost predictive accuracy by parallelly training different base estimators with mutually compensatory errors. They excel with efficient, high-variance base estimators like Decision Trees or ANN. Combining outputs of multiple models helps avoid the effect of overfitting on training set. Following algorithms will be presented for both types of problems, classification and regression. For that reason, *estimator* notation for base model in ensemble algorithms will be used.

Bagging

Bagging, short for Bootstrap Aggregating, is the simplest ensemble algorithm. In this algorithm, diversity is introduced at the data selection level. For each selected base classifier, subset of data used for training is selected using the bootstrapping method. Bootstrapping uses a uniform distribution to decide whether an element will be in a dataset or not. This means that having dataset X with m elements, each element will have a chance of $\frac{1}{m}$ to end up in a bootstrap sample in each step of selection process. If the element from dataset X is denoted as x, bootstrap sample as X_{boot} and sampler with S, following holds:

$$P(x \notin S) = 1 - \frac{1}{m} \implies P(x \notin X_{boot}) = \left(1 - \frac{1}{m}\right)^m \overset{m\to\infty}{\to} e^{-1} \approx 0.368 \tag{2.3}$$

Thus, each element will have a chance of approximately 36.8% to not end up in bootstrap sample for algorithm iteration for large datasets. In regression scenarios, the

outputs of each regressor are averaged and assigned as the output. In classification, majority voting is commonly used to determine the output label. This method has shown it self to be very efficient with models that yield high variance, for example ANN or Decision Trees. Practically any known machine learning (ML) algorithm can be used as the base classifier/regressor, and it's even possible to combine different types. Linear models, as base classifiers tend not to improve performance as the output ensemble model is, again, linear model. This can be illustrated by considering that the process essentially involves adding models like $y = ax + b$ where y is target, a is weight matrix, x input feature and b bias term. "Best" linear model can be already estimated by the number of different algorithms. The training process is as follows; First, the size of ensemble p is selected. Next step is assigning data points to certain estimator based on results of bootstraping. Finally, a combiner is used to aggregate the outputs of each model during inference.

Algorithm 10 Bagging

1: **Input:** Training set $\{(x_1, y_1), \ldots, (x_m, y_m)\}$, $y_i \in \mathbb{R}$
2: **Input:** Number of iterations p
3: **Input:** Base estimator
4: Initialize sampler vector $S \in \mathbb{R}^m$, $S_i = \frac{1}{m}$, for all $i = 1, \ldots, m$
5: **for** $l = 1$ to p **do**
6: Train tree model g_l using sampler weights S
7: **end for**
8: **Output:** Final model $B(x) = \mathrm{F}\left(h_1(x), .., h_p(x)\right)$

Random Forests

Random Forests algorithm is a modification of Bagging algorithm. In its basic form, apart from using bootstrapping to construct modified datasets, it uses random feature subspaces. This algorithm uses fixed base estimators, Decision Trees. These trees are typically grown to a substantial size, usually with a depth of around 5 to 10 splits. The GINI index or the mean decreased accuracy measure is used to identify the optimal feature and value for expanding the tree.

Key terminology includes Out-Of-Bag(OOB) error estimates which can be easily obtained by evaluating error from trees trained with bootstrap sample on elements left out of bag. Such measures are aggregated across each tree and result is called Generalization error. Generalization error provides a quick and unbiased estimate of prediction error. OOB procedure enables continuous tracking of changes during training upon iterating. It replaces role of *cross-validation*. When some trees are

overfitted, pruning can be employed. It is a method where least valuable leaves are removed to decrease size of tree. Such leaves provide least improvement in explained variability compared to others.

Random Forest algorithm is one of the most frequently used algorithms, but like Bagging, a significant drawback is loss of interpretability compared to ordinary Decision Tree.

Algorithm 11 Random Forests

1: **Input:** Training set $\{(x_1, y_1), \ldots, (x_m, y_m)\}$, $y \in \mathbb{R}$
2: **Input:** Number of iterations p
3: **Input:** Max tree depth d
4: Initialize sampler vector $S \in \mathbb{R}^m$, $S_i = \frac{1}{m}$, for all $i = 1, \ldots, m$
5: **for** $l = 1$ to p **do**
6: Randomly select feature subspace K
7: Train tree model g_l on subspace K using sampler weights S with depth at most d
8: **end for**
9: **Output:** Final model $RF(x) = \mathrm{F}\left(g_1(x), .., g_p(x)\right))$

Same as with Bagging, function F is majority voting in case of the classification problem, and average in case of the regression problem.

Boosting

Family of Boosting algorithms yields some of the best results in machine learning. Goal is to obtain a strong classifier from many weak ones. A strong classifier by itself gives low error on the test set. The Boosting algorithms can be divided into Adaptive Boosting (AdaBoost) algorithms and Gradient Boosters (GB) depending on the feature used for the sequential construction of estimators. This means that there won't be all base estimators defined in the first iteration, but each subsequent estimator will be defined taking into account the previous one. This will provide some good as well as some bad properties. With a large number of iterations, it will be able to overfit the estimator to the training set, so it will be important to adjust number of iterations based on the data set size. AdaBoost methods construct so-called weak learners in each iteration and assign them data depending on the distribution of the samplers. On the other hand, Gradient Boosters model the residuals of estimators in each iteration, based on some predefined loss function. Described procedure is same in the case of problems of classification and regression. Training of AdaBoost algorithm goes as follows:

Algorithm 12 AdaBoost Algorithm

Input: Training set $\{(x_1, y_1), \ldots, (x_m, y_m)\}$, $y_i \in \mathbb{R}$
Input: Weak learner
Input: Error function Err
Input: Number of iterations p
Initialize sampler vector $S \in \mathbb{R}^m$, $S_i = \frac{1}{m}$, for all $i = 1, \ldots, m$
for $l = 1$ to p **do**
    Train weak learner using sampler weights S to get estimator g_l
    $\epsilon_i = l_i \mathrm{Err}(y_i, g_l(x_i))$ for $i = 1, ..., m$
    $\mathrm{eps}_l = \sum_{i=1}^{m} \epsilon_i$
    $\beta_l = \frac{\mathrm{eps}_l}{1-\mathrm{eps}_l}$
    $s_i \leftarrow s_i \beta^{1-\mathrm{eps}_l}$ for $i = 1, ..., m$
    $s_i \leftarrow \frac{s_i}{\sum_{i=1}^{m} s_i}$
end for
Output: Final model $ABoost(x) = \mathrm{F}\left(\beta_1, ..., \beta_l, g_1(x), .., g_l(x)\right)$

Generally, any weak learner can be chosen, similar to Bagging. The error function, denoted as Err, may be defined differently depending on whether the context is regression or classification. Most often, the MSE is used for regression and $0 - 1$ error for classification. It can be proven that the AdaBoost algorithm minimizes *exponential loss*. Important detail is that the values ϵ_i are between 0 and 1 for each $i = 1, ..., m$. Function F is weighted majority voting in case of classification, and in the case of regression, it is a weighted average combiner. Here, the coefficients β_l, $l = 1, ..., p$ represent confidence level assigned to each estimator by algorithm for the final decision-making process[7].

Algorithm 13 Gradient Boosting

Input: Training set $\{(x_1, y_1), \ldots, (x_m, y_m)\}$, $y \in \mathbb{R}$
Input: Weak learner
Input: Differentiable loss function p
Input: Number of iterations p
Initialize starting model $F_o(x) = \text{argmin}_\gamma \sum_{i=1}^{m} L(y_i, \gamma)$
for $l = 1$ to p **do**
    Calculate $r_{il} = -\left[\frac{\partial L(y_i, F_{l-1}(x_i))}{\partial F_{l-1}(x_i)}\right]$ for $i = 1, \ldots, m$
    Train Weak learner $g_l(x)$ using dataset $\{(x_1, r_{1l}), ..., (x_m, r_{ml})\}$
    Calculate $\gamma_l = \text{argmin}_\gamma \sum_{i=1}^{m} L(y_i, F_{l-1}(x_i) + \gamma g_l(x_i))$
    Update model $F_l(x) = F_{l-1}(x) + \gamma_l g_l(x)$
end for
Output: Final model $F(x) = F_p(x)$

For the first iteration of the model, something simple like mean in case of regression, and most frequent class in case of classification can be used. For a more educated guess, Newton-Raphson algorithm can be utilized. Coefficient γ_l, $l = 1, .., , p$ is most often estimated numerically with Gradient Descent. Moreover, a threshold for disregarding low-value learners is often employed to make final learner stronger. In case of Decision Tree weak learner, this algorithm is called *GBM*.

Stochastic Gradient Boosting modifies standard GB to reduce overfitting. Base estimators are trained on random subsets of 50-80 percent of input data without replacement. Recent methods focus strictly on Decision Trees. *XGBoost*, a method introduced in 2014., features modified tree learning algorithms capable of handling sparse data and incorporates regularization coefficients to prevent overfitting. Additionally, an algorithm for optimizing the splitting threshold is employed. More details on this algorithm can be found in [6].

