Machine Learning Project

Team 2 Topic 2 : Voice recognition

International Semester October 2024

List of students:

Student 1 : Abdellah H'Lali
Student 2 : Harish Arvind Selvakumar
Student 3 : Jeremy Toinon
Student 4: Nissanth Nadarassin
Student 5 : Ivan Arturo Grandi Flores

Task distribution:

Student 1: Introduction and State of the Art - Abdellah H'Lali

Student 2: Solutions Available in Scikit-Learn - Harish Arvind Selvakumar

Student 3: Selection of a Single Solution - Jeremy Toinon

Student 4: Dataset Preparation and Preprocessing (code) - Nissanth Nadarassin

Student 5: Model Implementation and Evaluation (code) - Ivan Arturo Grandi Flores

State of the art

Speech recognition, the technology used to convert human speech into text or commands that can be understood by a computer, has revolutionised many fields, including personal assistance. This discipline combines languages, artificial intelligence (AI) and digital signal processing to offer natural communication solutions between humans and machines.

History and development

The history of speech recognition dates back to the 1950s, with the development of the first systems capable of recognising a limited number of words, such as Bell Labs' “Audrey” system, designed to identify numbers. In the 1970s, Hidden Markov Models (HMM) algorithms significantly improved accuracy, introducing probabilistic bases for analysing speech signals.
With the emergence of microprocessors and databases in the 1980s and 1990s, speech recognition systems became more robust, recognising hundreds of words. Since the 2010s, advances in artificial neural networks, in particular recurrent neural networks (RNN) and convolutional neural networks (CNN), have marked a decisive turning point, making it possible to process large quantities of data and improve performance in the face of complex challenges.

Current state of technology

Today, speech recognition is based on advanced deep learning techniques. Models such as the Transformer and architectures such as BERT or OpenAI's Whisper use contextual and semantic analysis to transcribe speech with impressive accuracy. These systems are integrated into a variety of applications: voice assistants (like Siri and Alexa), automatic transcription, voice commands for connected devices, and sentiment analysis based on voice.
In addition, the rise of massive databases and distributed computing systems has made it possible to train more complex models capable of handling different languages, accents and acoustic conditions. Current systems often use hybrid architectures combining HMM and neural networks or adopt a purely deep network approach.

Recent advances and trends

- Deep Learning: Deep learning has led to significant gains, with models such as WaveNet for speech synthesis and Baidu's DeepSpeech for speech recognition.
- Recurrent Neural Networks (RNN): These networks, such as Long Short-Term Memory (LSTM), handle sequential data efficiently and are often used to process audio streams.
- Convolutional networks (CNN): Although often associated with image analysis, CNNs are used to extract spectral features from audio data.

- Multimodality and context: The integration of contextual data (visual, environmental) enriches speech recognition, as demonstrated by multimodal systems.

Persistent challenges

Despite these advances, a number of challenges remain:

1. Noise: Noise interference can degrade system performance.
2. Accents and linguistic diversity: Regional and personal variations make it difficult to generalise models.
3. Voice variability: Changes in pitch, intensity or differences between male, female and child voices pose adaptation problems.
4. Ethical issues and biases: The systems may reproduce biases linked to training data, raising questions of equity and inclusion.

Solutions Available in Scikit-Learn

Scikit-Learn is a robust machine learning library in Python that provides a wide range of algorithms for classification, regression, clustering, and more. However, it does **not** offer direct audio processing or feature extraction capabilities (like MFCCs or spectrogram generation). These tasks are typically handled by specialized libraries such as **Librosa** or **PyDub**. Once features are extracted using these libraries, Scikit-Learn excels in building and evaluating machine learning models using these features.

Relevant Features for Machine Learning Models

To build an effective voice recognition system, extract meaningful audio features using Librosa and feed them into Scikit-Learn models:

- **MFCCs (Mel-Frequency Cepstral Coefficients):** Capture the timbral aspects of audio and are widely used in speech and speaker recognition.
- **Delta Features**: Represent the rate of change of MFCCs over time, capturing dynamic variations in audio signals.
- **Delta-Delta Features**: Represent the acceleration of MFCCs (second-order derivative), providing additional insights into temporal dynamics.
- **Spectrograms:** Represent the frequency spectrum of audio signals over time.
- **Chroma Features:** Indicate the intensity of each of the 12 different pitch classes.

