Challenges and Limitations of Maximum Likelihood Estimation (MLE) in Speech Data Analysis

Objective: Utilizing Maximum Likelihood Estimation (MLE) for Parameter Estimation in Speech Data Analysis

The objective is to calculate the coefficients αij and Aij for each pair of Ri and Sj by fitting all available data into a linear equation of the form FRi = αijFSj + Aij(αij − 1) using linear regression techniques. This approach incorporates MLE principles to optimize parameter estimation, particularly in scenarios involving missing values. The analysis will be conducted using the Peterson-Barney and Hillenbrand vowel datasets, aiming to enhance our understanding of speech characteristics and improve modeling accuracy.

Motivation Behind Linear Regression Approach for Comparing Formant Frequencies in Speech Analysis

We're using this equation for linear regression to find a simple way to compare formant frequencies between speakers. The equation is designed to easily show differences between genders: when speakers have similar genders, a value close to 1 for αij indicates a strong correlation in their formant frequencies. But if they have different genders, αij will be farther from 1, helping us identify distinct speech patterns. The Aij term adds more detail, capturing variations beyond gender. By applying this equation with linear regression, we aim to better understand how speakers' formant frequencies relate and improve our methods for analyzing speech.

Hillenbrand Speech Data

Data Pre Processing:

Adding Columns to ID by Sex

```
for x in range(df.shape[0]):
    if df.loc[x, 'ID'].startswith('m'):
        df.loc[x, 'Sex'] = 'male'
    elif df.loc[x, 'ID'].startswith('w'):
        df.loc[x, 'Sex'] = 'female'
    elif df.loc[x, 'ID'].startswith('b'):
        df.loc[x, 'Sex'] = 'boy'
    elif df.loc[x, 'ID'].startswith('g'):
        df.loc[x, 'Sex'] = 'girl'
```

And similar thing is done for vowels.

```
for x in range(df.shape[0]):
    if df.loc[x, 'ID'].endswith('ae'):
        df.loc[x, 'Vowel'] = 'ae'
    elif df.loc[x, 'ID'].endswith('ah'):
        df.loc[x, 'Vowel'] = 'ah'
    elif df.loc[x, 'ID'].endswith('aw'):
        df.loc[x, 'Vowel'] = 'aw'
    elif df.loc[x, 'ID'].endswith('eh'):
        df.loc[x, 'Vowel'] = 'eh'
    elif df.loc[x, 'ID'].endswith('er'):
        df.loc[x, 'Vowel'] = 'er'
    elif df.loc[x, 'ID'].endswith('ei'):
        df.loc[x, 'Vowel'] = 'ei'
    elif df.loc[x, 'ID'].endswith('ih'):
        df.loc[x, 'Vowel'] = 'ih'
    elif df.loc[x, 'ID'].endswith('iy'):
        df.loc[x, 'Vowel'] = 'iy'
    elif df.loc[x, 'ID'].endswith('oa'):
        df.loc[x, 'Vowel'] = 'oa'
    elif df.loc[x, 'ID'].endswith('oo'):
        df.loc[x, 'Vowel'] = 'oo'
    elif df.loc[x, 'ID'].endswith('uh'):
        df.loc[x, 'Vowel'] = 'uh'
    elif df.loc[x, 'ID'].endswith('uw'):
        df.loc[x, 'Vowel'] = 'uw'
```

Each row in our dataset corresponds to an individual's vowel speech data, containing information such as duration and formant frequencies at various percentiles. The ID for each entry follows a specific format: it begins with the individual's gender, followed by a gender-specific ID, and ends with the vowel identifier. To facilitate analysis, we are parsing the ID format and separating its components into distinct columns. This segmentation allows us to organize the data efficiently, enabling further exploration and interpretation.

Handling Missing Values:

```
missing_ids = set()

for index, row in df.iterrows():
    if any(pd.isnull(row)) or any(row == 0):
        missing_ids.add(row['ID'])

print("Missing IDs:")
print(list(missing_ids))
```

```
Missing IDs:
['m39', 'w34', 'm17', 'w28', 'b15', 'b17', 'w35', 'm47', 'g07', 'w21', 'b21', 'm31', 'm38', 'w15', 'w33', 'w47', 'w0
2', 'g02', 'w13', 'b08', 'g18', 'w30', 'b14', 'w08', 'w39', 'g14', 'm02', 'g19', 'w38', 'm34', 'w42', 'm44', 'b05', 'm
37', 'm25', 'w44', 'w07', 'w09', 'w50', 'g10', 'b01', 'g09', 'w27', 'm21', 'w46', 'm49', 'm20', 'g08', 'm19', 'b25',
'm28', 'm11', 'b23', 'g15', 'w41', 'b02', 'm35', 'w40', 'g21', 'w14', 'm42', 'm29', 'm04', 'w31', 'm50', 'g05', 'w17',
'w26', 'm32', 'w12', 'm09', 'g04', 'b19', 'm03', 'm01', 'm26', 'w06', 'm48', 'm10', 'm22', 'b09', 'b12', 'w11', 'm14',
'w05', 'b26', 'w32', 'm07', 'w29', 'g20', 'g13', 'b16', 'b04', 'w49', 'w25', 'w37', 'b22', 'w03', 'b27', 'w20', 'm45',
'g11', 'g12', 'b24', 'b03', 'm13', 'w45', 'b29', 'w01']
```

