Classification of aligned 16S DNA Sequences

Marco Kreuzer

2023-07-26

Contents

1 Introduction

Rapid advancements in genomic sequencing techniques have resulted in an exponential increase in the availability of biological sequence data, specifically 16S ribosomal DNA (rRNA) sequences. The 16S rRNA sequences, universally present in bacteria and archaea, are used as a fundamental tool for microbial ecology studies, including identification, classification, and phylogenetic analysis. However, the massive influx of 16S rRNA sequences demands more efficient and accurate computational tools for analysis, interpretation, and understanding.

Traditional analysis methods, such as alignment-based techniques, may become computationally expensive and potentially less accurate as the sequence data size increases. Recent years have witnessed the rise of machine learning, and more specifically, deep learning techniques, which have been successfully applied in various fields, including image recognition, natural language processing, and genomics. The key advantage of deep learning is its ability to learn complex patterns in high-dimensional data, thus making it an attractive option for 16S rDNA sequence analysis.

The 16S ribosomal RNA (rRNA) gene, typically found in the genome of bacteria and archaea, is a highly conserved component of the 30S small subunit of prokaryotic ribosomes. Its function is pivotal to the process of protein synthesis within the cell. This gene encodes for the 16S rRNA molecule, which forms an integral part of the ribosomal machinery used for translating mRNA into proteins. The significance of the 16S rRNA gene in scientific research extends beyond its biological function. It has become a crucial tool for phylogenetic studies and taxonomic classification of bacteria and archaea due to its high conservation levels and the presence of variable regions within the gene. The 16S rRNA gene contains nine hypervariable regions (V1-V9) that show substantial diversity among different bacterial and archaeal species. These variable regions are interspersed among conserved sequences. By sequencing 16S rRNA genes, researchers can identify and classify the microbial species present in a sample, making it a standard method in microbial ecology, metagenomics, and microbiome studies.

DNA Sequencing technologies have significantly improved in the past decade. While it was common practice to only sequence one of the nine variable regions of the 16S rRNA gene, it now has become possible to sequence the whole region.

In this report, deep learning techniques are for the classification of 16S rRNA genes are explored. More generally, it provides a framework for data engineering and classifying DNA sequences of any DNA alignment.

1. Description of the data
2. Data engineering
3. Confolutional Classifier
4. Variational Autoencoder
5. Discussion and Outlook

2 Data description and engineering

DNA sequences are typically stored in a `.fasta` file format. Here is an example of a single sequence in this format:

```
>sequence_id1
ATGCCTT
```

A DNA alignment refers to a method by which multiple DNA sequences are arranged to maximize the similarity between corresponding nucleotides at each position. This alignment therefore identifies regions of similarity, providing insights into the functional, structural, or evolutionary relationships between the sequences.

An example of a DNA alignment represented in `.fasta` format could look like this:

```
>sequence_id1
ATGCCTT-GGCA-AGCTTGG
>sequence_id2
ATGC-ATTGGCATAAG-TGG
>sequence_id3
ATGCGTTGG-ATAAGCTTGG
>sequence_id4
ATGC-CTTGGCAT-AG-T-G
```

In this alignment, DNA sequences from four different organims are compared. The '-' represents gaps inserted to maximize the sequence similarity at each position. The comparison highlights the conserved nucleotides (like 'ATGC' at the start of all sequences) and the variable positions (such as the fifth and seventh nucleotides).

