MACHINE LEARNING
Learning Challenge

Professor:	G.A. Chrupala
Date:	14th December 2018

Group number:	31
András Ádám	U947345
Irene Font Peradejordi	U169795
Maria Soares Eira	U900574

1 | Description of the computational learning experiments

During a data exploration analysis, we discovered that there were 105835 files split into a training and test set. Each file consisted of a matrix with the dimensions: number of timeframes x 13 MFCC features. The Mel Frequency Cepstral Coefficient (MFCC) analysis is a standard procedure of feature extraction that precedes most of the automatic speech recognition systems and, for this reason, we decided to use them instead of extracting our own features from the raw audio files.

Regarding the time frame dimension, it was verified that the files had different lengths, ranging from 20 to 1999 time frames. After a deeper analysis, we realized that only 6 documents had more than 1750 frames and those outlier files were not in the training neither in the test set. Excluding these files, we concluded that the biggest file had 99 frames so we concatenated all the files in a single matrix and added zeros at the end for the shorter lengths. The result was two 3D matrices, one for the training and another for the test set, with the shape: number files x 99 time-frames x 13 features.

Besides the features data, we also investigated the labels data and we concluded that there were 35 different labels that correspond to 35 English words. The frequency of labels is well distributed among the files having on average 2710 examples per label. We have therefore a multiclass classification problem and we will make use of the most appropriate learning algorithms combined with the feature engineering techniques to create an efficient speech recognition system.

Feature engineering

The first approach of feature engineering consisted of reducing the time dimension by computing the mean, maximum, minimum and standard deviation for each MFCC feature (matrix shape: number of files x 13 features). After the application of some learning models - Perceptron, K-Nearest Neighbours (KNN), and Decision Tree, (DT) - on this reduced matrix we concluded that compressing the time of audio files would reduce the characterization of the audio waves which is an important component of time series signals. Therefore, the second approach was to compress the different features into a single one in order to keep the time evolution. We started by standardizing all the features with the z-score method and then we created a new matrix by the concatenation of mean, maximum, minimum and standard deviation of each feature (new matrix shape: number of files x (99 timeframes x 4)). We implemented the KNN algorithm on this matrix which resulted in a better accuracy in the validation set (from 39% to 51%). Finally, we decided to implement Neural Networks (NN) to improve our test performance and we used as input the 3D matrix containing all the features and all the time frames for each of the files without any kind of feature engineering. For every model, we randomly split the data into a training and validation set to be able to evaluate our algorithm on unseen data.

Learning algorithms

The first learning algorithm applied was the **Perceptron** but soon we realized that for 35 labels this model would not be the most adequate. The **Decision Tree** has also shown poor performances, so we moved to the **KNN** algorithm. By tuning the parameters of this model, we achieved a maximum accuracy of 50% on the validation set which resulted in 26% on the test set. Unsatisfied with this result we moved to a more ambitious algorithm exploring **NN** capabilities which proved to be the most appropriate approach for a task of this nature. Accuracies of these models and respective parameters tuning can be found in Annexes. Perceptron, KNN and DT algorithms implemented in this work belong to Sklearn library and the NN to Keras library.

Parameter tuning

Although parameter tuning had been performed on all the models, we will only detail the optimization performed for the final approach - NN - which was also the most challenging. Following the Backpropagation standards to create a NN architecture we defined an initial set of parameters: **learning rate** was specified to the default value (0.001) and the **activation function** to Rectified Linear Unit (relu) as recommended on the Keras documentation from TensorFlow[1]. The number of **epochs** was adjusted according to the learning rate in order to achieve the best combination of both. The number and size of **hidden layers** was the most complex task to solve. Inspired by Hal Daumé III book[2] we applied the theorem 10 that states that "two-layer networks are universal function approximators" and we started by designing a NN with a single hidden layer. The following key question was: how many hidden units should we have in a two-layer network? To solve that question we used the empirically-derived rule-of-thumb from Jeff Heaton[3] "the optimal size of the hidden layer is usually between the size of the input and size of the output layer". In our case, we had a matrix of 99x13 features, (input size of 1287), and an output layer of 35 labels. Therefore, we started by a two-layer network with a hidden layer of 612 units which is approximately a number in the middle of the input and output layer. The initial NN