We've noticed a notable pattern: approximately 93 rows contain at least one missing value. This observation holds significant weight in our analysis, as discarding the data from these 93 individuals would lead to a substantial loss of valuable information. Hence, it becomes imperative to address the issue of missing values in a thoughtful manner, ensuring that we retain as much data as possible for meaningful insights and accurate modeling.

```
for col in df.columns:
    if col.startswith('F'):
        # Group by sex and vowel, calculate mean, and replace zero with mean
        df[col] = df.groupby(['Sex', 'Vowel'])[col].transform(lambda x: round(x.replace(0, x[x != 0].mean())))
```

In this section of the code, any missing values encountered are addressed through imputation. Specifically, if a missing value is detected, it is replaced with the average formant frequency value corresponding to the same sex and vowel category. Notably, data points with a value of 0 are excluded from this calculation to ensure accuracy in the imputation process. This approach aims to maintain the integrity of the dataset by substituting missing values with contextually relevant estimates derived from similar instances within the dataset.

Data Modification and MLE Calculation:

Segregating frequency column names into a list and Concatenating Speaker Data into Single Arrays for Linear Regression

```
frequencies = [col for col in df.columns if col.startswith('F')]
```

```
import numpy as np

# Get unique speaker IDs
unique_ids = df['ID'].unique()

# Initialize an empty list to store concatenated arrays for each speaker
all_speaker_arrays = []

# Iterate over unique speaker IDs
for speaker_id in unique_ids:
    # Select data for the current speaker ID
    speaker_data = df[df['ID'] == speaker_id]

    # Initialize an empty list to store frequencies for the current speaker
    speaker_frequencies = []

    # Iterate over frequencies
    for freq in frequencies:
        # Append frequencies for the current frequency to the list
        speaker_frequencies.extend(speaker_data[freq].values)

    # Convert the list of frequencies to a NumPy array
    speaker_array = np.array(speaker_frequencies)

    # Store the concatenated array for the current speaker ID
    all_speaker_arrays.append(speaker_array)
```

Using the linear regression module, we aim to fit the data of each pair of speakers into a linear equation.

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from tqdm import tqdm

# Initialize lists to store data
speaker_ids = []
alpha_ij_values = []
A_ij_values = []

# Iterate over unique combinations of speaker IDs
for i, speaker_id_i in enumerate(tqdm(unique_ids, desc="Calculating Coefficients")):
    for j, speaker_id_j in enumerate(unique_ids):
        if i != j:
            # Select arrays for the current pair of speaker IDs
            array_i = all_speaker_arrays[i].reshape(-1, 1)
            array_j = all_speaker_arrays[j].reshape(-1, 1)

            # Fit a linear regression model
            reg = LinearRegression()
            reg.fit(array_j, array_i)

            # Extract coefficients
            alpha_ij = reg.coef_[0][0]
            intercept = reg.intercept_[0]
            A_ij = intercept / (alpha_ij - 1)

            # Store coefficients for the current pair of speaker IDs
            speaker_ids.append((speaker_id_i, speaker_id_j))
            alpha_ij_values.append(alpha_ij)
            A_ij_values.append(A_ij)

# Create a DataFrame
coefficients_df = pd.DataFrame({
    'Speaker ID (Ri)': [speaker_id[0] for speaker_id in speaker_ids],
    'Speaker ID (Sj)': [speaker_id[1] for speaker_id in speaker_ids],
    'Alpha_ij': alpha_ij_values,
    'A_ij': A_ij_values
})

# Print the coefficients DataFrame
print("Matrix for A_ij and alpha_ij:")
print(coefficients_df)
```

We fitted the data of each pair of speakers into the linear equation FRi = αijFSj + Aij(αij − 1). Observations reveal that the value of Aij tends to exhibit larger positive or negative values when comparing male speakers to male speakers, as well as when comparing female speakers to female speakers, compared to comparisons between male and female speakers.

We're segregating data for each speaker who has a higher absolute value of Aij compared to the other speaker, and then keeping them in a new data frame. Specifically, we're considering values of Aij with a magnitude greater than 1000 as a reference for higher values.

