Report - Lab: SO Perceptron

name: Filip Špidla

email: spidlfil@fel.cvut.cz

In [ ]:

```
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
from os import listdir
from os.path import isfile, join

# load single example
def load_example( img_path ):

    Y = img_path[img_path.rfind('_')+1:-4]

    img = Image.open( img_path )
    img_mat = np.asarray( img )

    n_letters = len( Y )
    im_height = int(img_mat.shape[0])
    im_width = int(img_mat.shape[1]/n_letters)
    n_pixels = im_height*im_width

    X = np.zeros( [int(n_pixels+n_pixels*(n_pixels-1)/2),n_letters])
    for i in range(n_letters):

        # single letter
        letter = img_mat[:,i*im_width:(i+1)*im_width]/255

        # compute features
        x = letter.flatten()
        X[0:len(x),i] = x
        cnt = n_pixels
        for j in range(0,n_pixels-1):
            for k in range(j+1,n_pixels):
                X[cnt,i] = x[j]*x[k]
                cnt = cnt + 1

        X[:,i] = X[:,i]/np.linalg.norm(X[:,i])

    return X, Y, img

# load all examples from a folder
def load_examples( image_folder ):

    files = [f for f in listdir(image_folder) if isfile(join(image_folder, f))]

    X = []
    Y = []
    img = []
    for file in listdir(image_folder):
        path = join(image_folder, file)
        if isfile( path ):

            X_,Y_,img_ = load_example( path )
            X.append( X_ )
            Y.append( Y_ )
            img.append( img_ )

    return X, Y, img

# Load training examples
trn_X, trn_Y, trn_img = load_examples( 'ocr_names_images/trn' )

# Load testing examples
tst_X, tst_Y, tst_img = load_examples( 'ocr_names_images/tst' )
```

Assignment 1 (3 points) Implement the Perceptron algorithm for learning parameters (w ∈ R d·|A| , b ∈ R |A|) of the linear multi-class classifier (1). Use the provided training examples T m to learn parameters of the classifier. Report the sequence prediction error Rseq and the character prediction error Rchar computed on the provided testing examples S l . The output should be a single script (Jupyter notebook or Matlab) which learns the classifier and prints the computed testing errors.

In [ ]:

```
def perceptron_train(X, Y, epochs=100):

    #Map each unique label in Y to a unique integer identifier
    unique_labels = list(set(Y))
    label_to_id = {label: i for i, label in enumerate(unique_labels)}

    num_features = X[0].shape[0]
    num_classes = len(unique_labels)  #

    weights = np.zeros((num_features, num_classes))
    biases = np.zeros(num_classes)

    #training loop ofor each epoch
    for epoch in range(epochs):
        for features, label in zip(X, Y):
            label_id = label_to_id[label]

            summed_features = np.sum(features, axis=1)
            scores = np.dot(summed_features, weights) + biases
            predicted_class = np.argmax(scores)

            #update weights and biases if the prediction is incorrect
            if predicted_class != label_id:
                weights[:, label_id] += summed_features
                weights[:, predicted_class] -= summed_features
                biases[label_id] += 1
                biases[predicted_class] -= 1

    return weights, biases, unique_labels

def perceptron_predict(X, weights, biases, unique_labels):
    predictions = []
    for features in X:
        summed_features = np.sum(features, axis=1)
        scores = np.dot(summed_features, weights) + biases
        predicted_class = np.argmax(scores)
        predicted_label = unique_labels[predicted_class]
        predictions.append(predicted_label)

    return predictions

def compute_errors(actual_labels, predicted_labels):
    sequence_error_count = 0
    character_error_count = 0
    total_characters = 0

    # iterate over each pair of actual and predicted labels
    for actual, predicted in zip(actual_labels, predicted_labels):
        if not np.array_equal(actual, predicted):  # Compare arrays element-wise
            sequence_error_count += 1

        for char_actual, char_predicted in zip(actual, predicted):
            if char_actual != char_predicted:
                character_error_count += 1

        total_characters += len(actual)

    sequence_error_rate = sequence_error_count / len(actual_labels)
    character_error_rate = character_error_count / total_characters

    return sequence_error_rate, character_error_rate

w, b, lb = perceptron_train(trn_X, trn_Y)
Y_pred = perceptron_predict(tst_X, w, b, lb)
R_seq, R_char = compute_errors(tst_Y, Y_pred)

print("Sequence Prediction Error (R_seq):", R_seq)
print("Character Prediction Error (R_char):", R_char)
```

```
Sequence Prediction Error (R_seq): 0.13
Character Prediction Error (R_char): 0.11567516525023608
```