2.1 Data source

For this project, the SILVA database (https://www.arb-silva.de/) was used. This is a comprehensive resource that provides quality checked, and regularly curated datasets of aligned small (16S/18S, SSU) and large subunit (23S/28S, LSU) ribosomal RNA (rRNA) sequences for all three domains of life (Bacteria, Archaea, and Eukarya). In this study, an aligned version of reference sequences of the SSU (https://ftp.arb-silva.de/current/Exports/SILVA_138.1_SSURef_tax_silva_full_align_trunc.fasta.gz) was used. A crucial aspect of the SILVA database is that it includes hierarchical taxonomic information for each sequence in the sequence header. An example of a sequence header is given below:

```
>HG530070.1.1349 Bacteria;Actinobacteriota;Actinobacteria;Actinomycetales;
Actinomycetaceae;Trueperella;Trueperella pyogenes
```

Which corresponds to the following taxonomic levels:

```
>ncbi_identifier Domain;Kingdom;Phylum;Order;Family;Genus;Species
```

2.2 Data description

For this project, the dataset was subset to only include Bacteria and consists of ~1.9M sequences. The 16S sequence is typically 1500 base pairs (1.5 kb) long. Since a very diverse set of organisms are included in the data set, the alignment contains large amounts of gaps. Therefore, the total length of the alignment is 50000 base pairs long. The frequencies of bases are A: 0.73%, T: 0.6%, G: 0.91%, C: 0.66%. The remainder of poistions consists of gaps.

2.2.1 Sequence Taxonomy

Each sequence contains hierarchical taxonomic information as described above. However, many sequences do not contain all eight levels and would have to be curated manually. Therefore, the sequences were filtered to only include cases where the full taxonomy is known. This resulted in a dataset of 1788512 sequences (~1.7 M). Since machine learning classification tasks require to have multiple samples per class, the classes were filtered to include a minimum amount of samples per class. The number of unique classes per taxonomic level are given in Table 1.

Table 1: Number of classes within the domain Bacteria given a classification level. Min 1, Min 10 and Min 20 describe the number of classes that remain when each class has a minimum of 1,10 or 20 sequences.

Classification Level	Min 1	Min 10	Min 20
Kingdom	46	43	41
Phylum	114	110	103
Order	277	267	255
Family	582	558	525
Genus	3259	2520	2075
Species	151880	3947	1812

The taxonomic classes are highly unbalanced at every level in terms of members per class (see Figure 1).

2.3 Data Engineering and Processing

2.3.1 One-hot encoding

DNA sequences are represented as strings of nucleotides (A,T,C,G). In the context of deep learning, this representation has to be one-hot encoded. Furthermore, gaps were treated as follows: If the gap is larger than four consecutive positions, the positions were encoded as `N` (missing data), otherwise the data was encoded as `-`. Therefore, a nucleotide can be encoded into six values (e.g. `[0,1,0,0,0,0]`)

This task has been achieved with the custom python class `hot_dna`. To instantiate the class, a DNA sequence and the taxonomic description as described above have to be supplied. The methods of `hot_dna` can be used to encode and decode a one-hot encoded DNA sequence.