Mixture of Experts

Mixture of Experts is another interesting ensemble algorithm. Novelty compared to previous ensemble algorithms is an element called router. Router's responsibility is to figure out which model would have highest chance to estimate target value for input data point x as accurately as possible. p estimators or experts, either identical or distinct, are defined and trained concurrently alongside the router. Depending on implementation, expert receiving highest degree of support can be used to make an estimation, or multiple experts can be used but their decisions weighted according to output of router. Router is usually a neural network with multiple dense layers

and softmax activation at its output layer. Experts are trained on specific data splits whose combination of inputs and outputs can be successfully modeled. For this type of algorithm, specific loss functions have been developed. Two will be presented here. First is called *Competition encouraged loss function*. For a target vector y, p, number of experts, w_i, $i = 1, ..., p$ router output weights which sum up to 1 and $\hat{y}_i$ prediction vectors of each expert, loss can be calculated as:

$$L_{comp} = \sum_{i=1}^{p} w_i ||y - \hat{y}_i||^2 \tag{2.4}$$

As the name of loss suggests, experts are pitted against each other to get the highest support from router by allowing each of them to make it's own prediction without the impact of weights of other experts. When trained with this loss, router typically assigns a task of estimation to a single expert for each data point. Another popular loss function is called the *Cooperation encouraged loss function*. With the same notation, it is defined as:

$$L_{coop} = ||y - \sum_{i=1}^{p} w_i \hat{y}_i||^2 \tag{2.5}$$

In this case, parameters in each expert model are updated according to the overall loss. This method is prone to overfitting. Mixture of Experts algorithm is even used in modern systems like GPT-4 with purpose of minimizing number of parameters and therefore reducing inference time. When used for larger models, each expert is usually allocated to its own GPU. Its greatest flaw is instability during the training phase. These issues are mostly related to a primitive router output, i.e., assigning each data point to the same expert. Such issues are addressed by adding noise to the data points and applying a label smoothing penalty to the softmax output of the router. Another issue is its slow training. Recently, to address this, Binary-Tree and Fast-Feed-Forward Network architectures/modifications have been introduced.

2.3 Diversity

Diversity is a measure of difference between estimators inside ensemble. It can be introduced in various ways. For example, in previous section I defined ensemble algorithms that introduced diversity through managing input data set, in case of Bagging, Random Forests, Boosting(AdaBoost), and through prediction results in case of Mixture of Experts, Gradient Boosting Machines methods. Aswell as with using different estimators. However, naturally arising question is: "How to measure the amount of diversity ensemble possesses?". Probably the only completely feasible way is to find a difference in output labels between classifiers. So, the greater the

difference between predictions among base classifiers, the greater the diversity. Many diversity metrics have been proposed over the years. I include 3 of them here for pairwise and 3 non-pairwise(global) estimation of diversity.

Pairwise Measures

Here, for a set of p classifiers $G = \{g_1, ..., g_p\}$, diversity is estimated for each pair, so there are $\binom{p}{2}$ scores in total. To reach the final approximation, the average of those scores is taken over the number of combinations. Hence, in following formulas $i, j = 1, ..., p$ and the Oracle outputs are considered as the exponent, i.e. true/wrong classification. Let the input dataset be defined as X.

For classifiers g_i and g_j where $i \neq j$ is not necessarily true:

$N_{i,j}^{0,0}$ - Number of samples correctly classified by both classifiers

$N_{i,j}^{0,1}$ - Number of samples correctly classified by classifier g_j and incorrectly by g_i

$N_{i,j}^{1,0}$ - Number of samples correctly classified by classifier g_i and incorrectly by g_j

$N_{i,j}^{1,1}$ - Number of samples correctly classified by both classifiers g_i and g_j

1. **Disagreement Measure**
   Measure was proposed by Skalak in 1996. in his attempt to find a difference between two base classifiers.

$$Dis(g_i, g_j) = \frac{N_{i,j}^{0,1} + N_{i,j}^{1,0}}{N_{i,j}^{1,0} + N_{i,j}^{0,1} + N_{i,j}^{1,1} + N_{i,j}^{0,0}} \tag{2.6}$$

   which takes value of 0 if used classifiers make the same decisions, i.e. there is no diversity, and 1 if they classify each sample differently. In this scenario, decision is being made based on the Oracle output, which is as its mentioned earlier, indicator function on a set containing correct label. Global score can be reached by averaging over the number of combinations of classifiers:

$$Dis = \frac{2}{p(p-1)} \sum_{i=1}^{p} \sum_{j=i+1}^{p} Dis(g_i, g_j) \tag{2.7}$$

   This equation can be further simplified if it is acknowledged that $N_{i,j}^{1,0} + N_{i,j}^{0,1} + N_{i,j}^{1,1} + N_{i,j}^{0,0} = card(X)$. Then the simplified form of global score is:

$$Dis = \frac{2}{card(X)p(p-1)} \sum_{i=1}^{p} \sum_{j=i+1}^{p} (N_{i,j}^{0,1} + N_{i,j}^{1,0}) \tag{2.8}$$

2. **Q Statistics**
   Measure proposed by George Udny Yule in 1912., known as coefficient of colligation. It's use in measuring ensemble diversity was explored in [23]. Pairwise measure is defined as:

$$Q(g_i, g_j) = \frac{N^{0,0}_{i,j} N^{1,1}_{i,j} - N^{1,0}_{i,j} N^{0,1}_{i,j}}{N^{0,0}_{i,j} N^{1,1}_{i,j} + N^{1,0}_{i,j} N^{0,1}_{i,j}} \tag{2.9}$$

   It takes values between -1 and 1 with the same intuition behind it as in case of Disagreement measure.

   Global score is then:

$$Q = \frac{2}{p(p-1)} \sum_{i=1}^{p} \sum_{j=i+1}^{p} Q(g_i, g_j) \tag{2.10}$$

   Global score provides extra intuition. If $Q > 0$, votes for correctly classified instances tend to be the same; on the other hand, if $Q < 0$, vote is not unanimous. This helps in further assessing ensemble performance.

3. **Interrater Agreement**
   Statistical measure with value between 0 and 1 where lower the value, the higher the disagreement and hence the diversity. Usually, κ notation is used for it:

$$\kappa(g_i, g_j) = \frac{2(N^{1,1}_{i,j} N^{0,0}_{i,j} - N^{0,1}_{i,j} N^{1,0}_{i,j})}{(N^{1,1}_{i,j} + N^{0,1}_{i,j})(N^{1,0}_{i,j} + N^{0,0}_{i,j}) + (N^{1,1}_{i,j} + N^{1,0}_{i,j})(N^{0,1}_{i,j} + N^{0,0}_{i,j})} \tag{2.11}$$

   Global measure is again reached by averaging over the number of combinations of classifiers.

Global Measures

Global measures are somewhat more complicated to define as they must take into account all classifiers and elements in dataset.

4. **Entropy**
   Measure originating from informational theory. It is formulated here as:

$$Ent = \frac{1}{m}(2p - 1) \sum_{j=1}^{m} \min\{s(x_j), p - s(x_j)\} \tag{2.12}$$

   where x_j is element of data set X and $s(x_j)$ is number of correct classifications of x_j among all classifiers.

5. **Interrater Agreement**
   Measure developed for calculating inter-rater reliability(how much do the opinions of independent observers coincide in certain topic). It is derived using coincidence matrix. Procedure can be found in appendix of [13]. It is defined using κ notation as:

$$\kappa = 1 - \frac{\frac{1}{p}\sum_{j=1}^{m} s(x_j)(p - s(x_j))}{m(p-1)\hat{a}(1-\hat{a})} \tag{2.13}$$

   Here, $\hat{a}$ is average classification accuracy among all p classifiers.

6. **The Measure of Difficulty**
   A somewhat more interesting method examined here is measure of difficulty[9]. If X is defined as a random variable that takes values in $\{0, \frac{1}{p}, ..., 1\}$. Its density can be estimated such that, for p classifiers within an ensemble, the test is run on a dataset and for each of the m data points, is observed how many classifiers have correctly classified it. Those are counted, and a histogram is then constructed. If the distribution of volume across the bins is well-balanced, it indicates that ensemble is effective. Conversely, if only one or two bins dominate, this suggests that ensemble is underperforming, i.e. there is no significant diversity. This means that if these ensembles are presented with new, noisy data, they are likely to underperform. Each classifier's precision is represented by $\hat{a}_i$, and they are modeled with a binomial distribution, defined as $(n = m, p = \hat{a}_i)$ for $i = 1, ..., p$. The variance of the random variable is then computed as: $X = \frac{g_1+g_2+...+g_p}{p}$. Variance of random variable X is the global measure of diversity.

2.4 Ensemble Size

Number of models within an ensemble, depending on the situation or use, can enhance the overall prediction or score. That number can be 1, and it can also be 1000. Often, for models that are easy to train, larger ensembles are employed and diversity is introduced through bootstrapping on data. Whereas for larger models like Convolutional Neural Networks, low number of base models is being kept. Reason is the computational bottleneck during training and also during inference (in case of working on a real-time application). For this reason, algorithms such as Boosting are useless when considering them. Mixture of Experts algorithm also creates various issues during its training, but it actually achieves benefits during the inference since it can force use of only one model instead of several (depending on the loss function defined), one assigned by the router to make the prediction.

2.5 Universality

Concept of universality, in the context of an ensemble, refers to the choice of a base estimator. What is questioned, for a predefined algorithm, is whether the specific machine learning algorithm can be used as a base estimator. Some examples of algorithms where there is no universality are: Random Forests, BART (Bayesian Additive Regression Trees), Gradient Boosting Machines, and so on. With these methods, the use is restricted to tree-based models. In contrast, techniques like Bagging or Mixture of Experts allow for the combination of various types of estimators.