Assignment 2 (3 points) Implement the Perceptron algorithm for learning parameters (w ∈ R d·|A| , b ∈ R |A|, g ∈ R |A|2) of the linear structured output classifier (2). Evaluate the algorithm as specified in Assignment 1.

In [ ]:

```
def structured_perceptron_train(X, Y, epochs=300):
    unique_labels = list(set(Y))
    label_to_id = {label: i for i, label in enumerate(unique_labels)}

    num_features = X[0].shape[0]
    num_classes = len(unique_labels)

    weights = np.zeros((num_features, num_classes))
    biases = np.zeros(num_classes)
    g = np.zeros((num_classes, num_classes))  # Structured parameter
    for epoch in range(epochs):
        for features, label in zip(X, Y):
            label_id = label_to_id[label]

            summed_features = np.sum(features, axis=1)

            # Calculate scores with the structured output
            scores = np.dot(summed_features, weights) + biases
            for i in range(num_classes):
                for j in range(num_classes):
                    scores[i] += g[i][j]  # Incorporate structured parameter

            predicted_class = np.argmax(scores)

            if predicted_class != label_id:
                weights[:, label_id] += summed_features
                weights[:, predicted_class] -= summed_features
                biases[label_id] += 1
                biases[predicted_class] -= 1

                for i in range(num_classes):
                    g[label_id][i] += 1
                    g[predicted_class][i] -= 1

    return weights, biases, g, unique_labels


def structured_perceptron_predict(X, weights, biases, g, unique_labels):
    predictions = []
    for features in X:
        summed_features = np.sum(features, axis=1)
        scores = np.dot(summed_features, weights) + biases

        # Incorporate structured parameter g into the scores
        for i in range(len(unique_labels)):
            for j in range(len(unique_labels)):
                scores[i] += g[i][j]

        predicted_class = np.argmax(scores)
        predicted_label = unique_labels[predicted_class]
        predictions.append(predicted_label)

    return predictions

w, b, g, lb = structured_perceptron_train(trn_X, trn_Y)
Y_pred = structured_perceptron_predict(tst_X, w, b, g, lb)
R_seq, R_char = compute_errors(tst_Y, Y_pred)


print("Sequence Prediction Error (R_seq):", R_seq)
print("Character Prediction Error (R_char):", R_char)
```

```
Sequence Prediction Error (R_seq): 0.18
Character Prediction Error (R_char): 0.15722379603399433
```

Assignment 3 (3 points) Implement the Perceptron algorithm for learning parameters (w ∈ R d·|A| , b ∈ R |A|, v ∈ R |Y|) of the linear structured output classifier (4). Evaluate the algorithm as specified in Assignment 1.