- **Zero-Crossing Rate:** Measures the rate at which the signal changes sign, useful for detecting noisy signals.
- **Spectral Centroid, Bandwidth, Rolloff:** Describe the shape of the audio spectrum.

Available Classifiers in Scikit-Learn for Voice Recognition

Building an effective voice recognition system involves selecting appropriate classifiers that can handle the nuances of audio data. Below is a comprehensive explanation of each classifier, including how they work, their advantages and limitations, and code examples demonstrating their implementation using Scikit-Learn.

1. Support Vector Machines (SVM)

Support Vector Machines (SVMs) are supervised learning models that classify data by finding an optimal hyperplane to separate classes, making them effective in high-dimensional spaces. They are versatile with different kernel functions (e.g., linear, RBF) and robust against overfitting. However, they can be computationally intensive for large datasets, and their performance depends on the choice of kernel and parameter tuning.

---

2. Random Forest

Random Forest is an ensemble learning method that builds multiple decision trees during training and predicts by taking the mode of their outputs, improving accuracy and controlling overfitting. It handles high-dimensional spaces and large datasets effectively, reduces overfitting through averaging, and provides estimates of feature importance. However, it can be less interpretable than single decision trees and requires careful hyperparameter tuning for optimal performance.

---

3. K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple, instance-based learning algorithm that classifies new cases based on similarity, assuming that similar data points exist in close proximity. It is easy to understand and implement, requires no assumptions about data distribution, and works well with small datasets. However, it can be computationally intensive for large datasets and is sensitive to the scale of data and irrelevant features.

---

4. Multilayer Perceptron (MLPClassifier)

MLPClassifier is a feedforward artificial neural network with multiple layers of nodes, where each node (neuron) uses nonlinear activation functions and is fully connected to the next layer. It excels at learning complex patterns, supports various activation functions and optimizers, and is well-suited for large datasets. However, it requires careful hyperparameter tuning, is prone to overfitting without proper regularization, and can be time-consuming to train.

---

5. Gaussian Naive Bayes

Gaussian Naive Bayes is a variant of the Naive Bayes algorithm designed for continuous data, assuming that features follow a normal (Gaussian) distribution. It is fast, easy to implement, performs well with high-dimensional data, and requires minimal training data. However, it relies on a strong assumption of feature independence and is less flexible, making it unable to capture complex relationships between features.

---

6. Gradient Boosting Classifiers (HistGradientBoostingClassifier)

Gradient Boosting is a machine learning technique for regression and classification that builds models incrementally in a stage-wise fashion, optimizing an arbitrary differentiable loss function. The HistGradientBoostingClassifier, a Scikit-Learn implementation, is specifically optimized for large datasets. It offers high predictive accuracy, handles mixed data types effectively, and is efficient with large datasets. However, it can overfit if not properly tuned and typically requires longer training times compared to simpler models.

Comparative Evaluation

After running all classifiers, compare their performance to select the best model for your application.

Visualization of Results**Plotting Confusion Matrices**

Visualize confusion matrices for deeper insights into each classifier's performance by applying the `plot_confusion_matrix` function with the respective `y_pred`.

Hyperparameter Tuning

Improving model performance through hyperparameter tuning.

1. **Hyperparameter Tuning for KNN**: Optimize key parameters like the number of neighbors (`n_neighbors`), distance metric (`metric`), and weighting method (`weights`) to improve classification performance.

2. **Hyperparameter Tuning for MLPClassifier**: Adjust parameters such as hidden layer sizes (`hidden_layer_sizes`), activation function (`activation`), and solver (`solver`) to enhance learning and prevent overfitting.