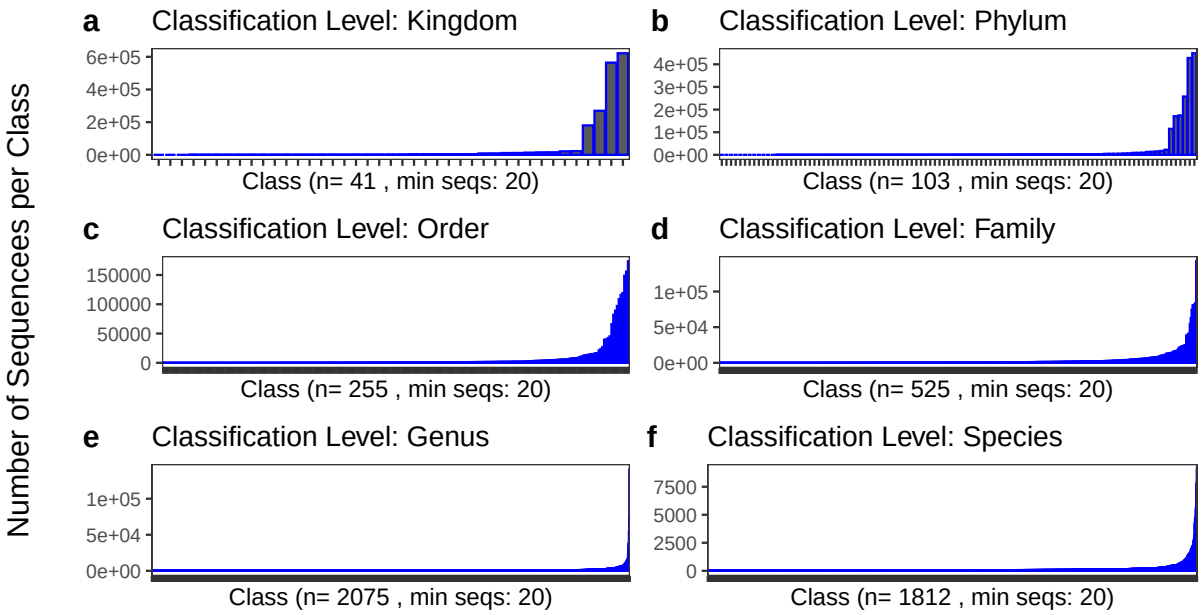

Figure 1: The datasets are highly imbalanced. **a-f**: The number of sequences in each of the classes at a specific taxonomic level. In all cases, only classes with more than 20 sequences were retained.

```python
class hot_dna:
    ### Class for One Hot Encoding DNA sequences
    def __init__(self, sequence, taxonomy):
        sequence = sequence.upper()
        self.sequence = self._preprocess_sequence(sequence)
        self.category_mapping = {'A': 0, 'C': 1, 'G': 2, 'T': 3, 'U': 3, '-': 4, 'N': 5}
        if sequence:
            self.onehot = self._onehot_encode(self.sequence)
        self.taxonomy = taxonomy.split(';')  # splitting by ';' to get each taxonomy level

    def _preprocess_sequence(self, sequence):
        ambiguous_bases = {'R', 'Y', 'S', 'W', 'K', 'M', 'B', 'D', 'H', 'V', '.',}
        new_sequence = ""
        for base in sequence:
            if base in ambiguous_bases:
                new_sequence += 'N'
            else:
                new_sequence += base
        # replace sequences of four or more '-' characters with 'N' characters
        new_sequence = re.sub('(-{4,})', lambda m: 'N' * len(m.group(1)), new_sequence)
        return new_sequence

    def _onehot_encode(self, sequence):
        integer_encoded = np.array([self.category_mapping[char] for char in sequence]).reshape(-1, 1)
        onehot_encoder = OneHotEncoder(sparse=False, categories='auto', handle_unknown='ignore')
        onehot_encoded = onehot_encoder.fit_transform(integer_encoded)

        # Fill missing channels with zeros
        full_onehot_encoded = np.zeros((len(sequence), 6))
```

```
        full_onehot_encoded[:, :onehot_encoded.shape[1]] = onehot_encoded

        return full_onehot_encoded

    def _onehot_decode(self, onehot_encoded):
        # Reverse the mapping dictionary
        reverse_category_mapping = {v: k for k, v in self.category_mapping.items()}
        # Convert one-hot encoding back to integer encoding
        integer_encoded = np.argmax(onehot_encoded, axis=1)
        # Convert integer encoding back to original sequence
        original_sequence = "".join(reverse_category_mapping[i.item()] for i in integer_encoded)
        return original_sequence
```

2.3.2 Processing large DNA alignments

In order to handle large DNA alignments, a strategy that saves RAM had to be applied. The basic idea was to read one sequence at a time, assign an index, and one-hot encode the DNA string. The results were then saved in a dictionary

```
{'sequence_id': '0', 'sequence_tensor': tensor([[0., 0., 0., 0., 0., 1.],
        [0., 0., 0., 0., 0., 1.],
        [0., 0., 0., 0., 0., 1.],
        ...,
        [0., 0., 0., 0., 0., 1.],
        [0., 0., 0., 0., 0., 1.],
        [0., 0., 0., 0., 0., 1.]])}
```