parameters were progressively updated to improve our accuracy results and we achieved an impressive validation accuracy of 98%. However, after submission, this value dropped to 68% which could be due to an **overfitting problem**. To mitigate this possibility, we investigated regularization techniques for NN and we found the **"Drop Out" Keras parameter,** where randomly selected neurons are ignored during training. If neurons are randomly dropped out of the network during training, other neurons will have to step in and handle the representation required to make predictions for the missing neurons.[4] This is believed to result in multiple independent internal representations. We started by a recommended drop-out value of 20% which resulted in a 70.8% of accuracy on the test set. Although better, it represented only a 2.8% improvement compared to the previous result and it was still showing a significant gap to the validation accuracy of 91%. To eliminate the overfitting possibility, we increased the drop-out value to 0.4 which consists of ignoring 40% of the input units. In accordance with the authors of the article, we could also get a better performance by adding more hidden layers, since it gives the model more of an opportunity to learn independent representations. This new approach resulted in a validation accuracy of 96% leading to a final test accuracy of 76.5%. During this long process of fine tuning we realized that test accuracy was consistently around 70% even with very different approaches. We were curious about whether we were generating the same results with different methods or having different predictions but coincidently achieving the same accuracy. To check what was actually happening we decided to compare the obtained prediction vectors and we concluded that there was on average a similarity of 65% evidencing that the results were actually different. We decided then to apply a creative method of combining these predictions and pick the most likely outcome for each file. We used all the parameter combinations that we had uploaded before and we generated more results by varying the number of hidden layers, number of units per layer, drop out proportion, application of dropout on more than one layer and epochs. We believed that the more diverse, the more informative this approach could be because it would mitigate systematic errors. We used 15 different sets of parameters, (an odd number so that ties are better settled). With this innovative method, we obtained a final accuracy of 79.78% on the test set.

Discussion

The results obtained on all the tuning procedures revealed that we were probably overfitting the training data. We tried to mitigate this situation by decreasing the number of epochs and introducing the drop-out parameter. Although we believed these methods would be enough to reduce this situation, we were limited to 7 trials on the test set to evaluate the generalization power of our solution. Even if we had accuracy values of more than 90% for different methods this was never a sign that the performance on the test set would be better. With extra time and more trials to spare, the following methods could have been applied to improve the performance of the speech recognition system developed on this group project:

- Cross-validation: instead of limiting our evaluation process to the validation test set, we could have done several different subsets and compare the results.
- Deep Learning: although we are aware that this powerful algorithm can be very efficient to solve this kind of tasks, it was not covered on this course and we do not have the sensitivity to optimize the parameters of such a complex model.

To close, we are satisfied with the final performance of our speech recognition system that exceeds the baseline and made us learn more about the Perceptron, KNN, DT and NN models as well as the complexity of tuning its parameters to achieve a model with a high generalization power.

2 | Detailed specification of the work done by group members:

Andràs took the first initiative to start the data exploration. Maria and Irene detected the outliers and went deep into the data exploration. Maria and Irene implemented the first KNN model, compressing the time but achieving a poor performance. Irene tried an LDA method and the perceptron algorithm. Maria implemented the KNNs adding the zeros and concatenating different statistical measures. Andràs researched if extracting our own features would be doable. He concluded it would be too time-consuming and was not too promising about improving significantly the results. Andràs implemented the NN. Maria, Andràs, and Irene tunned the parameters until we run out of trials in CodaLab.