Generalized Workflow for Machine Learning Tasks

- **Feature Extraction:** Extract meaningful features from raw data, such as `MFCCs, chroma features`, or `spectral contrast`. Specialized libraries like `Librosa` tools are often used.(`Scikit-Learn` itself does not provide audio processing tools or methods for feature extraction)
- **Data Preparation:** Organize data by relevant categories, extract features, and encode labels using methods like `LabelEncoder` for compatibility with machine learning models.
- **Data Splitting:** Split the dataset into `training` and `testing sets` to evaluate model performance.
- **Scaling:** Apply scaling techniques like `StandardScaler` to standardize features and improve model performance.
- **Classification:** Train models such as `SVM, Random Forest, KNN, MLPClassifier`, or `Gradient Boosting` depending on the task.
- **Evaluation:** Measure model performance using metrics like `accuracy`, `precision`, `recall`, and `F1-score` to assess effectiveness.
- **Visualization of Results:** Use visual tools such as `confusion matrices` or performance comparison charts to analyze errors and evaluate classifiers.

- **Hyperparameter Tuning:** Optimize model performance with tools like `GridSearchCV`, refining parameters for models like `SVM`, `KNN`, and `MLPClassifier`.

Final Considerations:

- **Model Selection**: Choose the classifier that offers the best trade-off between accuracy and computational efficiency for your specific use case.
- **Feature Engineering**: Experiment with different features and combinations to improve model performance.
- **Cross-Validation**: Use cross-validation to ensure your model generalizes well to unseen data.
- **Ensemble Methods**: Consider combining models (e.g., VotingClassifier) to leverage the strengths of multiple classifiers.

Selection of a single solution

In speech recognition, the choice of a machine learning model depends on several factors: accuracy, efficiency, implementation complexity and robustness to data variations. Two main models were considered in this project: KNN (K-Nearest Neighbors) and MLP (Multi-Layer Perceptron). T

Comparative Model Analysis (KNN vs MLP)

Accuracy :

KNN: Performs well on well-preprocessed data with little noise.
MLP: More robust, capable of capturing complex relationships in noisy or non-linear data.

Implementation complexity:

KNN: Very simple to implement, requires no training.
MLP: Requires a training phase with adjustable parameters (layer size, activation functions).

Training and inference time :

KNN: Very short training time, but slow predictions for large datasets.
MLP: Long training time, but fast predictions.

To illustrate that some Data about training depending on features:

<u>Comparison Tables of KNN and MLP</u>

For 20 features :

	KNN	MLP
Time of training	-1 min	-1 min
Accuracy	72%	73%

For 200 features :

	KNN	MLP
Time of training	1 min	5 min
Accuracy	76%	87%

For 1248 features :

	KNN	MLP
Time of training	5 min	25 min
Accuracy	86%	87%

To make the most of our choice of number of features, we used the algorithm grid_search. That allows us to use those two models with the best efficiency.

As we can see from these comparative tables, accuracy increases with the number of features. For example, with 1248 features, the accuracy is very high. However, this high accuracy may be due to overfitting, which is why we consider the performance of the MLP model with 200 features to be the best of all our tests.

Robustness and scalability:

KNN: Sensitive to data dimensionality, may require feature reduction (PCA).
MLP: More scalable and suitable for complex data.

In the test phase we tested many dimensions/options, as a result of which we noticed that MLP was always above KNN on all tests performed.

Why include KNN ?

Simplicity and speed:
KNN is ideal for rapid prototyping and well-preprocessed data.

Intuitive interpretation:
Easy to understand and explain, making it useful for academic or educational projects.

Why include MLP ?

High performance:
MLP performs best on complex or noisy datasets.

Flexibility:
Adjustable hyperparameters let you optimize the model for specific cases.

Complementarity of the two models

In this voice recognition project, the combined use of MLP (Multi-Layer Perceptron) and KNN (K-Nearest Neighbors) leverages the strengths of both models to create a comprehensive and efficient solution. KNN, with its simplicity and ease of implementation, serves as an excellent baseline to validate initial steps, such as feature extraction (MFCCs) and rapid evaluation of data quality. On the other hand, MLP, with its ability to model complex relationships and its robustness against noisy data, enhances the final performance on diverse vocal inputs. Together, these models offer an ideal combination for both quick prototyping and optimizing the system's overall

accuracy, while remaining adaptable to the project's specific requirements.

