Introduction to Machine Learning
Assignment 3
Cross Validation

Group 05
Bora Yilmaz (s3903125) & Felix Zailskas (s3918270)

October 27, 2021

Contents

Cross Validation

1. Introduction

Cross validation is used to evaluate how a machine learning model performs under decided constraints and parameters. It works by splitting up the data to train and test a machine learning model into a multitude of subsets. Each of those subsets will be used once as a testing set for the trained model, while all other subsets are used as a training set. This aims to validate the training method used and reduce errors like overfitting. In this report, cross validation will be implemented and then applied over Learning Vector Quantization 1 (LVQ) models. Cross validation aims to evaluate the machine learning model, in this case LVQ prototypes, by training the data set split into subsets for comparison purposes. Each time cross validation is to be applied, the data set will be split into multiple parts, and one of them will act as the testing set. The training will only make use of the training sets, and then evaluate the prototypes with respect to the test set. Through these evaluations, error rates will be determined for the training and the testing. Such error rates are then used to evaluate the model's accuracy, efficiency and to also identify overfitting or underfitting models (prototypes).
The report will mention how the cross validation was integrated to the LVQ1 system in section 2, display the results and their respective parameters in section 3 and then finally going into a discussion on the results and parameter selections in section 4.
The data set used in this application of cross validation to the LVQ consists of 100 data points. Each of the points is a two dimensional feature vector. Exactly half of the data points belong to the first class while the other half belongs to the second class.

2. Method

To apply cross validation to the data set we have to define a number M that describes the amount of subsets the data should be split into. Given this M we can say that each subset D_i has cardinality $|D_i| = \frac{P}{M}$, where P is the amount of data points in the data set (100 in our case).
Now one of these subsets will be defined as the test set while the other $M-1$ sets will be used for training. We then apply the LVQ algorithm using the training set and a set amount of prototypes for each class.
After the last training epoch the training error is recorded. Furthermore, the error on the test set, using the trained prototypes, is evaluated and recorded. Now a different subset D_i will be chosen to be the test set, while all others are combined into the new training set. This process is repeated until all subsets have been used as the test set. The mean value of all training and test errors is used for the further evaluation of the LVQ parameters.
This process can be repeated while changing parameters of the LVQ. This can give insights on good values for the parameters. For this analysis we evaluated both changes in the amount of prototypes per class K, as well as the learning rate η.
For evaluating K we used the values 1-10 for K, with a fixed learning rate of $\eta = 0.002$.
To evaluate η we used the values 0.001-0.02 in steps of 0.001 and a fixed amount of prototypes per class of $K = 1$.

3. Results

3.1. Modifying the Number of Prototypes K

In order to determine a good number of prototypes for the system we will perform cross validation using $M = 5$ and $M = 10$ subsets of the data. For each of those possibilities we will look at 1 to

10 prototypes per class. For all of the results presented in this subsection, the learning rate is set at $\eta = 0.002$.

3.1.1. $M = 5$

First we analyze the case with 5 subsets. Each subset contains 20 data points. Hence, each training set contains 80 data points and each test set contains 20 data points.

After running the LVQ algorithm with cross validation and a set amount of 100 epochs per training iteration we get the following results.

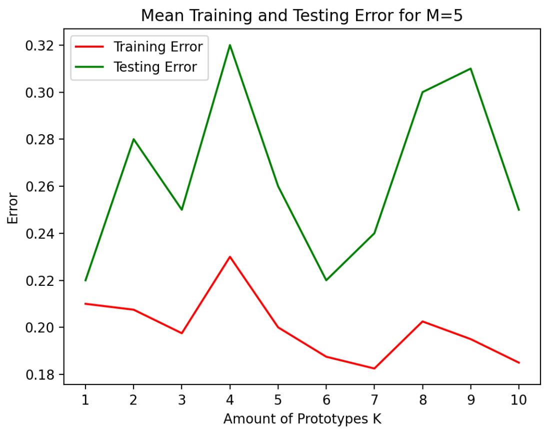

Figure 1: *Mean Training and Test Error for $K \in [1, 10]$, $M = 5$ after 100 epochs.*

In Figure 1 we can see that the mean of the training error gradually decreases, with some small fluctuations for $K = 4$ and $K = 8$. The mean of the testing error is larger than the training error for every K. It has its minima at $K = 1$ and $K = 6$ with a value of 0.22. There are strong fluctuations between different values for K in regard to the mean of the testing error.