In [ ]:

```
def perceptron_v_train(X, Y, epochs=2000):
    unique_labels = list(set(Y))
    label_to_id = {label: i for i, label in enumerate(unique_labels)}

    num_features = X[0].shape[0]
    num_classes = len(unique_labels)

    weights = np.zeros((num_features, num_classes))
    biases = np.zeros(num_classes)
    v = np.zeros(len(X), dtype=int)  # Ensure v is of integer type

    for epoch in range(epochs):
        errors = 0
        for idx, (features, label) in enumerate(zip(X, Y)):
            label_id = label_to_id[label]
            summed_features = np.sum(features, axis=1)
            scores = np.dot(summed_features, weights) + biases
            scores += v[idx]

            predicted_class = np.argmax(scores)

            if not np.array_equal([predicted_class], [label_id]):  # Convert to arrays
                weights[:, label_id] += summed_features
                weights[:, predicted_class] -= summed_features
                biases[label_id] += 1
                biases[predicted_class] -= 1
                v[idx] += 1
                errors += 1

        if errors == 0:
            break

    return weights, biases, v

def perceptron_v_predict(X, Y, weights, biases, v):
    unique_labels = list(set(Y))
    predictions = []
    for idx, features in enumerate(X):
        summed_features = np.sum(features, axis=1)
        scores = np.dot(summed_features, weights) + biases
        scores += v[idx]

        predicted_class = np.argmax(scores)
        predicted_label = unique_labels[predicted_class]
        predictions.append(predicted_label)

    return predictions


weights, biases, v = perceptron_v_train(trn_X, trn_Y)
Y_pred = perceptron_v_predict(tst_X, trn_Y, weights, biases, v)
Y_pred_sanity = perceptron_v_predict(trn_X, trn_Y, weights, biases, v)
R_seq, R_char = compute_errors(trn_Y, Y_pred_sanity)

print("sanity check (R_seq):", R_seq)
print("sanity check (R_char):", R_char)

R_seq, R_char = compute_errors(tst_Y, Y_pred)

print("Sequence Prediction Error (R_seq):", R_seq)
print("Character Prediction Error (R_char):", R_char)
```

```
sanity check (R_seq): 0.0
sanity check (R_char): 0.0
Sequence Prediction Error (R_seq): 0.13
Character Prediction Error (R_char): 0.11567516525023608
```

Assignment 4 (1 point) Summarize the testing errors of the three learned classifiers in a single table. Explain differences in the performance of the three classifiers. Point out the main advantages and disadvantages of each classification model.

| | R_{seq} | R_{char} | |-----------------------------------|----------------|------------------| | independent multi-class classifier| 0.13 | 0.115 | | structured, pair-wise dependency | 0.18 | 0.157 | | structured, fixed number of sequences | 0.13 | 0.115 |

Discussion of results

Independent linear multi-class classifier is the simplest one among implemented classifiers. It computes every feasible option independently and is quite fast, however, it completely disregards dependence between characters. I suspect it would be hampered by more complex data, that would for example contain high colinearity between features. It is the only model that does not consider any dependency between features.

Structured, pair-wise dependency considers dependences between two characters, allowing more complex model at the cost of higher computational complexity. I think it would work best for data where there is linear relationship between predicted variable and features.

As for structured, fixed number of sequences classifier, it predicts sequences with a fixed number of elements, which by itself presumes some relationship between characters. Sequences other than the ones presumed - such as any of the sequences which are not in training dataset, are not possible to be predicted, meaning that model could miss some dependencies between characters which would be present in training data. Therefore the model will likely be best for data with small number of possible sequences, but loses it's advantages with increasing number of possible sequences.

The Indpendent multi-class classifier and structured, fixed number of sequences classifier both reach the same accuracy of classification. Also, they both classify as one from the set of labels given in training data. This suggests to me that both classifiers likely reached perfect score for the training data (structured, fixed sequence has 0 error rate on training data for sure), and are classifying testing data in the same manner.

Structured, pair-wise dependency is worse at classifying sequences, but better at characters. This is likely because this classifier model is focused in particullar on relationships between singular characters, likely at the cost of sequence accuracy.

Assignment 5 (5 bonus points) Describe an instance of the Structured Output SVM algorithm for learning the classifier (2) which uses the character prediction error Rchar as the target loss function. Learn the classifier from the training data and report its test performance in terms of the sequence prediction error Rseq and the character prediction error Rchar .

Input of Structured Output SVM (Structured Vector Machine) is binary image as per second point of our assignment. It is a linear classifier designed for sequence prediction tasks. Each sequence is a string of characters, and the goal is to correctly predict these sequences.

SVMs use loss function to quantify the departure of prediction from the actual output variable. Rchar handily slots into this utility, since its one of two metrics we were using to monitor performance. We will try to minimise loss function - model's Rchar.

I was not explicitly told to implement this SVM myself, so I assume I can use libraries for SVM.

