Question 1

```
Training Error P=3:  0.01349862258953168
Testing Error P=3:  0.051451187335092345

Training Error P=4:  0.013223140495867768
Testing Error P=4:  0.03825857519788918

Training Error P=5:  0.01046831955922865
Testing Error P=5:  0.07387862796833773
```

Question 2:

```
Training Error P=3:  0.011294765840220386
Testing Error P=3:  0.05804749340369393

Training Error P=4:  0.011019283746556474
Testing Error P=4:  0.03562005277044855

Training Error P=5:  0.01046831955922865
Testing Error P=5:  0.07387862796833773
```

Question 3:

```
Top 2 Substrings:  [('TAGQE', 3573), ('DTAGQ', 5065)]
```

In [1]:

```
import pandas as pd
import numpy as np
```

In [2]:

```
# Read in data
training_data = pd.read_csv('./data/pa4train.txt', sep=" ", header=None)
testing_data = pd.read_csv('./data/pa4test.txt', sep=" ", header=None)
training_data.head()
print(training_data.shape)
```

```
(3630, 2)
```

In [3]:

```
# Function for calculating prediction error
def calc_error(Y_pred, Y_label):
    # Calculate Error Rate for Predicted Labels
    error = [0 for x,y in zip(Y_pred,Y_label) if x != y]
    error_rate = len(error)/len(Y_pred)
    return error_rate

# Function for computing the number of common substrings of length P
def common_substrings(S,T,p, sub_dict):
    common_substrings = {}
    T_string = T[0]
    T_label = T[1]
    for i in range((len(S) - p + 1)):
        v = S[i:(i+p)]
        v = str(v)
        if v in T_string:
            occurences_T = T_string.count(v)
            occurences_S = S.count(v)
            if v in sub_dict.keys():
                sub_dict[v] += (T_label)*(occurences_T*occurences_S)
            else:
                sub_dict[v] = (T_label)*(occurences_T*occurences_S)
            if T_string not in common_substrings.keys():
                common_substrings[v] = occurences_T*occurences_S
    return sum(common_substrings.values())

# Function for computing the number of common substrings of length P
def modified_substrings(S,T,p):
    common_substrings = {}
    for i in range((len(S) - p + 1)):
        v = S[i:(i+p)]
        v = str(v)
        if v in T:
            if T not in common_substrings.keys():
                common_substrings[v] = 1
    return sum(common_substrings.values())
```

In [4]:

```
class kernelPerceptron:
    # Constructor
    def __init__(self):
        self.w = []
        self.substrings = {}

    # Returns predicted label for sample data point
    # W is a list where each element is a tuple (Word, label)
    def kernel(self, sample_point, p):
        num_common = {-1: 0, 1: 0}
        for pair in self.w:
            label = pair[1]
            common_sub = common_substrings(sample_point[0], pair, p, sel
f.substrings)
            num_common[label] += common_sub
        label = 1
        if num_common[-1] > num_common[1]:
            label = -1
        return label

    # Modified version of string kernel
    def modifiedKernel(self, sample_point, p):
        num_common = {-1: 0, 1: 0}
        for pair in self.w:
            string = pair[0]
            label = pair[1]
            common_sub = modified_substrings(sample_point[0], string, p)
            num_common[label] += common_sub
        label = 1
        if num_common[-1] > num_common[1]:
            label = -1
        return label

    # train perceptron and modify w parameter
    def train(self, kernel_type,  training_data, p):
        self.w = []
        label = training_data.shape[1] - 1
        for index, row in training_data.iterrows():
            true_label = row[label]
            if kernel_type == 'modified':
                kernel_label = self.modifiedKernel(row ,p)
            else:
                kernel_label = self.kernel(row ,p)
            if kernel_label*true_label <= 0:
                # misclassified string, modify w
                self.w.append((row[0], true_label))

    # Make predictions for each data point in testing data
    def predict(self, kernel_type, test_data, p):
        # Predict label for each data point
        predictions = []
        for index, row in test_data.iterrows():
            if kernel_type == 'modified':
                predictions.append(self.modifiedKernel(row, p))
            else:
                predictions.append(self.kernel(row, p))
        return predictions
```

Question 1

In [5]:

```
# Tests
clf = kernelPerceptron()
clf.train('normal',training_data, 2)
y_pred_train = clf.predict('normal',training_data, 2)
y_true_train = training_data[(training_data.shape[1] - 1)]
y_pred_test = clf.predict('normal',testing_data, 2)
y_true_test = testing_data[(testing_data.shape[1] - 1)]
print("Training Error P=2: ",calc_error(y_pred_train, y_true_train))
print("Testing Error P=2: ",calc_error(y_pred_test, y_true_test))
```

```
Training Error P=3:  0.07024793388429752
Testing Error P=3:  0.0870712401055409
```