Why KNN and MLP and not others

There are a number of models that could have been used to carry out this project, such as Random forest, SVM or core Gaussian Naive Bayes. However, when testing the latter, with different numbers of features, we noticed that on average, they were less accurate than KNN and MLP by around 10% each time. Thanks to our classroom work, our familiarity with KNN was essential to the success of this project. That's why KNN and MLP were the obvious choice for our group.

Preprocessing of audio files

Audio data inherently contains variations in length, amplitude, and noise levels, making it unsuitable for direct input into machine learning models. Preprocessing standardizes and structures the data by extracting meaningful features, reducing dimensionality, and ensuring compatibility with model requirements. The steps below highlight how this is achieved.

Handling Variable Audio File Lengths
Audio files often vary in length, leading to inconsistent feature dimensions after extraction.
Solution:

- Truncation: If the audio exceeds a fixed length, it is truncated to a predefined duration.
- Padding: If the audio is shorter than the target length, zeros are appended to achieve consistency.
- Target Frames: For this project, the target number of frames was set to 32, ensuring all feature vectors had a consistent length of 39*32=1248.

Feature Extraction

Feature extraction transforms raw audio into representations that capture its essential characteristics while discarding redundant information.
The features used in this project are:

1.MFCC (Mel-Frequency Cepstral Coefficients):

Captures the spectral properties of the audio that correlate with human perception of sound.
Process:
Compute the Short-Time Fourier Transform (STFT) of the signal.Apply a Mel filter bank to emphasize frequencies relevant to human hearing.Take the logarithm of the energy spectrum to mimic the human auditory system response. Apply the Discrete Cosine Transform (DCT) to decorrelate features and reduce dimensionality.
Output: 13 coefficients representing the audio's frequency content.

2.Delta Features:

Capture the temporal dynamics of the MFCCs by calculating their first derivative over time.
Significance: Helps models identify patterns related to how audio features change over time, such as transitions between phonemes.

3.Delta-Delta Features:

Provide higher-order temporal information by calculating the second derivative of the MFCCs.
Significance: Further enhances the model's ability to capture temporal variations in the audio signal.

Final Feature Matrix:
All three feature types (MFCC, Delta, and Delta-Delta) are combined to create a feature matrix of size 39*T,where T is the number of frames.

Padding and Flattening
Padding:
Ensures all feature matrices are the same size, addressing variability in audio length.
How: Zero-padding is applied to shorter matrices to reach the target frame count.
Flattening:
Converts the 2D feature matrix into a 1D vector for input into machine learning models.
How: The feature matrix of size 39*32 is flattened into a vector of length 1248.
Dimensionality Reduction Using PCA

Why PCA?
High-dimensional data can lead to overfitting and increased computational cost. Principal Component Analysis (PCA) reduces dimensionality while retaining the most important information.
Example: Reducing 1248 features to 200 components significantly simplifies the data without sacrificing critical information.

How PCA Works:

- Standardization: Features are standardized to have zero mean and unit variance to ensure fair treatment of all dimensions.
- Covariance Matrix: PCA computes the covariance matrix to understand relationships between features.
- Eigen Decomposition: Eigenvalues and eigenvectors of the covariance matrix are computed. The eigenvectors represent the principal components.
- Projection: Data is projected onto the top k principal components, where k is the desired number of dimensions (e.g., 200 in this project).

Impact:

- Reduces noise and redundancy.
- Prevents overfitting by limiting the model's ability to memorize spurious patterns in the data.

Conclusion

To conclude the project we selected the MLP classification as the best one because it got the best results and with not a lot of dimensions to process. The final result is 0.87 of accuracy with 200 features for this model. To follow the procedure of the competition we created a csv file where we assign labels of the test set but it is not too useful as we don't have the labels of it. If we want to validate the results in the testing set that has around 150000 audio files, we must listen to the audio manually and compare with the labels assigned with the model.

This project has helped us increase our understanding about supervised learning and classification tasks. A great thing about doing this project is that we faced realistic tasks that companies work on like SIRI or ALEXA that recognize voice commands. Because of that we thank AGH and EFREI Paris, and we hope to do similar or more challenging projects in the future.

Thank you

[Read the README.txt file if you want to run the training models]