In [ ]:

```python
from sklearn.svm import LinearSVC

def flatten_training_data(X, Y):
    flattened_data = []
    labels = []

    for i, (data_point, word) in enumerate(zip(X, Y)):
        for char_index in range(len(word)):
            # Collect the char_index-th element from each of the 8256 features
            char_features = data_point[:, char_index]
            flattened_data.append(char_features)
            labels.append(word[char_index])

    return np.array(flattened_data), np.array(labels)

def compute_character_error(actual_labels, predicted_labels):
    character_error_count = 0
    for actual, predicted in zip(actual_labels, predicted_labels):
        if actual != predicted:
            character_error_count += 1
    return character_error_count / len(actual_labels)

def train_svm_with_custom_loss(X, Y, epochs=10, tolerance=0.001):
    best_model = None
    lowest_char_error = float('inf')

    for epoch in range(epochs):
        model = LinearSVC(random_state=epoch, tol=1e-5)
        model.fit(X, Y)
        Y_pred = model.predict(X)

        char_error = compute_character_error(Y, Y_pred)
        if char_error <= lowest_char_error:
            print(f"updating eror to {char_error}")
            lowest_char_error = char_error
            best_model = model

        if char_error < tolerance:
            print("tolerand enough")
            break

    return best_model

def predict_and_evaluate(model, tst_X, tst_Y):
    predicted_words = []
    for data_point in tst_X:
        predicted_word = ''
        for char_index in range(data_point.shape[1]):
            char_features = data_point[:, char_index]
            predicted_char = model.predict([char_features])[0]
            predicted_word += predicted_char
        predicted_words.append(predicted_word)

    R_seq, R_char = compute_errors(tst_Y, predicted_words)
    return predicted_words, R_seq, R_char

flattened_data, labels = flatten_training_data(trn_X, trn_Y)
model = train_svm_with_custom_loss(flattened_data, labels)
predicted_words, R_seq, R_char = predict_and_evaluate(model, tst_X, tst_Y)

print("Predicted words:", predicted_words)
print("Proper words:", tst_Y)
print("Sequence Prediction Error (R_seq):", R_seq)
print("Character Prediction Error (R_char):", R_char)
```

```
updating eror to 0.012027491408934709
updating eror to 0.012027491408934709
updating eror to 0.012027491408934709
updating eror to 0.012027491408934709
updating eror to 0.012027491408934709
updating eror to 0.012027491408934709
updating eror to 0.012027491408934709
updating eror to 0.012027491408934709
updating eror to 0.012027491408934709
updating eror to 0.012027491408934709
Predicted words: ['raleh', 'fioya', 'raleh', 'fiosb', 'raieh', 'ciiytord', 'won', 'ra