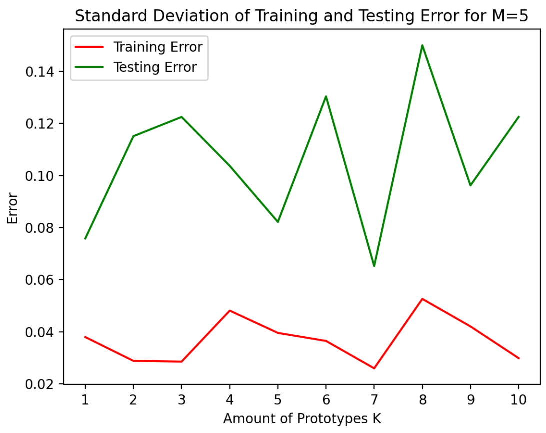

Figure 2: *Standard Deviation of Training and Test Error for $K \in [1, 10]$, $M = 5$ after 100 epochs.*

In Figure 2 we can see that the standard deviation of the training error oscillates around 0.04, with a small amplitude. The standard deviation of the testing error is larger than that of the training error for every K. The standard deviation of the testing error seems to oscillate around 0.10, with a larger amplitude.

To investigate the effect of a larger maximum epoch value, we have rerun this analysis with 700 epochs per training iteration.

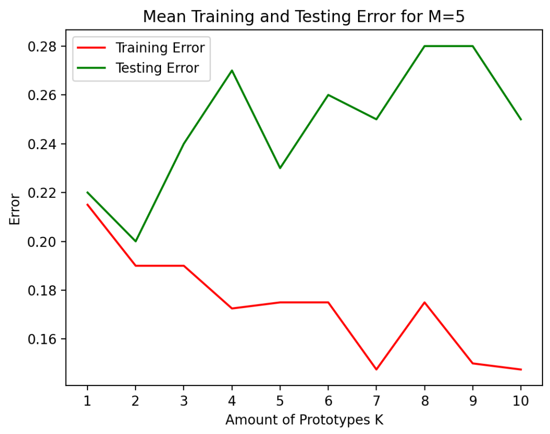

Figure 3: *Mean Training and Test Error for $K \in [1, 10]$, $M = 5$ after 700 epochs.*

In Figure 3 we can see that the mean of the training error constantly decreases, with the exception of $K = 8$. The mean of the testing error however, increases gradually for $K > 2$. This disconnect between training and testing error is an indication of overfitting.

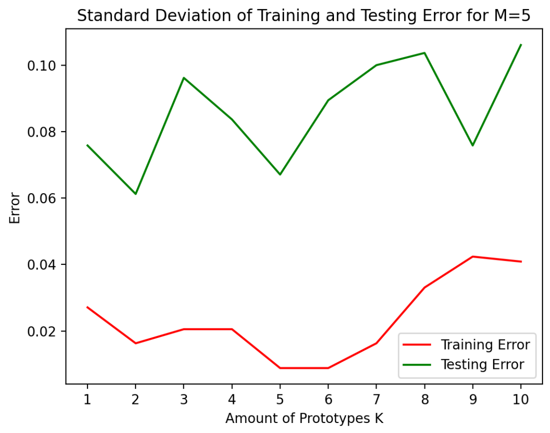

Figure 4: *Standard Deviation of Training and Test Error for $K \in [1, 10]$, $M = 5$ after 700 epochs.*

In Figure 4 we can see that the standard deviation of the training error decreases until $K = 6$. For $K > 6$ the standard deviation of the training error increases. The standard deviation of the testing error is larger than that of the training error for every K. The standard deviation of the testing error seems to oscillate around 0.09, with a medium sized amplitude.

3.1.2. $M = 10$

Now we analyze the case with 10 subsets. Each subset contains 10 data points. Hence, each training set contains 90 data points and each test set contains 10 data points.