Chapter 3

Experiment-Automatic Speech Recognition Task

Speech recognition is one of the oldest problems addressed by machine learning. Hundreds of methods have been proposed to solve it. Initially, the problem was attempted to be solved using Hidden Markov Models[19], but the results were not satisfactory. The next step was made when the problem was tried to be addressed using Artificial Neural Networks. Further improvements came after the two-part system was introduced, the Encoder-Decoder system. The encoder is responsible for gathering relevant data from the input waveform, and the decoder is responsible for generating translation. Recurrent Neural Networks were originally used, combined with Convolutional Networks, but from 2017., the focus shifted to Transformer Neural Networks[24]. The Transformer architecture, and its modifications like Conformer[8] and Branchformer[18], hold the state-of-the-art (SOTA) performance on all benchmark datasets to this day. In this experiment, multiple representations extracted from the waveform will be utilized, and with the application of the introduced methods and algorithms, the goal is to achieve improved results. The Librispeech dataset will be employed for this specific problem. Librispeech is somewhat of a standard regarding speech recognition problems, consisting of 960 hours of speech in the training set, and around 5 hours of speech in smaller datasets 'dev-clean', 'dev-other', 'test-clean', and 'test-other' which are used for model tuning and inference. It is derived from audiobooks that are part of the LibriVox project. These are public domain audiobooks read by volunteers, so the dataset covers a wide range of accents and recording conditions. There are 2,484 different speakers, some male and some female. It is sampled at 16Khz. All 960 hours will be used as the training set to train the Transformer models, and for development, the 'dev-clean' dataset will be used. Models will be evaluated on remaining three datasets.

3.1 Feature Extraction

Feature extraction phase can be split into two groups. Acoustical features, ones which are obtained by applying transformations to waveform, and language features, ones that are obtained by applying transformations to raw transcripts.

Acoustic Features

Most of well-known sound representations for task of speech recognition are being used. *spafe* [15] library was used for the extraction. Details about features are excluded in this work.

1. Log-Mel-Spectrogram
2. Group Delay Mel Spectrogram
3. Constant Q-Transform
4. MFCC
5. Gammatone Spectrogram

The pre-processing is concluded with the application of spectral augmentation and standardization.

Language Features

BPM tokenizer is used to create unigrams for the first trial and characters for the second. Each sentence is converted to a sequence of tokens which start with SOS token, and end with EOS token. BLANK and PAD token are used for leaving space and padding respectively. During the training phase, all samples inside mini-batch are padded to a size of the largest one.
Example of a transcript and it's encoded form for unigram (a) and character (b) tokenizer:

(a)

Transcript: `yesterday is history tommorow is a mystery`

Unigram: `_yesterday_is _history _tom mo row _is _a _mystery`

Tokenized: `[1750, 24, 1075, 903, 924, 2036, 24, 9, 2731]`

(b)

Transcript: `yesterday is history tomorrow is a mystery`

Charseq: `_ y e s t e r d a y _ i s _ h i s t o r y _ t o m o r r o w _ i s _ a _ m y s t e r y`

Tokenized: `[4, 22, 5, 12, 6, 5, 13, 14, 7, 22, 4, 10, 12, 4, 11, 10, 12, 6, 8, 13, 22, 4, 6, 8, 17, 17, 8, 13, 8, 19, 4, 10, 12, 4, 7, 4, 17, 22, 12, 6, 5, 13, 22]`

3.2 Modelling

For modeling, standard Transformer architecture is adapted for working with waveforms. Layer normalization is utilized instead of batch normalization, and also 1D convolutional layers before the input embedding to extract essential features and reduce the length of input sequence. Additionally, weights are shared between character embedding layer and output layer of the last stack in decoder. Residual connections are used before each module in stack. Their role is explored in[10] and serve to prevent an unwanted behaviour of gradients, i.e. *exploding/vanishing gradients*. Intuitively, if optimizer deems layer causes harm to the overall performance, it will zero-out weights inside it and simply use a residual connection to pass the information further. That is very useful in very deep architectures such as ResNets. As an optimizer, Adam[12] was selected and as a loss function, label smoothed cross-entropy with a softening coefficient $\lambda = 0.1$ with aim of avoiding behaviour of model getting stuck in a local optima during training phase. The *backward* pass will not be discussed here.

Below, one can find chosen hyperparameters and architecture used.

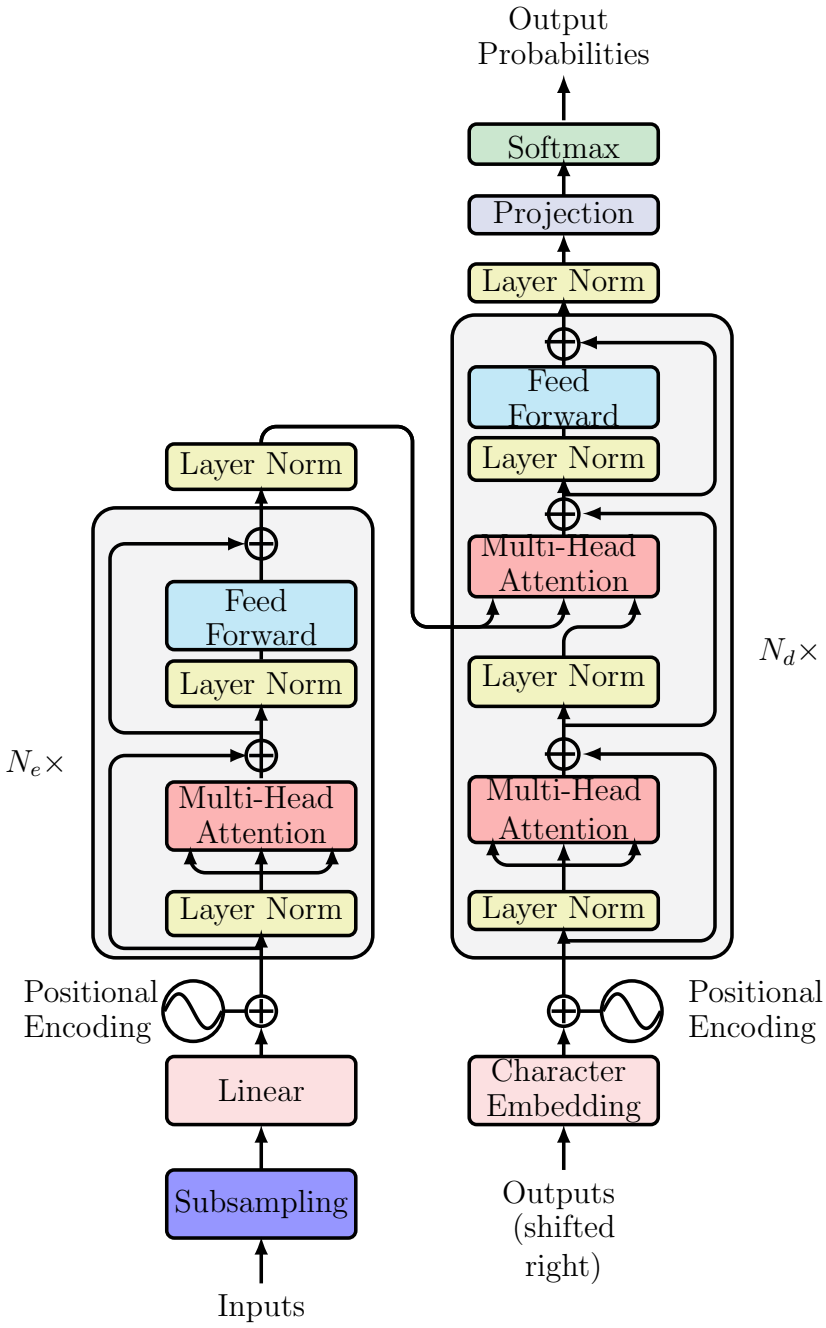


Figure 3.1: Transformer architecture

Architecture modules

- **Subsampling Module**

  The module consists of 2 one-dimensional convolutional layers, that is, the kernel with which it works is one-dimensional. Although 1D convolution on images is not often used, here it works, and significantly reduces computation cost so making the training easier. As with a 2D convolutional layer, filter is passed through the input data, but here only along the x axis, i.e., in one direction only. More filters mean more passes, and a larger output dimension. Also, more parameters. Filter "weights" are optimized using *backpropagation* algorithm. After each convolutional layer, GLU activation function is applied with the aim of reducing the number of channels (they are reduced in half). This is done by fitting a linear network with the number of input and output channels equal to those that the output from the convolutional layer has, then the output of linear layer is divided in half (by channels dimension), sigmoid function is applied to the second half(it can also be applied to the first half) of the expression, and the dot product between first and second parts of the expression is computed. Visualization of module is at 3.2.

