# probability\_calculator.py saved under 4. Source code folder

‘’’

Below function will calculate the **probability** and **p-value**. For this purpose, the **alignment score** will be used (calculated as part of the alignment\_sequences.py) as a measure of how good the match is. The probability can be derived from the alignment score (e.g., using a statistical distribution), and the p-value can indicate how likely it is that the match happened by chance.

For simplicity, a basic statistical model is assumed where we compare the alignment score against a **null distribution** of scores from the database to calculate the p-value.

To derive to the stretched goal, the following executable steps are used:

**calculate\_probability**: This function computes the probability of the best alignment score based on a normal distribution derived from the alignment scores of the database sequences.

**calculate\_p\_value**: This calculates the **p-value** based on the Z-score.

**generate\_score\_distribution**: This generates a distribution of alignment scores for all sequences in the database, which we then use to calculate the p-value.

**scipy.stats**: We use the scipy library to perform the **Z-test** for calculating the p-value.

‘’’

import numpy as np

from scipy import stats

def calculate\_probability(alignment\_score, score\_distribution):

"""Calculate the probability of the alignment score based on the distribution of scores."""

# For simplicity, use a Z-test to calculate the probability

mean\_score = np.mean(score\_distribution)

std\_dev = np.std(score\_distribution)

z\_score = (alignment\_score - mean\_score) / std\_dev

p\_value = stats.norm.cdf(z\_score) # p-value for the Z-score

return p\_value

def calculate\_p\_value(best\_score, score\_distribution):

"""Calculate the p-value for the best alignment score."""

p\_value = calculate\_probability(best\_score, score\_distribution)

return p\_value

def generate\_score\_distribution(database\_sequences, mystery\_sequence, align\_sequences):

"""Generate a distribution of alignment scores from the database sequences."""

scores = []

for db\_sequence in database\_sequences:

alignment = align\_sequences(mystery\_sequence.seq, db\_sequence.seq)

if alignment:

scores.append(alignment[2]) # Store the alignment score

return scores

# Example usage:

# Assuming `find\_best\_match` returns the best alignment score and the alignment itself

def main():

# Generate a score distribution based on the sequences in the database

database\_sequences = ... # Loaded dog breeds sequences

mystery\_sequence = ... # Loaded mystery sequence

score\_distribution = generate\_score\_distribution(database\_sequences, mystery\_sequence, align\_sequences)

# Get the best match (alignment score)

best\_match, best\_score = find\_best\_match(mystery\_sequence, database\_sequences)

# Calculate the p-value for the best alignment score

p\_value = calculate\_p\_value(best\_score, score\_distribution)

print(f"Best Score: {best\_score}")

print(f"P-Value: {p\_value}")

if \_\_name\_\_ == "\_\_main\_\_":

main()