After running the LVQ algorithm with cross validation and a set amount of 100 epochs per training iteration we get the following results.

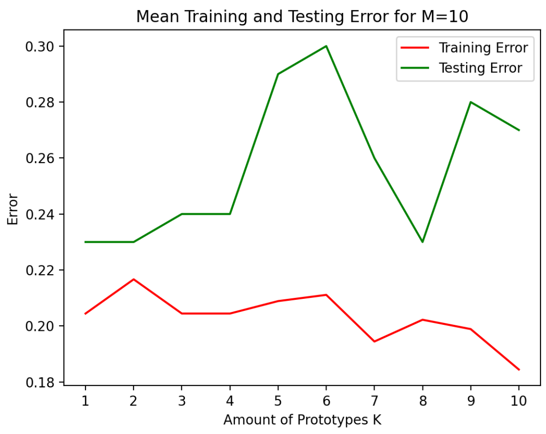

Figure 5: *Mean Training and Test Error for $K \in [1, 10]$, $M = 10$ after 100 epochs.*

In Figure 5 we can see that the mean of the training error gradually decreases with increasing K. The mean of the testing error is larger than the mean of the training error for every K. It has its minima at $K = 1, K = 2$ and $K = 8$ with a value of 0.23. There is a big increase in the mean of the testing error for $K = 5$ and $K = 6$. After reaching a minimum at $K = 8$ the mean of the testing error increases again.

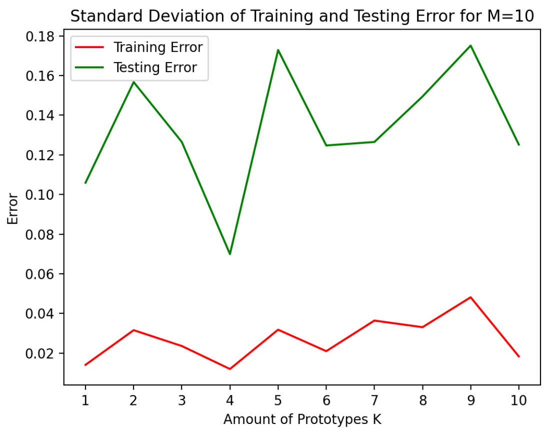

Figure 6: *Standard Deviation of Training and Test Error for $K \in [1, 10]$, $M = 10$ after 100 epochs.*

In Figure 6 we can see that the standard deviation of the training error seems to oscillate around 0.02, with a small amplitude and a small fluctuation around $K = 8$ and $K = 9$. The standard deviation of the testing error is larger than that of the training error for every K. The standard deviation of the testing error seems to oscillate around 0.13, with a large amplitude and a momentary plateau at $K = 6$ and $K = 7$.

3.2. Modifying the Learning Rate η

For all of the results presented in this subsection, the amount of prototypes is set to $K = 1$. The learning rate starts at $\eta = 0.001$ and is increased by 0.001 in each iteration of the algorithm with a maximum of $\eta = 0.02$.

3.2.1. $M = 5$

First we analyze the case with 5 subsets. Each subset contains 20 data points. Hence, each training set contains 80 data points and each test set contains 20 data points.

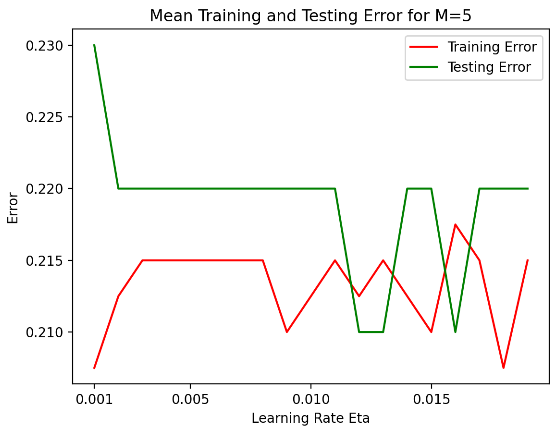

Figure 7: *Mean Training and Test Error for $\eta \in [0.001, 0.02]$, $M = 5$ after 100 epochs.*

In Figure 7 we can see that the mean of the training error starts at its minimum for $\eta = 0.001$, while the mean of the testing error is at its maximum there. The mean of the testing error is constant at 0.22 for a value of η between 0.002 and 0.011. It then fluctuates between 0.22 and 0.21 for η between 0.011 and 0.017. The mean of the training error has a constant value of 0.215 for η between 0.003 and 0.008, and then seems to oscillate around 0.213 with a small amplitude.