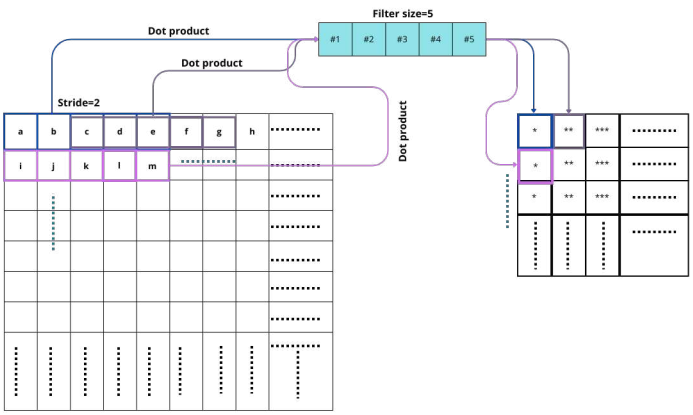


Figure 3.2: Conv1D on image

- **Feed Forward Module**
  The feed forward module consists of 2 fully connected layers, i.e., linear layers. Feed Forward Neural Networks, often called Artificial Neural Networks (ANNs), are the most basic form of neural networks. The input is sent to the first neuron of the first layer and it passes through all the neurons and layers to the output. The weights are then optimized backward, using the *backpropagation* algorithm. In my case, I use 2 such networks/layers. The first reduces the number of channels, while the second increases them, usually by 4×, and a RELU activation function is applied at the output of the layers, which zeroes all values less then 0 on the input expression. General module architecture can be seen at 3.3

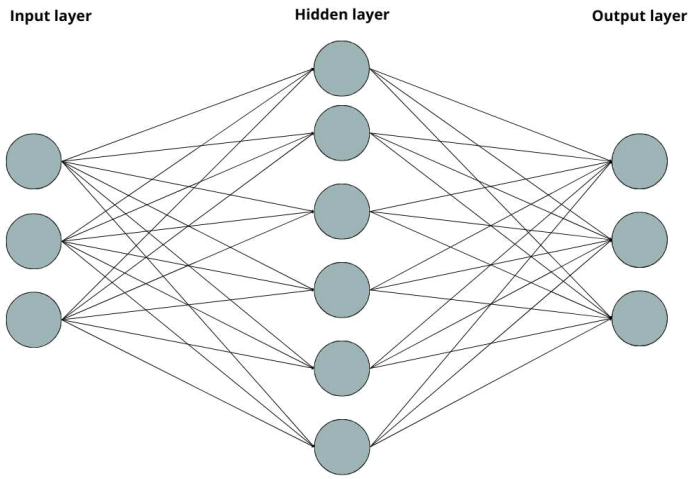


Figure 3.3: Visualization of FFN Module

- **Layer Normalization**
  A layer that performs feature normalization. Previously, batch normalization[11] was used, which looked at an entire batch of samples to estimate the mean and variance of the mini-batch distribution, and then normalized each sample. With

layer normalization[1], each element is normalized using only its own statistical properties. After normalization, values are scaled and shifted using γ and β parameters which are tuned during network training so that the wider range of values can be observed i.e., not to necessarily be restricted to the [0,1] set.

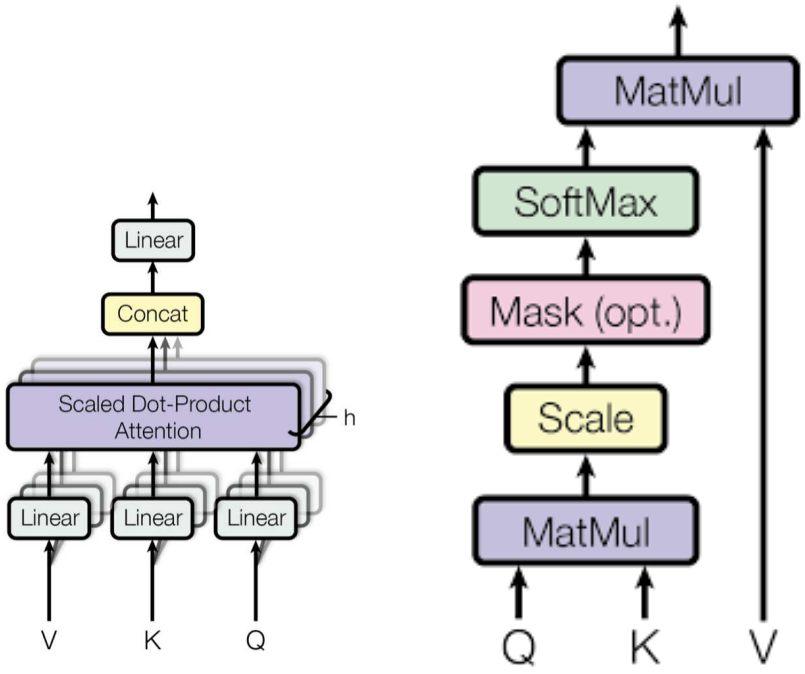
- **Multi-Head-Attention Layer**

Attention layer appeared [2] in 2015. in a paper addressing machine translation problem. This layer is often explained through an inquiry system. Query (Q), value (V), and key (K) matrices are defined as the products of the source and corresponding weight matrices W^Q, W^V, W^K respectively. Within an inquiry context, the query represents a specific element for which representation is sought, the key denotes a value that describes the query, and the value signifies the offered value/response. Passing through attention layer is often referred to as communication phase. Unlike standard attention layer, which observes all inputs simultaneously, multi-head attention layer divides source matrix into n_{heads} pieces along dimension d_{model} so that with my selection of hyperparameters, I have 64 as one dimension for Q, K, V. In case of self-attention, same input is used for query, key, and value, while in case of cross-attention, V and K are obtained from encoder. Common misconception is that query, key, and value matrices are same in case of self-attention. As mentioned, they are obtained by multiplying with differently initialized weight matrices on right. After obtaining Q, K, and V matrices, following graphic illustration 3.4, first multiply Q and K, then apply a mask to their output, used to avoid undesired interactions (only during training phase). After applying mask, softmax function is applied to scale values between 0 and 1, and finally, resulting matrix is multiplied with matrix V. This procedure is applied to each separated piece, then they are concatenated and multiplied by another weight matrix at the end. Notice that weight matrices here are standard linear layers without bias term. Also, in originally released paper, number of attention heads is marked with h, but n_{heads} notation is used in this work.

$$T_i = Attention(Q_i, K_i, V_i) = Softmax\left(\frac{Q_i K_i^T}{\sqrt{d_{model}}}\right) V_i \tag{3.1}$$

for $i = 1, ..., n_{heads}$. T_i matrices are then concatenated and multiplied with another weight matrix.

$$MultiHead(Q, K, V) = Concat(T_1, ..., T_{n_{heads}}) W^O \tag{3.2}$$



(a) Multi Head Attention

(b) Scaled Dot Product Attention

Figure 3.4: Multi Head Attention Visualizations (Vaswani et al. 2017.)

Using this approach, computation is accelerated in comparison to the standard self-attention layer with a single head, enabling parallelization across the number of attention heads. Additionally, it allows the capture of globally less significant connections among tokens.

- **Projection**
  Projection layer is a linear layer which, after the layer normalization has been applied projects output of the last layer of decoder shaped (T,d_{model}) to dimension of vocabulary, i.e. d_{vocab}.

- **Positional Encoding**
  An addition to the extracted embedding in the encoder and decoder. It is added to provide positional information to the data. Following two expressions are

used:

$$PE(p, 2i) = \sin\left(\frac{p}{10000^{\frac{2i}{d_{model}}}}\right) \tag{3.3}$$

$$PE(p, 2i+1) = \cos\left(\frac{p}{10000^{\frac{2i}{d_{model}}}}\right) \tag{3.4}$$

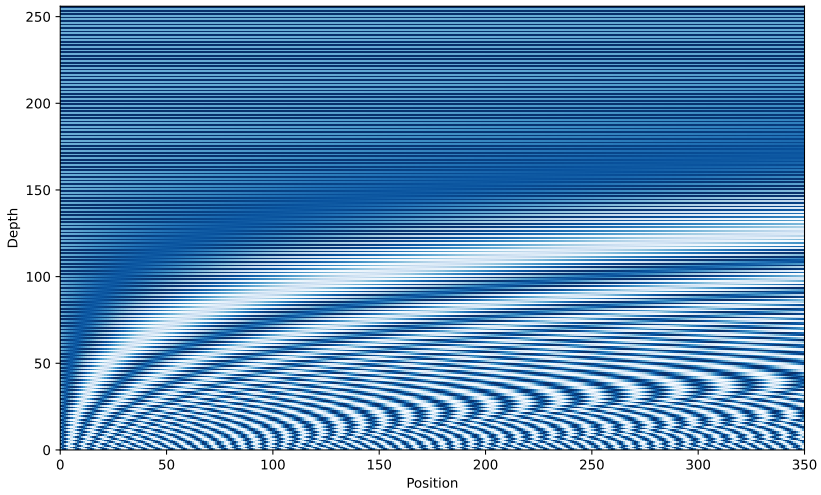


Figure 3.5: Positional encoding

- **Softmax** Activation function used at the output of final layer. Sometimes modified to log-softmax with purpose of numerical stability. For a vector $x \in \mathbb{R}^n$ it is defined as:

$$Softmax(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{3.5}$$

Selected Hyperparameters For Base Classifiers

- Preprocessing
  - Number of Mel Bins: 80
  - Sampling Rate: 16000
  - Hop Size: $0.01s$

  - Frame Length: $0.025s$
  - Vocabulary Type: Unigram - Charseq
  - Vocabulary Size: 10000 - 32
  - SOS Token: $< s >$
  - EOS Token: $< /s >$
  - PAD Token: $< PAD >$
  - BLANK Token: $< BLANK >$

- Encoder
  - Number of Layers : 12
  - Dropout : 0.1
  - Number of Neurons in FFN: 2048
  - Model Dimension: 256
  - Embedding Dimension: 256
  - Number of Attention Heads: 4

- Decoder
  - Number of Layers : 12
  - Number of Attention Heads: 4
  - Dropout : 0.1
  - Number of Neurons in FFN: 2048