3 | Codalab group name: Group31

4 | References

1. *Train your first neural network: basic classification,* (2018), TensorFlow, www.tensorflow.org/tutorials/keras/basic_classification, visited on: 13-12-2018

2. Hal Daumé III, *A course in Machine Learning* (work in progress), http://ciml.info/ visited on: 14-12-2018

3. Jeff Heaton, *Introduction to Neural Networks in Java (2012*), www.heatonresearch.com/book/, visited on: 14-12-2018

4. Srivastava N. et al, (2014), *Dropout: A Simple Way to Prevent Neural Networks from Overfitting,* http://jmlr.org/papers/v15/srivastava14a.html, visited on: 13-12-2018

5 | Annex

Table of parameter setting for the **Perceptron**, **Decision Tree** and **K-Nearest Neighbor**

Model	**Data Processing methods**	**Parameters**	**Validation set Score**	**Test set Score**
Perceptron	Concatenation of the mean, stdev, max, min	Epoch: 5, 10,20,40	0.059	
Decision Tree	Concatenation of the mean, stdev, max, min	Depth: 80	0.182	
KNN	Time compression - Mean	Minkowski / N=2	0.326	
		Euclidean / N=2	0.326	
		Minkowski / N=1	0.386	
		Minkowski / N=10	0.295	
	Time compression - stdev	Minkowski / N=1	0.209	
	Time compression - Max	Minkowski / N=1	0.199	
	Time compression - Abs Max	Minkowski / N=1	0.157	
	Time compression - Min	Minkowski / N=1	0.233	
	Concatenation of the mean, stdev, max, min	Minkowski / N=1	0.509	0.26

Table of parameter setting for the **Neural Network**

Tuning parameters	**NN Architecture:** **Number and size of layers** **Drop out value (DO)**	**Validation set Score**	**Test set Score**
Learning rate: 0.001, Loss: sparse_categorical_crossentropy Epochs: 10	Layer 1: 128 Layer 2: 35	0.586	
	Layer 1: 128 Layer 2: 128 Layer 3: 35	0.783	
	Layer 1: 128 Layer 2: 128 Layer 3: 128 Layer 4: 35	0.827	
Learning rate: 0.001 Loss: sparse_categorical_crossentropy Epochs: 20	Layer 1: 128 Layer 2: 35	0.620	
	Layer 1: 128 Layer 2: 128 Layer 3: 35	0.821	
	Layer 1: 128 Layer 2: 128 Layer 3: 128 Layer 4: 35	0.853	

Tuning parameters	**NN Architecture:** **Number and size of layers** **Drop out value (DO)**	**Validation set Score**	**Test set Score**
Learning rate: 0.01 Loss: sparse_categorical_crossentropy Epochs: 20	Layer 1: 128 Layer 2: 128 Layer 3: 128 Layer 4: 35	0.03	
Learning rate: 0.0001 Loss: sparse_categorical_crossentropy Epochs: 20	Layer 1: 128 Layer 2: 128 Layer 3: 128 Layer 4: 35	0.939	
Learning rate: 0.0001 Loss: sparse_categorical_crossentropy Epochs: 30	Layer 1: 128 Layer 2: 128 Layer 3: 128 Layer 4: 35	0.973	
Learning rate: 0.0001 Loss: sparse_categorical_crossentropy Epochs: 20	Layer 1: 256 Layer 2: 128 Layer 3: 64 Layer 4: 35	0.959	
Learning rate: 0.0001 Loss: sparse_categorical_crossentropy Epochs: 30	Layer 1: 256 Layer 2: 128 Layer 3: 64 Layer 4: 35	0.982	68%
Learning rate: 0.001 Loss: sparse_categorical_crossentropy Epochs: 20	Layer 1: 256 Layer 2: 128 Layer 3: 64 Layer 4: 35	0.916	
Learning rate: 0.00001 Loss: sparse_categorical_crossentropy Epochs: 20	Layer 1: 256 Layer 2: 128 Layer 3: 64 Layer 4: 35	0.627	
Learning rate: 0.00001 Loss: sparse_categorical_crossentropy Epochs: 40	Layer 1: 256 Layer 2: 128 Layer 3: 64 Layer 4: 35	0.777	
Learning rate: 0.0005 Loss: sparse_categorical_crossentropy Epochs: 20	Layer 1: 612 Layer 2: 35	0.842	
Learning rate: 0.0003 Loss: sparse_categorical_crossentropy Epochs: 20	Layer 1: 612 Layer 2: 35	0.942	
Learning rate: 0.0001 Loss: sparse_categorical_crossentropy Epochs: 20	Layer 1: 612 Layer 2: 35	0.983	70.12%
Learning rate: 0.0005 Loss: sparse_categorical_crossentropy Epochs: 20	Layer 1: 250 Layer 2: 35	0.841	
Learning rate: 0.0008 Loss: sparse_categorical_crossentropy Epochs: 20	Layer 1: 250 Layer 2: 35	0.726	
Learning rate: default Loss: sparse_categorical_crossentropy Epochs: 10	Layer 1: 128 Layer 2: 35	0.592	
Learning rate: default Loss: sparse_categorical_crossentropy Epochs: 10	Layer 1: 99 Layer 2: 35	0.530	
Learning rate: 0.0001 Loss: sparse_categorical_crossentropy Epochs: 10	Layer 1: 612 DO: 0.2 Layer 2: 35	0.909	70.8%