```
# Filter coefficients DataFrame for the first speaker ID
first_speaker_id = unique_ids[0]
first_speaker_coefficients = coefficients_df[coefficients_df['Speaker ID (Ri)'] == first_speaker_id]

for index, row in first_speaker_coefficients.iterrows():
    print(f"A_ij for Speaker ID {row['Speaker ID (Sj)']}: {row['A_ij']}")
```

```
ndf = coefficients_df[(coefficients_df['A_ij']<-1000) | (coefficients_df['A_ij']>1000) ]

ndf.head()
```

We're creating a data frame to display, for each speaker, the count of male and female speakers with an absolute Aij value greater than 1000.

```
import pandas as pd

# Group by 'Speaker ID (Ri)'
grouped_df = ndf.groupby('Speaker ID (Ri)')

# Initialize lists to store results
speaker_ids = []
male_counts = []
female_counts = []

# Iterate over each group
for speaker_id, group in grouped_df:
    male_count = group[group['gender'] == 'm']['Speaker ID (Sj)'].nunique()
    female_count = group[group['gender'] == 'w']['Speaker ID (Sj)'].nunique()

    # Append results to lists
    speaker_ids.append(speaker_id)
    male_counts.append(male_count)
    female_counts.append(female_count)

# Create a new DataFrame to store the results
result_df = pd.DataFrame({
    'Speaker ID': speaker_ids,
    'Male Count': male_counts,
    'Female Count': female_counts
})

result_df
```

Calculating average male and female count for each gender with Aij having an extreme value

```
# Calculate average male count for IDs starting with "M"
avg_male_count_m = round(result_df[result_df['Speaker ID'].str.startswith('m')]['Male Count'].mean())

# Calculate average female count for IDs starting with "M"
avg_female_count_m = round(result_df[result_df['Speaker ID'].str.startswith('m')]['Female Count'].mean())

# Calculate average female count for IDs starting with "W"
avg_female_count_w = round(result_df[result_df['Speaker ID'].str.startswith('w')]['Female Count'].mean())

# Calculate average male count for IDs starting with "W"
avg_male_count_w = round(result_df[result_df['Speaker ID'].str.startswith('w')]['Male Count'].mean())

print(f"Average male count for IDs starting with 'M': {avg_male_count_m}")
print(f"Average female count for IDs starting with 'M': {avg_female_count_m}")
print(f"Average female count for IDs starting with 'W': {avg_female_count_w}")
print(f"Average male count for IDs starting with 'W': {avg_male_count_w}")
```

Average male count for IDs starting with 'M': 18

Average female count for IDs starting with 'M': 2

Average female count for IDs starting with 'W': 23

Average male count for IDs starting with 'W': 6

It's evident from the data that there's a substantial difference in the average counts between same-gender pairs and opposite-gender pairs with higher values of Aij. Specifically, there's a notably higher average count of individuals of the same gender compared to the opposite gender in the category of speakers with higher Aij values.

Peterson & Barney Speech Data

For the Peterson-Barney dataset, the overall process remains largely unchanged from the previous dataset (Hillenbrand). However, there are notable differences:

1. ***No Missing Values: Unlike the Hillenbrand vowel speech data, the Peterson-Barney dataset doesn't contain any missing values. Therefore, there's no need for additional handling of missing data.***
2. ***No Formant Frequencies at Percentiles: In contrast to the Hillenbrand data, the Peterson-Barney dataset doesn't include columns for formant frequencies at different percentiles. Only the formant frequencies themselves are provided.***
3. ***Exclusion of Duration Data: Duration data is not considered in the Peterson-Barney dataset.***

Despite these differences, the overall procedure for processing and analyzing the Peterson-Barney vowel speech data remains largely the same as outlined previously for the Hillenbrand dataset.

Average male count for IDs starting with 'M': 13

Average female count for IDs starting with 'M': 2

Average female count for IDs starting with 'W': 11

Average male count for IDs starting with 'W': 5

The data from the Peterson-Barney dataset supports the conclusions drawn from the Hillenbrand dataset.

Similar to the Hillenbrand dataset, we observe a significantly higher count of individuals of the same gender compared to the opposite gender in the category of speakers with higher Aij values.

Data Source: The data for the Peterson-Barney and Hillenbrand vowel datasets is sourced from the GitHub repository you provided.