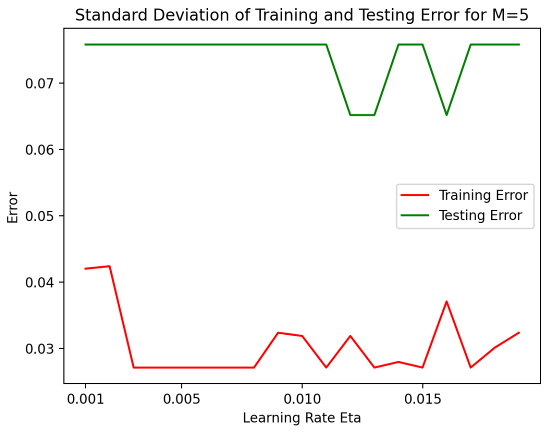

Figure 8: *Standard Deviation of Training and Test Error for $\eta \in [0.001, 0.02]$, $M = 5$ after 100 epochs.*

In Figure 8 we can see that the standard deviation of the testing error stays constant at 0.075

for $0.001 \leq \eta \leq 0.011$. It then fluctuates between 0.075 and 0.065 until reaching its constant value of 0.075 again for $\eta \geq 0.016$. The standard deviation of the learning rate has a sharp decrease until 0.03. It then stays constant at 0.025 for $0.025 \leq \eta \leq 0.008$. For $\eta \geq 0.008$ the standard deviation of the training error seems to be oscillating around 0.03 with a small amplitude.

3.2.2. $M = 10$

Now we analyze the case with 10 subsets. Each subset contains 10 data points. Hence, each training set contains 90 data points and each test set contains 10 data points.

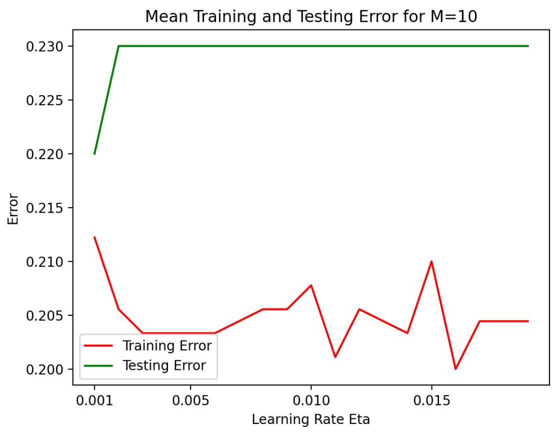

Figure 9: *Mean Training and Test Error for $\eta \in [0.001, 0.02]$, $M = 10$ after 100 epochs.*

In Figure 9 we can see that the mean of the testing error is minimal with a value of 0.22 for $\eta = 0.001$. For all other values of η the mean of the testing error stays constant at 0.23. The mean of the training error decreases slightly for $0.001 \leq \eta \leq 0.005$. It then increases for $0.005 \leq \eta \leq 0.01$, before oscillating around 0.205 with a small amplitude.

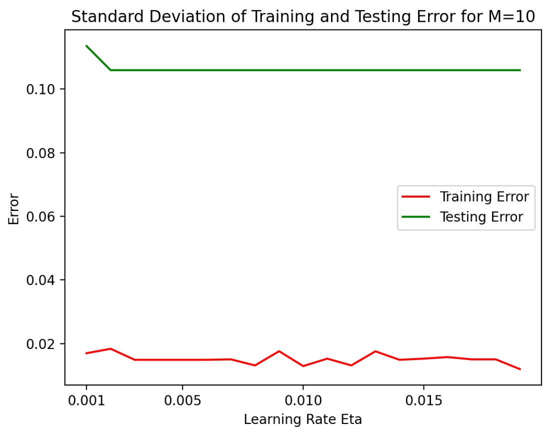

Figure 10: *Standard Deviation of Training and Test Error for $\eta \in [0.001, 0.02]$, $M = 10$ after 100 epochs.*

In Figure 10 we can see that the standard deviation of the testing error stays constant at 0.11 for all values of η. The training error stays almost constant at 0.01, with very little fluctuation, for all values of η.