ifh', 'tioyd', 'dwlgft', 'raipb', 'ralph', 'hulnn', 'drwck', 'fhllip', 'rioyd', 'plii
ip', 'fioyd', 'cuipk', 'yg', 'dwldht', 'ralpu', 'fiogd', 'dwlght', 'fioyd', 'ty', 'qu
inn', 'ty', 'ceuz', 'steve', 'joseeh', 'ty', 'mdx', 'jacp', 'max', 'dwisdt', 'bc', 'c
rwz', 'dwipht', 'mox', 'ty', 'bxouc', 'nninn', 'eluij', 'ralyh', 'quinn', 'dwight',
'ftoyd', 'nax', 'brock', 'jark', 'qulnn', 'bo', 'cruz', 'crdz', 'ralph', 'ho', 'floy
d', 'joseph', 'ty', 'yreva', 'oo', 'ty', 'ty', 'ralph', 'steve', 'drdw', 'max', 'jaa
u', 'tg', 'drew', 'dwiyht', 'ralph', 'brock', 'crut', 'bo', 'joseph', 'ralph', 'qvin
n', 'qnimw', 'by', 'jacu', 'crue', 'max', 'max', 'cruc', 'flold', 'mnx', 'ckuz', 'dwl
gnt', 'tyoyd', 'tloyd', 'ty', 'cruz', 'po', 'fgoyd', 'rloyd', 'tg', 'max', 'floxd',
'ty', 'kcnyd', 'max', 'quinn', 'ralph', 'elvis', 'grey', 'arcz', 'iloyd', 'hvgh', 'cr
uz', 'qninn', 'stfye', 'tg', 'may', 'crua', 'cruz', 'bo', 'mau', 'gaeg', 'do', 'gnim
n', 'quinn', 'floyd', 'quinn', 'bxvck', 'hhgh', 'drew', 'ty', 'steva', 'cruz', 'quin
n', 'dvidhl', 'qulnn', 'ty', 'dwigkd', 'jobapk', 'dwtyht', 'max', 'max', 'quiwn', 'el
uis', 'quinn', 'jack', 'dwlaht', 'steve', 'zloyd', 'quinn', 'cruz', 'cruz', 'jack',
'dwight', 'jacn', 'uock', 'baock', 'droch', 'crvz', 'floyd', 'jnck', 'joreyk', 'rglr
h', 'clircord', 'dwiaht', 'quinn', 'floyd', 'bo', 'ty', 'dwigtt', 'guinn', 'tt', 'jac
k', 'ho', 'cruz', 'ccuz', 'dwiqht', 'brack', 'ralph', 'steve', 'bicck', 'ty', 'ty',
'cruz', 'quinn', 'ts', 'crvz', 'yclth', 'jack', 'bo', 'max', 'qutnn', 'ty', 'max', 'j
nck', 'qminn', 'max', 'pkclip', 'floyd', 'ty', 'quinn', 'ouinn', 'cruz', 'elvls', 'b
o', 'floyd', 'quinu', 'ty', 'huyh', 'fioyd', 'dwiqht', 'qulnn', 'stcve', 'brock', 'go
seph', 'jaok', 'dwiyhr', 'rlogd', 'max', 'jack', 'mrx', 'rmqqh', 'quinn', 'quynn', 's
rck', 'josaph', 'mat', 'flogd', 'elvig', 'floyd', 'mnx', 'steve', 'eluib', 'mox', 'el
vis', 'max', 'cruz', 'broct', 'dwiqht', 'floyd', 'floyd', 'mox', 'crdz', 'eluis', 'ja
ck', 'dwight', 'devyn', 'uoseph', 'fteve', 'dwiyht', 'jacr', 'mdx', 'nax', 'ty', 'ma
x', 'quinn', 'bo', 'bg', 'ty', 'philip', 'brock', 'josepn', 'owlght', 'tq', 'max', 't
teve', 'qvinn', 'jdck', 'hioyd', 'bo', 'dioyd', 'cruz', 'jacb', 'wax', 'devyn', 'bo',
'dwiehy', 'ty', 'cruz', 'ho', 'bfoce', 'bo', 'fioyd', 'oruz', 'bo', 'cruq', 'cruj',
'rloyd', 'bo', 'jack', 'qreg', 'qy', 'quinn', 'crut', 'max', 'ty', 'stevt', 'gteve',
'bo', 'floyd', 'josepx', 'steve', 'dwighb', 'hagh', 'jmcx', 'qalnu', 'drew', 'cyuz',
'huxh', 'oevyq', 'max', 'huqh', 'iy', 'btuch', 'steue', 'du', 'ho', 'flogd', 'stuve',
'dnighe', 'ho', 'quinn', 'jach', 'jnck', 'iy', 'clifford', 'tloyd', 'hugd', 'ty', 'ma
x', 'crnz', 'ehoyd', 'steve', 'td', 'dryw', 'rloyd', 'dloyd', 'fioyd', 'floyd', 'steu
z', 'tloyd', 'mav', 'crut', 'do', 'max', 'cvvz', 'oevyn', 'brock', 'quinn', 'shebe',