- Optimizer & Loss function
  - Optimizer : Adam
  - Loss Function : Label Smoothed Cross Entropy

- Learning Rate

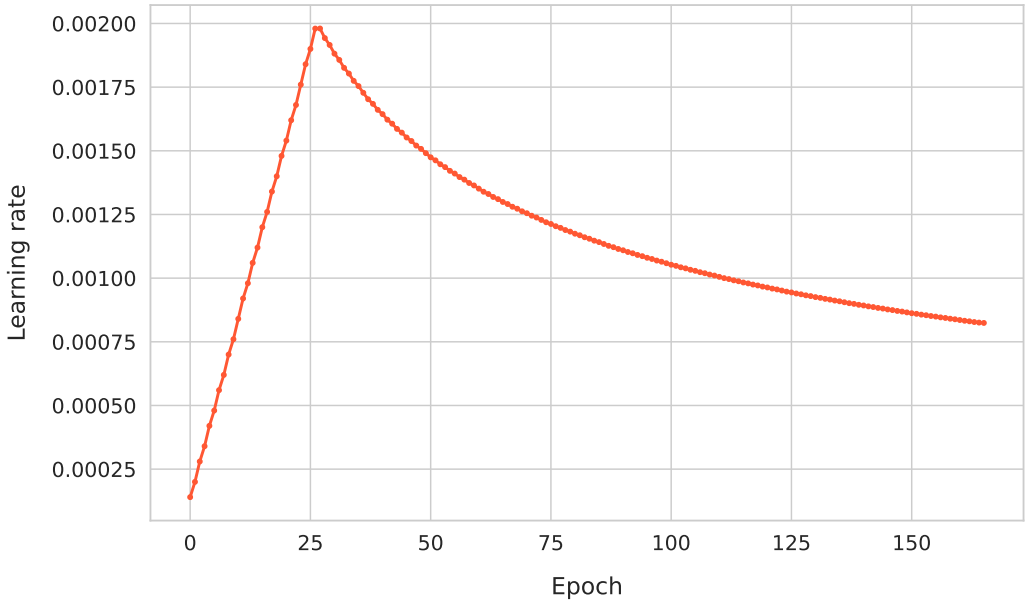


Figure 3.6: Learning Rate

Forward Pass During Training

The input sequence first passes through a subsampling module where the main goal is to reduce its length thus preventing OOM(out of memory) issues. It then encounters a linear layer whose aim is to extract the desired features. Positional information is added to the output of the linear layer, and then it is sent to the encoder stacks. As it can be seen in Figure 3.1, they are composed of multi-head attention and feed-forward modules, in front of which is performed layer normalization, and which are connected by residual connections. After the data passes through encoder stack N_e times, layer normalization is applied to the output of the last stack, and this information is then transferred to the decoder through a multi-head attention layer. During training, ground truth transcript is passed as input to the decoder, where each token is then embedded into a vector of size d_{model}. Positional information is added to this vector. A mask is applied to the extracted sequence, which is intended to prevent interaction with future tokens (those yet to be predicted). The data undergoes self-attention and cross-attention, utilizing information from the encoder, and finally passes through a feed-forward module. Following N_d passes through the decoder stack, layer normalization is applied to the output, which is then projected onto a dimension equal to the size of the vocabulary. Subsequently, softmax is applied to scale the values between

0 and 1. Token is selected using one of the decoding algorithms and fed back into the input of the decoder. During the training of the network, decoder is allowed to see previous ground truth tokens when generating a new one, while during inference it has to rely on its own generated tokens. Mathematically speaking, during training phase it tries to find $P(\hat{y_k}|y_{k-1}, ..., y_0, x)$, and during inference $P(\hat{y_k}|\hat{y}_{k-1}, ..., \hat{y_0}, x)$ where x is the input data, i.e., the waveform in this example. In literature the explained process during training is usually reffered to as *Teacher forcing*. Models which work this way are called *autoregressive* models.

Forward Pass During Inference

During inference phase, the mask is not being applied during decoding, and the input transcript given to the decoder is a vector of size max_len that starts with the SOS token and is filled with PAD tokens to the end. Decoding stops when the EOS token is produced.

Decoding Algorithm During Inference

Specific algorithms can be used during the inference phase for decoding. Most well-known ones are Greedy Decode and Beam Search. Beam Search takes into account multiple choices for a potential token, and using aligning algorithm, it calculates a score for each such sequence of choices, and selects the best one. Number of choices that will be considered, and the length of the beam are hyperparameters. Beam Search represents current standard among such algorithms. On the other hand, much simpler Greedy Decode algorithm only considers the possibility of choosing the token that has the highest probability (numerical support). Simpler algorithm will be used in this thesis primarily for computational reasons.

Evaluation Metrics

In the task of speech recognition, word error rate and character error rate metrics are ones commonly used for benchmarking. word error rate(WER) and character error rate(CER) are calculated as following:

$$WER = \frac{S_w + D_w + I_w}{N_w} = \frac{S_w + D_w + I_w}{S_w + D_w + C_w}$$

$$CER = \frac{S_c + D_c + I_c}{N_c} = \frac{S_c + D_c + I_c}{S_c + D_c + C_c}$$

where $\_w$ and $\_c$ indexes marks level at which changes are being observed. In the case of WER metric, changes at word level are being observed, while with CER metric changes at character level are observed. General notations are as follows:

- S - number of substitutions
- D - number of deletions
- I - number of insertions
- C - number of correct elements
- N - number of elements in the reference (N=S+D+C).

Chapter 4

Results

Complete experiment was conducted using two distinct vocabularies. In total, ten Transformer Neural Networks were trained, each going through 90 to 160 passes over dataset, using a batch size of around 230. Training each network took 24 hours for unigrams, where vocabulary consisted of 10,000 tokens, and 36 hours for character tokens per base classifier. Models were trained using Fairseq framework[17], utilizing an RTX3090 for majority of training and dual RTX6000ADA GPUs for additional training when deemed necessary. Unigram model has 29,536,256 learnable parameters, while character token model has about 3 million fewer. This difference is due to projection layer, where unigrams are projected to a dimension of 10,000, while characters are projected to a dimension of 32, matching size of vocabularies.

After training phase, parameter dictionaries were extracted, and a separate pipeline was created for individual experiments. Fewer experiments were conducted with unigrams due to impracticality of using trainable ensemble algorithms with such a large class set. At the beginning of each section, training losses will be presented according to the chosen loss function, i.e., cross-entropy. For the development phase, I used *dev-clean* dataset which is somewhat less complex, therefore slightly lower loss will be seen on development set then on the training set. Experiments will be organized starting with basic ones and gradually moving to more complex ones. For training combiners and assessing diversity measures, I also used *dev-clean* dataset. Experiments were evaluated on *test-clean, dev-other, test-other* datasets, ensuring comprehensive assessment across various data conditions and robust evaluation of model performance.

4.1 Unigram Based Models

Firstly, the approach with unigrams is observed. Despite extra parameters, inference time is approximately three times lower compared to working with characters. The reason is that generating a "complete" token string (a full sentence) requires far fewer passes through the decoder. In the following graphical representations, training losses can be observed. Models with Group Delay features and Gamma features failed to capture some specific properties of the input features and were thus useless, even detrimental to ensemble performance during deeper analysis. Model with Constant q Transform features performs slightly worse then the remaining two based on MEL and MFCC features, but it will be shown later that it also contributes to the diversity of the ensemble.

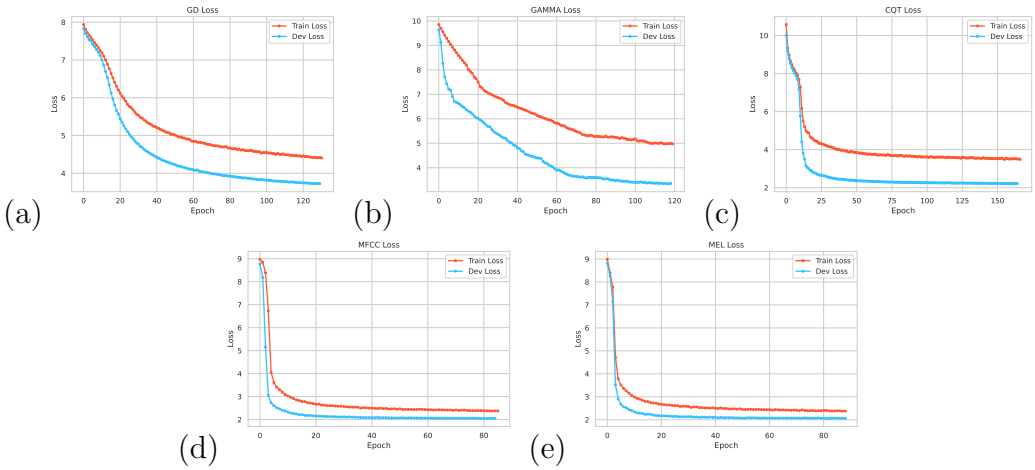


Figure 4.1: (a) GD (b) GAMMA (c) CQT (d) MFCC (e) MEL

Comparing Model Performances

Individual models were evaluated on described three datasets and compared with the current SOTA model, E-Branchformer. It's important to note that the actual SOTA is even lower(better), but was achieved using alternative datasets, so it won't be included here. Most studies don't include CER metric as it has already been reduced to very low levels, and therefore it is simply denoted as "$< 1\%$". In the following table, scores for each model can be seen:

Table 4.1: Single Model Performance

data	**metric**	**MFCC**	**CQT**	**MEL**	**GAMMA**	**GD**	**SOTA**
test-clean	CER	2.11	4.43	2.05	49.66	45.62	$< 1\%$
	WER	5.07	9.16	5.05	68.65	68.37	2.49
dev-other	CER	6.27	12.55	5.91	57.67	55.65	$< 1\%$
	WER	12.77	22.07	11.91	81.09	81.06	5.68
test-other	CER	6.63	13	6.06	63.92	59.32	$< 1\%$
	WER	13.7	23.25	12.78	89.24	86.64	5.61

Non Trainable Combiners

Since the output labels are continuous, impact of simple, non trainable combiners 5 is explored. When combining models, Gamma and Group delay models were excluded as they only worsened performance. For example, I experimented with a simple average combiner and obtained WER and CER scores of 5.73% and 2.5%, respectively, which are worse then the score of (better) base model. Following table shows the results, where 'GEOM' indicates geometric mean, 'HARM' harmonic mean, 'MAX' maximum, and 'AVG' simple average combiner.