4. Discussion

One of the the prominent features out of all the figures combined is how the testing error is always greater than the training error. This is the expected outcome since the prototypes are trained on the training data. The prototypes then have a better fit for the training data, and thus give a low error compared to the testing error.
In Figure 3, overfitting can be observed as the testing error stops decreasing and starts increasing (significantly), for $K > 2$, while the training error keeps decreasing. This occurs because the training takes longer (700 epochs), so then the prototypes are tailored too closely to the training set and are therefore overfitted. We can see that with a high number of prototypes and high number of epochs, overfitting will occur eventually because the prototypes will be positioned too specifically with respect to their training data and possible outliers.

4.1. Parameter K

To give a conclusion with respect to the choice of number of prototypes K, various figures need to be compared. Better K options can be deduced simply by looking at the minimal points of testing error. The training error does not tell much with respect to the data set as a whole, which is the actual concern. That is why the minimal points in the testing error are a good deducing point for better K selection. The results from trainings which make use of 100 epochs will be considered since 700 epochs tend to cause overfitting as discussed. For $M = 5$, in Figure 1, $K = 1, 3, 6$ are the best because they give the three minimal testing errors. In particular, significant spikes in testing error are seen in Figure 1. From $K = 4$ to $K = 6$, there is a huge decrease in testing error. This proposes that $K = 6$ could be a really good amount of prototypes. However, at $M = 10$, as seen in Figure 9, $K = 6$ is a bad option in terms of achieving a low testing error. For $M = 10$, as seen from Figure 5, the three minimal testing errors are at $K = 1, 2, 8$. While $K = 8$ was high with smaller M as 5, now it shows a minimum. Based on this, it could be hypothesised that $K = 8$ may provide this low testing error by random chance. As a conclusion on the selection of the parameter K value, overall the values $K = 1, 2, 3$ show low and minimal testing error for both $M = 5$ and $M = 10$. It is tempting to increase both K and M to explore how the cross validation would perform and maybe give better or worse results, yet at the end the cross validation parameters need to be kept simple (low in value), in order to avoid overfitting and infeasible computation times.

4.2. Different η Values

Changing the learning rate parameter η has been explored and the results are seen in Figure 7, Figure 8, Figure 9 and Figure 10. Testing for this parameter via cross validation is helpful since the learning rate parameter is a critical part of the training. It determines how fast and intense (in terms of learning), each training epoch is. For $M = 5$, a learning rate between 0.012 and 0.017 works well because they yield the lowest testing errors as seen in Figure 7. For $M = 10$, a learning rate of 0.001 proved the best while higher learning rates gave the same testing error as seen in Figure 9.

4.3. Standard Deviations of Errors

Generally, a low standard deviation means consistent results for different testing sets. In all figures displaying the standard deviations, the standard deviation of the training error is low and not fluctuating intensely. This is expected since a large amount of the training data does not change when changing the training set. When changing the testing set only $\frac{P}{M}$% of the training set actually changes. Therefore, we expect similar results for these changing training sets. Meanwhile, for the testing error, there are many significant fluctuations, and this is also expected since the models will get better or worse for each different K. When the standard deviations of the two are compared, it is also observed that the training error has a way lower standard deviation in each trial / figure. So in conclusion, while the testing error results are not consistent between different models with different K values, it can be said that the training error stays consistent, with respect to testing especially.

5. Contribution

Both team members attended the lab session for this assignment and both agree that the contribution to this assignment was equal and fair.

5.1. Code

The development of the code for the assignment was done in a lab session. Therefore, the contribution to the code base was entirely equal for both group members.

5.2. Report

Sections were split in half and thus the report was worked on by both members. After looking over the report together one last time, it was finalized and ready.