'jach', 'max', 'rdlph', 'rkoxd', 'dwight', 'cfaz', 'quinn', 'cruz', 'flouj', 'tloyd',
'guinn', 'bo', 'qcimn', 'cruz', 'floyd', 'ploya', 'philip', 'rteve', 'philip', 'mar',
'mav', 'max', 'quinn', 'qoinn', 'goseph', 'ux', 'cyuz', 'bo', 'eluis', 'dacu', 'turn
n', 'max', 'ifcx', 'josepb', 'floyd', 'ralph', 'dojryh', 'vy', 'bo', 'jdch', 'ty', 'c
cve', 'drtw', 'dwight', 'craz', 'tloyd', 'tax', 'dwlohe', 'greg', 'crue', 'gteve', 'm
af', 'jack', 'mnx', 'steve', 'tloyo', 'ho', 'hush', 'dwigkf', 'sy', 'fcoyd', 'rloqd',
'ralqh', 'voscpn', 'jack', 'bugh', 'cruz', 'ywighj', 'cruz', 'fioyd', 'bo', 'max', 'l
udf', 'akviq', 'floyd', 'crue', 'ty', 'ho', 'pmiilp', 'dvcw', 'duinn', 'ty', 'phoyd',
'philip', 'tg', 'do', 'orus', 'hugh', 'broek', 'joseph', 'dwight', 'dcvyn', 'dvew',
'duight', 'ernz', 'phiiie', 'dwioht', 'ioseph', 'qked', 'joseph', 'tg', 'jaex', 'joge
ph', 'jacd', 'jock', 'max', 'ccuz', 'ralph', 'doseeh', 'bo', 'xy', 'joseph', 'quiuz',
'crqz', 'dtgyd', 'cruz', 'tu', 'crvt', 'max', 'deryv', 'sloyd', 'max', 'crut', 'max',
'floyd', 'drer', 'gteve', 'yy', 'ralpk', 'drew', 'dvew', 'crus', 'hugb', 'floyd', 'ei
vis', 'bo', 'ralph', 'flood', 'mdx', 'cruz', 'puinn', 'seeve', 'elvis', 'ordak', 'sme
ve', 'tx', 'yuinn', 'jock', 'cruz', 'jeunn', 'dwight', 'cruy']
Proper words: ['ralph', 'floyd', 'ralph', 'floyd', 'ralph', 'clifford', 'max', 'ralp
h', 'floyd', 'dwight', 'ralph', 'ralph', 'quinn', 'brock', 'philip', 'floyd', 'phili
p', 'floyd', 'ralph', 'ty', 'dwight', 'ralph', 'floyd', 'dwight', 'floyd', 'ty', 'qui
nn', 'ty', 'cruz', 'steve', 'joseph', 'ty', 'max', 'jack', 'max', 'dwight', 'bo', 'cr
uz', 'dwight', 'max', 'ty', 'brock', 'quinn', 'elvis', 'ralph', 'quinn', 'dwight', 'f
loyd', 'max', 'brock', 'jack', 'quinn', 'bo', 'cruz', 'cruz', 'ralph', 'bo', 'floyd',
'joseph', 'ty', 'steve', 'bo', 'ty', 'ty', 'ralph', 'steve', 'drew', 'max', 'jack',
'ty', 'drew', 'dwight', 'ralph', 'brock', 'cruz', 'bo', 'joseph', 'ralph', 'quinn',
'quinn', 'ty', 'jack', 'cruz', 'max', 'max', 'cruz', 'floyd', 'max', 'cruz', 'dwigh
t', 'floyd', 'floyd', 'ty', 'cruz', 'bo', 'floyd', 'floyd', 'ty', 'max', 'floyd', 't
y', 'devyn', 'max', 'quinn', 'ralph', 'elvis', 'greg', 'cruz', 'floyd', 'hugh', 'cru
z', 'quinn', 'steve', 'ty', 'max', 'cruz', 'cruz', 'bo', 'max', 'greg', 'bo', 'quin
n', 'quinn', 'floyd', 'quinn', 'brock', 'hugh', 'drew', 'ty', 'steve', 'cruz', 'quin
n', 'dwight', 'quinn', 'ty', 'dwight', 'joseph', 'dwight', 'max', 'max', 'quinn', 'el
vis', 'quinn', 'jack', 'dwight', 'steve', 'floyd', 'quinn', 'cruz', 'cruz', 'jack',
'dwight', 'jack', 'jack', 'brock', 'brock', 'cruz', 'floyd', 'jack', 'joseph', 'ralp
h', 'clifford', 'dwight', 'quinn', 'floyd', 'bo', 'ty', 'dwight', 'quinn', 'ty', 'jac
k', 'bo', 'cruz', 'cruz', 'dwight', 'brock', 'ralph', 'steve', 'brock', 'ty', 'ty',
