Details about our solution to the ML Challenge (91.77% accuracy) are described below.

Preprocessing and feature engineering

We used the available feature file containing the Mel-Frequency Cepstral Coefficients (MFCC) to build our model by appending each dimension to the object we wanted to create. After initial exploratory data analysis, we saw that the audio files differed in duration time so we had to normalize their length in order to use all features instead of reducing information by shortening longer files. To do so, we performed zero padding on the time-domain signal matrix. Through further analysis, we uncovered six observations that had the dimension of (1999, 13), and were neither in training nor test set. These observations were ignored, and the padding was performed taking into account the next biggest shape which was (99, 13) for some audios. Afterward, we created dictionary mapping paths of the audios to their corresponding two-dimensional feature matrix, and this dictionary was used to create the train and the test feature matrices (which were 3 dimensional). The features were scaled using StandardScale, and after that, the training set was split up keeping some part as validation set. It was ensured that the class distribution between training and validation set was identical using the stratify parameter (to avoid disproportionate abundance of some classes in the training set). We observed that there was a class imbalance, with some classes having almost twice the representation as another in the training data. Hence, while training our neural networks, we sometimes used the class weight parameter, to give greater attention to the minority classes while computing loss. Alternatively, we also tried upscaling the underrepresented classes in the feature matrix and feeding the modified training set into the neural network (without the class weight parameter). During the experimentation phase, new features were designed which included the delta MFCC and delta delta MFCC features. However, we didn't use them in the final version of our model, as we achieved better accuracy without them. The final version of our solution involves the usage of a convolution neural network (2D) (a decision we made after trying out numerous other classifiers).

Learning Algorithms & Parameter Tuning

According to Ou and Murphy (2007), the best classifier for a multiclass classification in speech recognition would be the neural network. However, we wanted to also include other possible classifiers as well (Suuny, Peter and Jacob, 2013). After first testing the Naives Bayes (0.3685%) and Decision Tree (0.0748%), we then proceeded with KNN (k=3,5,7,9,16,20, highest k=20, 61.80%). It was then followed by Support Vector Machines (59.38%). A random forest tree with PCA was also tested (40%). The first attempt with neural networks gave 65% before further tuning parameters. Multiple model trainings with different parameters were then performed to discover the best option. This included: tuning the learning rate between 0.03 to 0.0001, using optimizer functions like Adam, SGD and RMSProp, iterating across 20 to 200 epochs, using activation functions ReLu, LeakyRelu, Tanh and their combinations, and starting with 2 layers and experimenting with more. The train and test set were split 80:20. We then started building layers with 32, 64 and 128 units. We began with a smaller node size (to filter out the relevant information) and moved towards a larger node size at the end of the network (to focus more deeply on the relevant information obtained). On top of the input layer, we added LeakyReLu, Conv2D, Dropout(to reduce overfitting), MaxPooling2D and Flatten layers before adding the Dense layer with the softmax activation function to get a probability distribution that sums to one. In our current model, we have applied class weights on the y_train (due to unbalanced classes). To compile the model we used the categorical Cross-Entropy loss function which is used for multi-class classification.

Current performance:

Accuracy of 91.77% was achieved on the test data. Optimal parameters for the CNN include using LeakyReLu, Adam optimizer with the learning rate of 0.0004 and 100 epochs. The accuracy improved by around 8 percent just by decreasing the learning rate from 0.03 to 0.004. By adding different hidden layers and LeakyReLu throughout the layer build-up, the accuracy increased by an additional 4 percent.

Codalab account name: u848144
Group number: 16

Workload by member:

Giandrick Dabian, u465726

- Experimented with SVM and Naive Bayes
- Performed Random forest tree with PCA
- Hyperparameter tuning with SVM and Neural Network
- Brainstorming new approaches
- Finalizing the report

Isabella L. Dintinjana, u818451

- Experimented with KNN and Decision Tree
- Hyperparameter tuning with KNN and Neural Network
- Explored the audio files
- Brainstorming new approaches
- Writing the report

Aayushi Pandey, u848144

- Coded the complete implementation of the solution in python
- Performed the exploratory data analysis of the dataset (understanding the features supplied, identifying deviant observations etc.)
- Carried out preprocessing (creating training and test feature matrices with zero padding)
- Engineered new features (delta mfcc, delta delta mfcc) and implemented ways to handle class imbalance (eg- through upsampling, class weight parameter)
- Experimented with Random Forest and Neural Networks (ANN/CNN)
- Designed and performed hyperparameter tuning of the final CNN model
- Helped in finalizing the report

References:

Suuny, S., Peter, S. D., & Jacob, K. P. (2013). Performance of different classifiers in speech recognition. Int. J. Res. Eng. Technol, 2(4). Retrieved from: https://pdfs.semanticscholar.org/2e7f/a8cc746d2891a9fdfb7351ee782824dd2bd7.pdf

Ou, G., & Murphey, Y. L. (2007). Multi-class pattern classification using neural networks. Pattern Recognition, 40(1), 4-18., DOI: https://doi.org/10.1016/j.patcog.2006.04.041

Note:

The sources of the parts of code which we have borrowed are mentioned in the python file.