6. Code Appendix

6.1. cross_validation.py

```
import random
import math
import numpy as np
import pandas as pd
from plotting import *
from matplotlib import pyplot as plt
from calculation import *


def main():
    # Reading in the Data File
    df = pd.read_csv("data/lvqdata.csv", sep=",")

    # Determining the size of the input vectors and the amount of
   vectors
    amt_points, input_dim = df.shape

    # Adding label to the data frame
    labels = np.concatenate((np.full((50, 1), 0, dtype=int), np.full
   ((50, 1), 1, dtype=int)), axis=None)
    df["Label"] = labels
```

```
# Setting learning variables
learning_rate = 0.001
max_learning_rate = 0.02
learning_rate_increase = 0.001
t_max = 100
threshold = 10
amt_protos = 1  # adjust for different prototype amount per class
amt_subsets = 10  # adjust for different M of the cross validation
seed = 1
random.seed(seed)

# Creating the subsets of the data used in the cross validation
# In the case that the data frame cannot be divided equally into
# M subsets then the amount of points n in one subset is rounded
up
df = df.sample(frac=1, random_state=seed).reset_index(drop=True)
n = math.ceil(df.shape[0] / amt_subsets)
list_df = [df[i:i+n] for i in range(0, df.shape[0], n)]

# Data structures to store the different values during the cross
validation
# The error lists will contain tuples (x, y) where x = mean, y =
std dev
# The index i of the lists represents the tuple corresponding to i
 + 1 prototypes
training_error_list = []
testing_error_list = []
prototype_list = []
prototype_trace_list = []
learning_rate_list = []

while learning_rate <= max_learning_rate:
    print(f"Learning Rate = {learning_rate}")
    training_error, test_error, prototypes, prototype_trace = \
        cross_validation(list_df, amt_subsets, amt_protos, t_max,
learning_rate, threshold)
    training_error_list.append(training_error)
    testing_error_list.append(test_error)
    prototype_list.append(prototypes)
    prototype_trace_list.append(prototype_trace)
    learning_rate_list.append(learning_rate)
    learning_rate += learning_rate_increase

print(training_error_list)
print(testing_error_list)

ticks = [0.001, 0.005, 0.01, 0.015, 0.02]
plot_curve(training_error_list, testing_error_list, 0, x_data=
learning_rate_list, x_lab="Learning Rate Eta",
               ticks=ticks ,y_lab="Error", title=f"Mean Training and
    Testing Error for M={amt_subsets}")
    plot_curve(training_error_list, testing_error_list, 1, x_data=
    learning_rate_list, x_lab="Learning Rate Eta",
               ticks=ticks, y_lab="Error", title=f"Standard Deviation
    of Training and Testing Error for M={amt_subsets}")
    plt.show()


main()
```