Table 4.2: Non Trainable Combiners

data	**metric**	**AVG**	**GEOM**	**HARM**	**MAX**	**SOTA**
test-clean	CER	1.86	1.87	1.868	2.28	$< 1\%$
	WER	4.66	4.66	4.66	5.38	2.49
dev-other	CER	5.31	5.29	5.29	6.68	$< 1\%$
	WER	10.73	10.71	10.70	12.87	5.68
test-other	CER	5.49	5.48	5.46	6.98	$< 1\%$
	WER	11.49	11.47	11.45	13.79	5.61

Weighted Average Combiner

Following the impressive results with simple combiners, the question arises whether even better performance can be achieved. In the case of the weighted average combiner, the goal is to determine optimal linear combination of coefficients $\delta_{GAM}, \delta_{MEL}, \delta_{CQT}, \delta_{MFC}, \delta_{GD}$ such that $\delta_{GAM} + \delta_{MEL} + \delta_{CQT} + \delta_{MFC} + \delta_{GD} = 1$ and delta cofficients are real. Indexes are adjusted so the notation is more readable.

$$\hat{Y}_w = \delta_{GAM} \begin{bmatrix} \hat{Y}_{GAM1} \\ \hat{Y}_{GAM2} \\ \vdots \\ \hat{Y}_{GAM_{10k}} \end{bmatrix} + \delta_{MEL} \begin{bmatrix} \hat{Y}_{MEL1} \\ \hat{Y}_{MEL2} \\ \vdots \\ \hat{Y}_{MEL_{10k}} \end{bmatrix} + \delta_{CQT} \begin{bmatrix} \hat{Y}_{CQT1} \\ \hat{Y}_{CQT2} \\ \vdots \\ \hat{Y}_{CQT_{10k}} \end{bmatrix} + \delta_{MFC} \begin{bmatrix} \hat{Y}_{MFC1} \\ \hat{Y}_{MFC2} \\ \vdots \\ \hat{Y}_{MFC_{10k}} \end{bmatrix} + \delta_{GD} \begin{bmatrix} \hat{Y}_{GD1} \\ \hat{Y}_{GD2} \\ \vdots \\ \hat{Y}_{GD_{10k}} \end{bmatrix} \tag{4.1}$$

I firstly evaluated all base classifiers on 'dev-clean' dataset so to avoid any connection with the actual test sets. For a referent measure, I selected WER. MFCC model obtained 4.76%, MEL model obtained 4.63% ,CQT model obtained 8.406% , GAMMA model obtained 69.99% GD model obtained 66.21% WER.

Using the coefficients from 2, I found: $\delta_{GAM} = -0.12289, \delta_{MEL} = 0.439, \delta_{CQT} = 0.34665, \delta_{MFC} = 0.43482, \delta_{GD} = -0.09762$.

Following results are obtained:

Metric	test-clean	dev-other	test-other
CER	1.7667	5.26	5.286
WER	4.506	10.70	11.267

Table 4.3: Weighted Average Combiner Scores

I acknowledge that these weights may not be the best possible choice, but they represent a solid starting point.

4.2 Diversity Between Classifiers

As it is previously mentioned, diversity is a key ingredient in constructing an efficient ensemble. It can be divided into positive and negative. In this case, negative diversity comes from the GAMMA and GD models, while the remaining three contribute positive diversity. Generally, diversity is positive as long as it does not harm overall performance. In Chapter Two, some methods for its measurement were presented. In this context, one measure will be used for each pair (Disagreement 2.6), along with a global measure derived from a histogram (Difficulty measure;there will only be graphical representation).

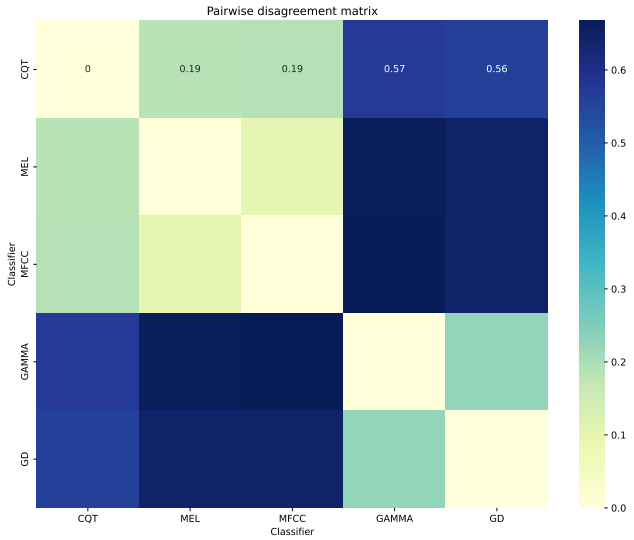


Figure 4.2: Disagreement Graph

Based on the graph, it can be observed that the highest diversity is exhibited by the GD and GAMMA models, but as its mentioned, these are instances of negative diversity. On the other hand, CQT demonstrates relatively good diversity, while MEL and MFCC exhibit very low diversity, which is expected considering the nature of how each feature is derived.

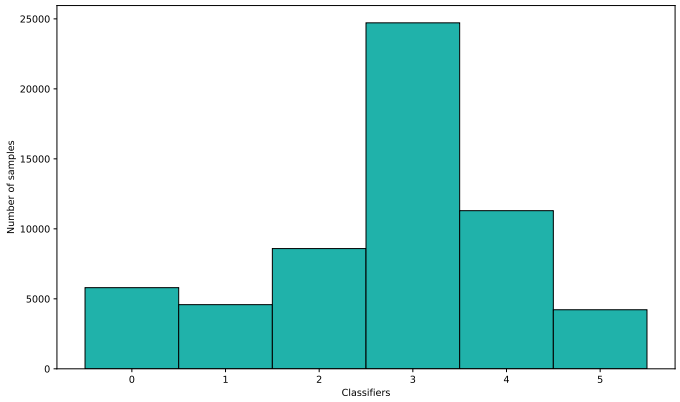


Figure 4.3: Difficulty Histogram

In the histogram, it's evident that the majority of samples were correctly classified by three classifiers. This aligns with expectations, as the GD and GAMMA models generally yield poorer performance. This fact contributes to the relatively small number of instances classified correctly by all five classifiers. It can be observed that there is a subset of data that only one classifier correctly identifies, and similarly for two classifiers also. This signals that the ensemble is indeed successful, but can yet be improved by further studying sources of errors.

4.3 Character Based Models

Now let's consider the case where the tokens are characters. The vocabulary consists of 32 characters in total. Their advantage is that they less frequently encounter contextual errors compared to unigrams and there are no OOV(out of vocabulary) elements. In the following graphs, similar to unigrams, losses during training can be observed for each individual base classifier. Here, the GD model did not achieve nearly as good results as in the case of unigrams. Also, the GAMMA model achieved surprisingly good results, even on par with MEL and MFCC. The CQT model performed somewhat worse compared to the unigram case. During following experiments GD model will not be used as it produces defective outputs.

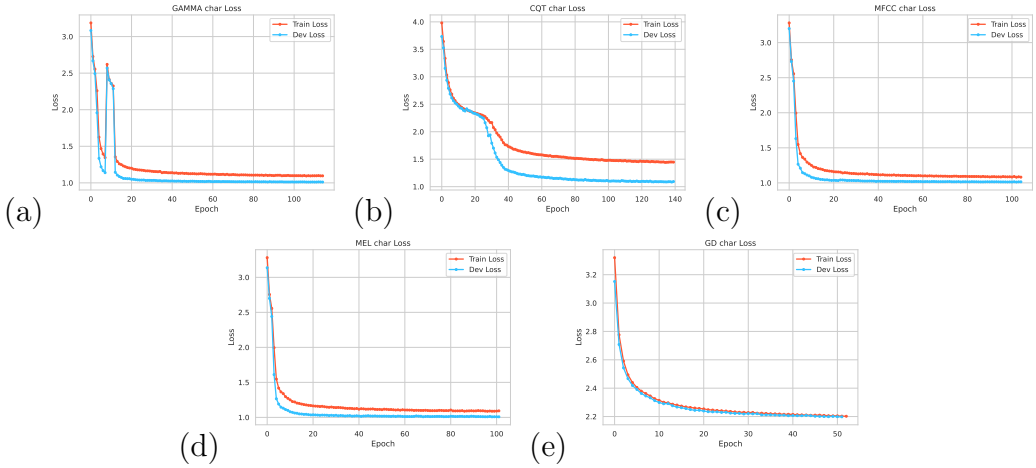


Figure 4.4: (a) GAMMA (b) CQT (c) MFCC (d) MEL (e) GD

Comparing Model Performances

Similarly as with unigrams, each of four selected models were evaluated on three testing subsets. Scores are in the following table:

Table 4.4: Single Model Performance

data	**metric**	**MFCC**	**CQT**	**MEL**	**GAMMA**	**SOTA**
test-clean	CER	4.51	25.08	4.36	4.69	$< 1\%$
	WER	7.72	21.6	7.43	7.99	2.49
dev-other	CER	8.19	23.99	7.65	8.56	$< 1\%$
	WER	15.14	39.71	14.13	15.8	5.68
test-other	CER	8.77	25.08	8.15	9.48	$< 1\%$
	WER	16.73	41.69	15.32	17.50	5.61

Average Combiner and Modifications

Since the output labels are continuous, impact of simple, non trainable combiners is observed 5, 6.