In [14]:

```
# Tests
clf_p_3 = kernelPerceptron()
clf_p_3.train('normal',training_data, 3)
y_pred_train = clf_p_3.predict('normal',training_data, 3)
y_true_train = training_data[(training_data.shape[1] - 1)]
y_pred_test = clf_p_3.predict('normal',testing_data, 3)
y_true_test = testing_data[(testing_data.shape[1] - 1)]
print("Training Error P=3: ",calc_error(y_pred_train, y_true_train))
print("Testing Error P=3: ",calc_error(y_pred_test, y_true_test))
```

```
Training Error P=3:  0.013498622589531680
Testing Error P=3:  0.05145118733509234 5
```

In [7]:

```
clf_p_4 = kernelPerceptron()
clf_p_4.train('normal',training_data, 4)
y_pred_train = clf_p_4.predict('normal', training_data, 4)
y_true_train = training_data[(training_data.shape[1] - 1)]
y_pred_test = clf_p_4.predict('normal',testing_data, 4)
y_true_test = testing_data[(testing_data.shape[1] - 1)]
print("Training Error P=4: ",calc_error(y_pred_train, y_true_train))
print("Testing Error P=4: ",calc_error(y_pred_test, y_true_test))
```

```
Training Error P=4:  0.013223140495867768
Testing Error P=4:  0.03825857519788918
```

In [8]:

```
clf_p_5 = kernelPerceptron()
clf_p_5.train('normal',training_data, 5)
y_pred_train = clf_p_5.predict('normal',training_data, 5)
y_true_train = training_data[(training_data.shape[1] - 1)]
y_pred_test = clf_p_5.predict('normal',testing_data, 5)
y_true_test = testing_data[(testing_data.shape[1] - 1)]
print("Training Error P=5: ",calc_error(y_pred_train, y_true_train))
print("Testing Error P=5: ",calc_error(y_pred_test, y_true_test))
```

```
Training Error P=5:  0.010468319559228650
Testing Error P=5:  0.07387862796833773
```

Question 2

In [9]:

```
# Tests
clf = kernelPerceptron()
clf.train('modified',training_data, 3)
y_pred_train = clf.predict('modified',training_data, 3)
y_true_train = training_data[(training_data.shape[1] - 1)]
y_pred_test = clf.predict('modified',testing_data, 3)
y_true_test = testing_data[(testing_data.shape[1] - 1)]
print("Training Error P=3: ",calc_error(y_pred_train, y_true_train))
print("Testing Error P=3: ",calc_error(y_pred_test, y_true_test))
```

```
Training Error P=3:  0.011294765840220386
Testing Error P=3:  0.05804749340369393
```

In [10]:

```
# Tests
clf = kernelPerceptron()
clf.train('modified',training_data, 4)
y_pred_train = clf.predict('modified',training_data, 4)
y_true_train = training_data[(training_data.shape[1] - 1)]
y_pred_test = clf.predict('modified',testing_data, 4)
y_true_test = testing_data[(testing_data.shape[1] - 1)]
print("Training Error P=4: ",calc_error(y_pred_train, y_true_train))
print("Testing Error P=4: ",calc_error(y_pred_test, y_true_test))
```

```
Training Error P=4:  0.011019283746556474
Testing Error P=4:  0.03562005277044855
```

In [11]:

```
# Tests
clf = kernelPerceptron()
clf.train('modified',training_data, 5)
y_pred_train = clf.predict('modified',training_data, 5)
y_true_train = training_data[(training_data.shape[1] - 1)]
y_pred_test = clf.predict('modified',testing_data, 5)
y_true_test = testing_data[(testing_data.shape[1] - 1)]
print("Training Error P=5: ",calc_error(y_pred_train, y_true_train))
print("Testing Error P=5: ",calc_error(y_pred_test, y_true_test))
```

```
Training Error P=5:  0.010468319559228650
Testing Error P=5:  0.07387862796833773
```

Question 3

In [13]:

```
sorted_sub = sorted(clf_p_5.substrings.items(), key=lambda x: x[1])
print("Top 2 Substrings: ",sorted_sub[-2:])
```

```
Top 2 Substrings:  [('TAGQE', 3573), ('DTAGQ', 5065)]
```