6.2. CALCULATION.PY

```
import math
import random
import statistics
import pandas as pd


def calc_distance(row, prototype):
    return math.sqrt((row['X']-prototype['X'])**2 + (row['Y']-
    prototype['Y'])**2)


def get_closest_prototype(prototypes, row):
    smallest_dist = float('inf')
    for index, prototype in prototypes.iterrows():
        curr_dist = calc_distance(row, prototype)
        if curr_dist < smallest_dist:
            smallest_dist = curr_dist
            closest_prototype = prototype
            closest_idx = index
    return closest_prototype, closest_idx


def calc_error(df, prototypes):
    fail = 0
    for index, row in df.iterrows():
        closest_prototype, _ = get_closest_prototype(prototypes, row)
        fail += int(closest_prototype['Label']) != int(row['Label'])
    return fail / df.shape[0]


def create_prototypes_random(amt_per_class, df):
    prototypes = pd.DataFrame(columns=['X', 'Y', 'Label'])
    prototype_trace = pd.DataFrame(columns=['trace_X', 'trace_Y'])
    class_zero_df = df.loc[df['Label'] == 0].reset_index(drop=True)
    class_one_df = df.loc[df['Label'] == 1].reset_index(drop=True)
    # create list of possible indexes to ensure different prototypes
    are chosen
    zero_idx_list = list(range(class_zero_df.shape[0]))
    one_idx_list = list(range(class_one_df.shape[0]))
    for i in range(0, amt_per_class):
        idx_zero = random.choice(zero_idx_list)
        idx_one = random.choice(one_idx_list)
        zero_idx_list.remove(idx_zero)
        one_idx_list.remove(idx_one)
        proto_zero = class_zero_df.iloc[[idx_zero]].to_numpy()[0]
        proto_one = class_one_df.iloc[[idx_one]].to_numpy()[0]
        prototypes = prototypes.append({'X': proto_zero[0], 'Y':
   proto_zero[1], 'Label': proto_zero[2]},
                                    ignore_index=True)
        prototype_trace = prototype_trace.append({'trace_X': [
   proto_zero[0]], 'trace_Y': [proto_zero[1]]},
                                              ignore_index=True)
        prototypes = prototypes.append({'X': proto_one[0], 'Y':
   proto_one[1], 'Label': proto_one[2]},
                                    ignore_index=True)
        prototype_trace = prototype_trace.append({'trace_X': [
   proto_one[0]], 'trace_Y': [proto_one[1]]},
                                              ignore_index=True)
    return prototypes, prototype_trace


def cross_validation(list_df, amt_subsets, amt_protos, t_max,
   learning_rate, threshold):
    training_error_list = []
    testing_error_list = []
    prototype_list = []
    prototype_trace_list = []
    for test_set_idx in range(amt_subsets):
        print(f"Test set index = {test_set_idx}")
        # determine which set is the testing set and prepare the
   training set
        test_set = list_df[test_set_idx].reset_index(drop=True)
        training_set = pd.concat(list_df[:test_set_idx] + list_df[
   test_set_idx + 1:], ignore_index=True)
        prototypes, prototype_trace = create_prototypes_random(
   amt_protos, training_set)
        # perform the current training
        train_err, protos, trace = epoch_loop(t_max, prototypes,
   learning_rate, prototype_trace, threshold, training_set)
        # store values
        training_error_list.append(train_err)
        testing_error_list.append(calc_error(test_set, protos))
        prototype_list.append(protos)
        prototype_trace_list.append(trace)
    return (statistics.mean(training_error_list), statistics.stdev(
   training_error_list)),\
           (statistics.mean(testing_error_list), statistics.stdev(
   testing_error_list)),\
           prototype_list, prototype_trace_list
```

```
def epoch_loop(t_max, prototypes, learning_rate, prototype_trace,
    threshold, training_set):
    curr_errors = []
    print("Training...")
    for i in range(0, t_max):
        for index, row in training_set.iterrows():
            # randomize
            training_set = training_set.sample(frac=1).reset_index(
    drop=True)
            # for current point check closest prototype
            closest_prototype, closest_idx = get_closest_prototype(
    prototypes, row)
            # update the prototype
            function = (lambda x, y: x + y) if int(closest_prototype['
    Label']) == int(row['Label']) else (
                lambda x, y: x - y)
            closest_prototype['X'] = function(closest_prototype['X'],
                                              learning_rate * (row['X'
    ] - closest_prototype['X']))
            closest_prototype['Y'] = function(closest_prototype['Y'],
                                              learning_rate * (row['Y'
    ] - closest_prototype['Y']))
            prototype_trace.iloc[closest_idx]['trace_X'].append(
    closest_prototype['X'])
            prototype_trace.iloc[closest_idx]['trace_Y'].append(
    closest_prototype['Y'])
        curr_error = calc_error(training_set, prototypes)
        curr_errors.append(curr_error)
    print("Training complete!", end="\n\n")
    return curr_error, prototypes, prototype_trace
```

6.3. PLOTTING.PY

```
from matplotlib import pyplot as plt


def plot_curve(train, test, data_idx, x_data, ticks=None, title="",
    x_lab="", y_lab=""):
    plt.figure()
    color_train = "red"
    color_test = "green"
    plt.title(title)
    plt.xlabel(x_lab)
    plt.ylabel(y_lab)
    if ticks is not None:
        plt.xticks(ticks)
    plt.plot(x_data, [tup[data_idx] for tup in train], color=
    color_train, label="Training Error")
    plt.plot(x_data, [tup[data_idx] for tup in test], color=color_test,
     label="Testing Error")
    plt.legend()
```