Table 4.5: Non Trainable Combiners

data	**metric**	**AVG**	**GEOM**	**HARM**	**MAX**	**BORDA**	**SOTA**
test-clean	CER	4.22	4.26	4.27	4.793	4.54	$< 1\%$
	WER	7.22	7.22	7.23	8.294	7.82	2.49
dev-other	CER	7.02	7.034	7.02	7.87	8.14	$< 1\%$
	WER	13.06	13.08	13.07	14.46	15.22	5.68
test-other	CER	7.67	7.68	7.67	8.43	8.54	$< 1\%$
	WER	14.53	14.53	14.52	15.82	16.4	5.61

I conducted experiments with various subsets of base classifiers and found that using all of them together yielded the best performance. For instance, when I removed CQT due to its notably poorer performance compared to others, relative performance dropped by approximately 3%.

Weighter Average Combiner

For the weighted average combiner similarly as before, I tried to find $\delta_{GAM}, \delta_{MEL}, \delta_{CQT}, \delta_{MFC}$ such that the $\delta_{GAM} + \delta_{MEL} + \delta_{CQT} + \delta_{MFC} = 1$ and delta cofficients are real. Indexes are adjusted so the notation is more readable.

$$\hat{Y}_w = \delta_{GAM} \begin{bmatrix} \hat{Y}_{GAM_1} \\ \hat{Y}_{GAM_2} \\ \vdots \\ \hat{Y}_{GAM_{32}} \end{bmatrix} + \delta_{MEL} \begin{bmatrix} \hat{Y}_{MEL_1} \\ \hat{Y}_{MEL_2} \\ \vdots \\ \hat{Y}_{MEL_{32}} \end{bmatrix} + \delta_{CQT} \begin{bmatrix} \hat{Y}_{CQT_1} \\ \hat{Y}_{CQT_2} \\ \vdots \\ \hat{Y}_{CQT_{32}} \end{bmatrix} + \delta_{MFC} \begin{bmatrix} \hat{Y}_{MFC_1} \\ \hat{Y}_{MFC_2} \\ \vdots \\ \hat{Y}_{MFC_{32}} \end{bmatrix} \quad (4.2)$$

Following the same approach as with unigrams, MFCC model obtained 7.458%, MEL model obtained 7.36%, CQT model obtained 21.44% and GAMMA model obtained 7.9112% WER.

Using the coefficients from 2 I found: $\delta_{GAM} = 0.278663, \delta_{MEL} = 0.287689, \delta_{CQT} = 0.147510, \delta_{MFC} = 0.286137$.

Following results are obtained:

Metric	test-clean	dev-other	test-other
CER	4.23	6.846	7.462
WER	7.13	12.766	14.21

Table 4.6: Weighted Average Combiner Scores

Generalized Stacking

For the next experiment, let's consider a classifier $g : \mathbb{R}^{32} \rightarrow \{1, ..., 32\}$ s.t. input data is numerical support obtained for each token and the labels are the ground truth tokens. The experiment is being done with different amounts of tokens and different ML algorithms. It has been experimented with training set consisting of chunks of *train-clean-100* and *train-other-500* datasets having around 500000 tokens in total. For the development set, I took around 150000 tokens from *dev-clean* subset. Following are the results on the development set.

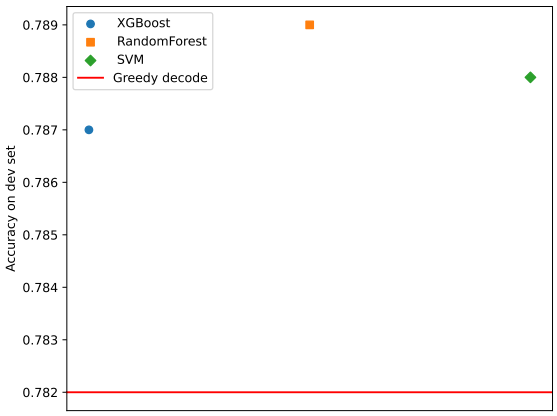


Figure 4.5: Results With Meta Classifiers

As it can be seen, experiment did not yield any significant improvement in accuracy score compared to simple greedy selection. One possible reason is a natural class imbalance, where specific characters are simply more often used and thus the algorithms have a harder time adapting. Most of the ensemble algorithms were able to learn the principle of greedy decode but could not of improved further from that. I experimented with usage of Random Forests on the actual dataset with MELs and it actually decreased the baseline4.4 CER and WER score to 4.777 and 8.295 respectively. Certain unexpected behaviour was introduced also, such as deformed sequences in some samples.

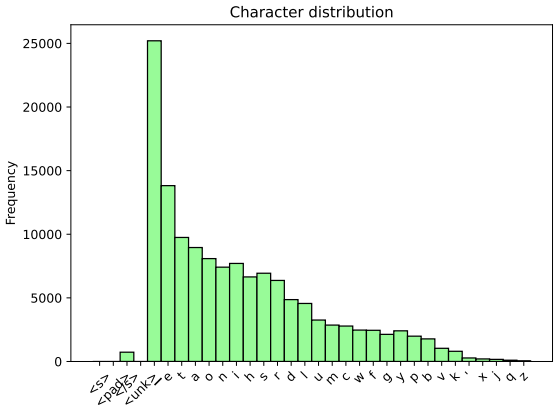


Figure 4.6: Distribution of Characters For *dev-clean*

Decision Template

For the final experiment, decision template was chosen. Algorithm 7 is used. The experiment is conducted at three levels, depending on the amount of tokens used to train the decision templates. At the Figure 4.6 it can be observed that there is a significant class imbalance, so it is expected that a larger training set will provide a better estimation of the base classifiers decisions for each class. Furthermore, this method is suitable for the given problem because class imbalance has no impact on the decision templates themselves; the greater the number of elements for a certain class, the larger the divisor for their sum. Swain & Ballard similarity was used as it proved to be more effective then Mahalanobis. The results are in the following table:

Table 4.7: Decision Template Combiner

data	**metric**	**S**	**M**	**L**	**SOTA**
test-clean	CER	4.23	4.2	4.21	$< 1\%$
	WER	7.16	7.11	7.12	2.49
dev-other	CER	6.92	6.95	6.94	$< 1\%$
	WER	12.9	12.91	12.9	5.68
test-other	CER	7.63	7.56	7.57	$< 1\%$
	WER	14.43	14.53	14.44	5.61

S, M, and L denote the quantities of tokens used for training the templates, corresponding to $200,000$, $600,000$, and $1M$ tokens, respectively. This combiner profits most from correctly picked training set. Hence, it is hypothesized that a better selection of training data could improve results. The first batch was derived from a subset of data with minimal noise, the second batch from a noisier subset, and the third batch once again from a less noisy subset. I acknowledge the further potential of this method in task of speech recognition.

4.4 Diversity Between Classifiers

Using a similar procedure as with unigrams, following graphs are obtained:

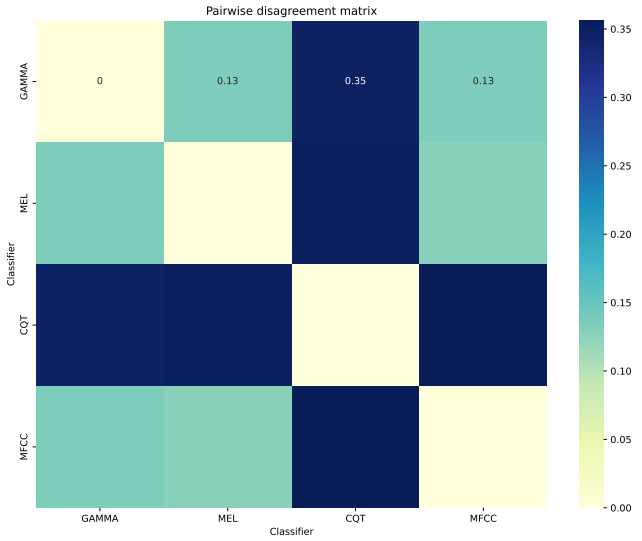


Figure 4.7: Disagreement Graph

It can be observed that decisions of GAMMA, MEL, and MFCC are very similar, especially when compared with ones on unigrams. CQT model contributes the most to diversity, in a positive way.