Tuning parameters	**NN Architecture:** **Number and size of layers** **Drop out value (DO)**	**Validation set Score**	**Test set Score**
Learning rate: 0.0001 Loss: sparse_categorical_crossentropy Epochs: 10	Layer 1: 612 DO: 0.3 Layer 2: 35	0.883	
Learning rate: 0.0001 Loss: sparse_categorical_crossentropy Epochs: 10	Layer 1: 612 DO: 0.4 Layer 2: 35	0.845	
Learning rate: 0.0001 Loss: sparse_categorical_crossentropy Epochs: 20	Layer 1: 612 DO: 0.4 Layer 2: 35	0.917	
Learning rate: 0.0001 Loss: sparse_categorical_crossentropy Epochs: 20	Layer 1: 612 DO: 0.4 Layer 2: 612 Layer 3: 35	0.962	76.5%
Learning rate: 0.0001 Loss: sparse_categorical_crossentropy Epochs: 10	Layer 1: 612 DO: 0.4 Layer 2: 306 Layer 2: 35	0.874	
Learning rate: 0.0001 Loss: sparse_categorical_crossentropy Epochs: 10	Layer 1: 612 DO: 0.5 Layer 2: 35	0.805	
Learning rate: 0.0001 Loss: sparse_categorical_crossentropy Epochs: 12	Layer 1: 612 DO: 0.2 Layer 2: 35	0.927	
Learning rate: 0.0001 Loss: sparse_categorical_crossentropy Epochs: 20	Layer 1: 612 DO: 0.2 Layer 2: 35	0.965	
Learning rate: 0.0001 Loss: sparse_categorical_crossentropy Epochs: 10	Layer 1: 612 DO: 0.2 Layer 2: 612 DO:0.2 Layer 3: 612 Layer 3: 35	0.845	
Learning rate: 0.0001 Loss: sparse_categorical_crossentropy Epochs: 20	Layer 1: 612 DO: 0.2 Layer 2: 612 DO:0.2 Layer 3: 612 Layer 3: 35	0.917	
Learning rate: 0.0001 Loss: sparse_categorical_crossentropy Epochs: 10	Layer 1: 612 DO: 0.2 Layer 2: 612 Layer 3: 306 Layer 3: 35	0.952	
Learning rate: 0.0001 Loss: sparse_categorical_crossentropy Epochs: 10	Layer 1: 612 DO: 0.2 Layer 2: 612 DO: 0.2 Layer 3: 306 Layer 3: 35	0.923	
Final approach of averaging all the outcomes			**79.8%**