'cruz', 'quinn', 'ty', 'cruz', 'ralph', 'jack', 'bo', 'max', 'quinn', 'ty', 'max', 'j
ack', 'quinn', 'max', 'philip', 'floyd', 'ty', 'quinn', 'quinn', 'cruz', 'elvis', 'b
o', 'floyd', 'quinn', 'ty', 'hugh', 'floyd', 'dwight', 'quinn', 'steve', 'brock', 'jo
seph', 'jack', 'dwight', 'floyd', 'max', 'jack', 'max', 'ralph', 'quinn', 'quinn', 'j
ack', 'joseph', 'max', 'floyd', 'elvis', 'floyd', 'max', 'steve', 'elvis', 'max', 'el
vis', 'max', 'cruz', 'brock', 'dwight', 'floyd', 'floyd', 'max', 'cruz', 'elvis', 'ja
ck', 'dwight', 'devyn', 'joseph', 'steve', 'dwight', 'jack', 'max', 'max', 'ty', 'ma
x', 'quinn', 'bo', 'bo', 'ty', 'philip', 'brock', 'joseph', 'dwight', 'ty', 'max', 's
teve', 'quinn', 'jack', 'floyd', 'bo', 'floyd', 'cruz', 'jack', 'max', 'devyn', 'bo',
'dwight', 'ty', 'cruz', 'bo', 'brock', 'bo', 'floyd', 'cruz', 'bo', 'cruz', 'cruz',
'floyd', 'bo', 'jack', 'greg', 'ty', 'quinn', 'cruz', 'max', 'ty', 'steve', 'steve',
'bo', 'floyd', 'joseph', 'steve', 'dwight', 'hugh', 'jack', 'quinn', 'drew', 'cruz',
'hugh', 'devyn', 'max', 'hugh', 'ty', 'brock', 'steve', 'bo', 'bo', 'floyd', 'steve',
'dwight', 'bo', 'quinn', 'jack', 'jack', 'ty', 'clifford', 'floyd', 'hugh', 'ty', 'ma
x', 'cruz', 'floyd', 'steve', 'ty', 'drew', 'floyd', 'floyd', 'floyd', 'floyd', 'stev
e', 'floyd', 'max', 'cruz', 'bo', 'max', 'cruz', 'devyn', 'brock', 'quinn', 'steve',
'jack', 'max', 'ralph', 'floyd', 'dwight', 'cruz', 'quinn', 'cruz', 'floyd', 'floyd',
'quinn', 'bo', 'quinn', 'cruz', 'floyd', 'floyd', 'philip', 'steve', 'philip', 'max',
'max', 'max', 'quinn', 'quinn', 'joseph', 'ty', 'cruz', 'bo', 'elvis', 'jack', 'quin
n', 'max', 'jack', 'joseph', 'floyd', 'ralph', 'joseph', 'ty', 'bo', 'jack', 'ty', 'c
ruz', 'drew', 'dwight', 'cruz', 'floyd', 'max', 'dwight', 'greg', 'cruz', 'steve', 'm
ax', 'jack', 'max', 'steve', 'floyd', 'bo', 'hugh', 'dwight', 'ty', 'floyd', 'floyd',
'ralph', 'joseph', 'jack', 'hugh', 'cruz', 'dwight', 'cruz', 'floyd', 'bo', 'max', 'h
ugh', 'elvis', 'floyd', 'cruz', 'ty', 'bo', 'philip', 'drew', 'quinn', 'ty', 'floyd',
'philip', 'ty', 'bo', 'cruz', 'hugh', 'brock', 'joseph', 'dwight', 'devyn', 'drew',
'dwight', 'cruz', 'philip', 'dwight', 'joseph', 'greg', 'joseph', 'ty', 'jack', 'jose
ph', 'jack', 'jack', 'max', 'cruz', 'ralph', 'joseph', 'bo', 'ty', 'joseph', 'quinn',
'cruz', 'floyd', 'cruz', 'ty', 'cruz', 'max', 'devyn', 'floyd', 'max', 'cruz', 'max',
'floyd', 'drew', 'steve', 'ty', 'ralph', 'drew', 'drew', 'cruz', 'hugh', 'floyd', 'el
vis', 'bo', 'ralph', 'floyd', 'max', 'cruz', 'quinn', 'steve', 'elvis', 'brock', 'ste
ve', 'ty', 'quinn', 'jack', 'cruz', 'devyn', 'dwight', 'cruz']
Sequence Prediction Error (R_seq): 0.63
Character Prediction Error (R_char): 0.21104815864022664
```