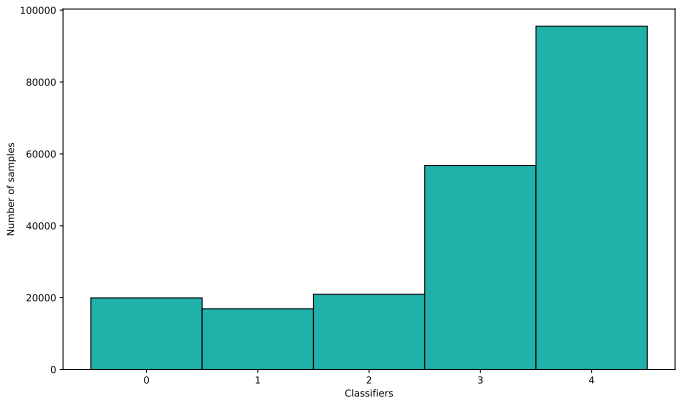


Figure 4.8: Difficulty Histogram

There are many easy points, meaning those for which all four classifiers made the correct decision. Conversely, looking at the number of instances where fewer classifiers

are accurate, there are subsets of data whose elements are correctly identified by only one classifier, and likewise, there are those that two classifiers correctly recognize, and so on. Ultimately, the provided histogram indicates that used ensemble is successful.

Chapter 5

Conclusion

Firstly, it was observed that varying the size of the vocabulary can significantly influence a model's performance. The CQT model, for example, shows far better results with unigrams compared to characters, whereas the GAMMA model is more effective with a smaller vocabulary, though the reasons for these differences are unclear. It was observed that using unigrams resulted in significantly better results compared to using characters as tokens. In both methodologies, non-trainable combiners led to a 5-7% relative improvement in WER score on all 3 testing datasets, while trainable ones achieved an increase of 7-10%. The best improvement was achieved by decision templates combiner at the *test-clean* dataset in case of character tokens. The weighted average yields the best performance on the other datasets. These results are significant, considering use of a more compact transformer model, yet results on par with much larger versions were attained by integrating multiple smaller models. This accomplishment aligns with my initial objective. Additionally, the transformer ensemble is suitable for parallelized inference in accordance with the number of used features, efficiently tackling the issue of slow inference encountered in larger transformers, ones with over 70 million learnable parameters.

Bibliography

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton, *Layer normalization*, 2016.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio, *Neural machine translation by jointly learning to align and translate*, 2016.

[3] Yaxin Bi, David Bell, Hui Wang, Gongde Guo, and Kieran Greer, *Combining multiple classifiers using dempster's rule of combination for text categorization*, Modeling Decisions for Artificial Intelligence (Berlin, Heidelberg) (Vicenç Torra and Yasuo Narukawa, eds.), Springer Berlin Heidelberg, 2004, pp. 127–138, ISBN 978-3-540-27774-3.

[4] Wenzhi Cao, Vahid Mirjalili, and Sebastian Raschka, *Rank consistent ordinal regression for neural networks with application to age estimation*, Pattern Recognition Letters **140** (2020), 325–331, ISSN 0167-8655, `http://www.sciencedirect.com/science/article/pii/S016786552030413X`.

[5] Peter Cheeseman, *In defense of probability*, Proceedings of the Ninth International Joint Conference on Artificial Intelligence (2003).

[6] Tianqi Chen and Carlos Guestrin, *XGBoost*, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, aug 2016, `https://doi.org/10.1145%2F2939672.2939785`.

[7] Harris Drucker, *Improving regressors using boosting techniques*, Proceedings of the 14th International Conference on Machine Learning (1997).

[8] Anmol Gulati, James Qin, Chung Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, and Ruoming Pang, *Conformer: Convolution-augmented transformer for speech recognition*, 2020.

[9] L.K. Hansen and P. Salamon, *Neural network ensembles*, IEEE Transactions on Pattern Analysis and Machine Intelligence **12** (1990), no. 10, 993–1001.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, *Deep residual learning for image recognition*, 2015.

[11] Sergey Ioffe and Christian Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, 2015.

[12] Diederik P. Kingma and Jimmy Ba, *Adam: A method for stochastic optimization*, 2017.

[13] Ludmila I. Kuncheva, *Combining pattern classifiers*, A JOHN WILEY SONS, INC., 2004.

[14] Miron B. Kursa and Witold R. Rudnicki, *Feature selection with the boruta package*, Journal of Statistical Software **36** (2010), no. 11, 1–13, https://www.jstatsoft.org/index.php/jss/article/view/v036i11.

[15] Ayoub Malek, *Spafe: Simplified python audio features extraction*, Journal of Open Source Software **8** (2023), no. 81, 4739, https://doi.org/10.21105/joss.04739.

[16] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, *Efficient estimation of word representations in vector space*, 2013.

[17] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli, *fairseq: A fast, extensible toolkit for sequence modeling*, Proceedings of NAACL-HLT 2019: Demonstrations, 2019.

[18] Yifan Peng, Siddharth Dalmia, Ian Lane, and Shinji Watanabe, *Branchformer: Parallel mlp-attention architectures to capture local and global context for speech recognition and understanding*, 2022.

[19] L. R. Rabiner and B. H. Juang, *Hidden markov models for speech recognition — strengths and limitations*, Speech Recognition and Understanding (Berlin, Heidelberg) (Pietro Laface and Renato De Mori, eds.), Springer Berlin Heidelberg, 1992, pp. 3–29.

[20] Galina Rogova, *Combining the results of several neural network classifiers*, pp. 683–692, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, ISBN 978-3-540-44792-4, https://doi.org/10.1007/978-3-540-44792-4_27.

[21] Lior Rokach, *Taxonomy for characterizing ensemble methods in classification tasks: A review and annotated bibliography*, Computational Statistics Data Analysis **53** (2009), 4046–4072.

[22] Xintong Shi, Wenzhi Cao, and Sebastian Raschka, *Deep neural networks for rank-consistent ordinal regression based on conditional probabilities*, 2021.

[23] E. Tang, Ponnuthurai Suganthan, and Xin Yao, *An analysis of diversity measures*, Machine Learning **65** (2006), 247–271.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, *Attention is all you need*, 2023.

[25] David H. Wolpert, *Stacked generalization*, Neural Networks (1992), 241–259, ISSN 0893-6080, https://www.sciencedirect.com/science/article/pii/S0893608005800231.

Sažetak

U ovome radu fokus je bio na metodama kombiniranja klasifikatora. Osnovni algoritmi nisu detaljno opisani već su korišteni kao temelj uz pretpostavku o znanju u području strojnog učenja. Prvo se izvodi nekoliko kombinatora, tj. algoritama koji stvaraju sintezu među izlazima više klasifikatora. Zatim su predstavljeni najčešće korišteni algoritmi u ansamblima, vođeni težnjom za stvaranjem optimalnog ansambla, kroz koncept raznolikosti opisani su ključni elementi uspješnog ansambla. Nakon toga su testirane neke od prikazanih metoda na problemu prepoznavanja govora koristeći poznati benchmark skup podataka *Librispeech*. Problematika je riješena korištenjem naprednih Transformerskih neuronskih mreža. Demonstrirana je visoka preciznost modela u prepoznavanju govora. Provedena je opsežna analiza pri čemu je korištena većina prezentiranih algoritama, uz primjenu 5 vizualnih reprezentacija audio signala. Rezultati su prikazani grafički i objašnjeni korištenjem mjera raznolikosti. Na kraju rada potvrđena je hipoteza da ansambli kada se pravilno koriste nadilaze standardne modele, bilo da se radi o Linearnoj regresiji, KNN-u ili Transformerskim neuronskim mrežama.

Summary

In this work, the focus was on methods of combining classifiers. Base algorithms were not described in detail but used as a foundation with the assumption of knowledge in the field of machine learning. Initially, several combiners, i.e., algorithms that create a synthesis among the outputs of multiple estimators, were explored. Then, the most commonly used algorithms in ensembles were presented and driven by the desire to create an optimal ensemble, key elements of a successful ensemble were described through the concept of diversity.

Subsequently, some of the presented methods were tested on the problem of speech recognition using the well-known benchmark dataset *Librispeech*. The problem is addressed using advanced Transformer Neural Networks, demonstrating high precision of the model in speech recognition. An extensive analysis was conducted, employing most of the presented algorithms along with five visual representations of audio signal. Results were graphically displayed and explained using measures of diversity. Ultimately, this work confirms the hypothesis that ensembles, when used correctly, surpasses standard models, whether it be Linear Regression, KNN, or Transformer Neural Network.

Životopis

Rođen sam 23. listopada 1998. u Zagrebu. Nakon završetka školovanja u Osnovnoj školi Sesvetski Kraljevec, nastavio sam obrazovanje u Tehničkoj školi Jelkovec, diplomiravši 2017. godine. Radi sklonosti prema matematici, upisao sam preddiplomski studij matematike (nastavnički smjer) na PMF-u u Zagrebu te ga završio u rujnu 2021. postavši time sveučilišni prvostupnik. Iste godine sam započeo diplomski studij Matematičke statistike na istome studiju. Tijekom drugoga semestra diplomskog studija pronašao sam svoj interes u području Umjetne inteligencije (AI) i usmjerio svoje daljnje obrazovanje u tom smjeru. Nedugo zatim, zaposlio sam se kao data analyst u Rimac korporaciji, radeći paralelno uz studij. U četvrtom semestru, kao član tima *Unsupervised Lemons*, osvojio sam Lumen 2023., najveće natjecanje iz područja Data Science u našoj regiji. Ovaj rad je dokaz mojeg entuzijastičnog pristupa području AI. Nakon završetka studija, planiram nastaviti karijeru u tom području.