3. Article_norm0_Results

May 8, 2022

1 Incremental margin algorithm for large margin classifiers

[1]:
```
# Imports
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from math import sqrt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report,␣
 ↪accuracy_score, roc_curve, auc
from sklearn import datasets
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import LinearSVC
from sklearn import preprocessing
from sklearn.datasets import load_digits
import time
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.utils import shuffle
from sklearn.model_selection import StratifiedKFold, cross_val_score, KFold
from scipy.stats import sem
from numpy import linalg as LA
from copy import deepcopy
import warnings
warnings.filterwarnings("ignore")
```

1.1 Calculating the margin

[2]:
```
def compute_margin(X, y, w, b):
    margin = []
    for i in range(y.shape[0]):
        margin.append((y[i]*(np.dot(X[i,:], w)+b))/sqrt(sum(w**2)))
    if min(margin) >= 0:
        return min(margin)
    return 0
```

1.2 L0 norm

[3]:
```
def L0_norm(w, threshold):
    l0_norm = 0
    for wi in w:
        if abs(wi) > threshold:
            l0_norm += 1
    return l0_norm
```

2 ELM

2.1 ELM with IM

[4]:
```
import random
from sklearn.base import BaseEstimator, ClassifierMixin
class IM_ELM(BaseEstimator, ClassifierMixin):

    # Inicialization of important parameters
    def __init__(self, n_neurons, eta=0.1, lambda_param=0.01,␣
 ↪delta_margin=10^-3,
                 IMA_iterations=10, max_updates=10000):
        self.n_neurons = n_neurons              # Neurons of hidden layer osf␣
 ↪ELM
        self.eta = eta                          # Learning rate
        self.lambda_param = lambda_param        # Param important of soft␣
 ↪margin
        self.delta_margin = delta_margin        # (1 + delta_margin) * fixed␣
 ↪margin defines the minimum next margin of IMA
        self.IMA_iterations = IMA_iterations    # Maximum number of iterations␣
 ↪of IMA
        self.max_updates = max_updates          # Maximum number of updates in␣
 ↪one execution of FMP
        self.w = np.array([])                   # Vector of weights of the last␣
 ↪layer of the ELM obtained after the training of the IMA
        self.w_elm = np.array([])               # Vector of weights of the last␣
 ↪layer of the ELM obtained after the normal training of ELM
        self.H = np.array([])                   # H matrix of ELM (obtained␣
 ↪with training data)
        self.Z = np.array([])                   # Z matrix of ELM
        self.b = 0

    # Fixed Margin Algorithm
    def FMP_algorithm(self, X, y, w_init, b_init, fixed_margin, idx, s):
        t = 0
        iterations = 0
        w = w_init
```

```
        b = b_init
        norm_w = sqrt(sum(w**2))
        last_t = -1
        lambda_t = 0
        alpha = np.zeros((X.shape[0]))
        while True:
            last_t = t
            e=0
            for k in range(0, y.shape[0]):
                i = int(idx[k])
                if(y[i]*(np.dot(X[i,:], w)+b) <= fixed_margin * norm_w - self.
↪lambda_param * alpha[i]):
                    if norm_w != 0:
                        lambda_t = 1 - (self.eta*fixed_margin)/norm_w
                    else:
                        lambda_t = 1
                    alpha = alpha * lambda_t
                    alpha[i] = alpha[i] + self.eta
                    w = w * lambda_t + self.eta * y[i] * X[i,:]
                    norm_w = sqrt(sum(w**2))
                    b = b + self.eta*y[i]
                    t += 1
                    e += 1
                    if k > s:
                        s += 1
                        j = s
                    else:
                        j=e
                    idx[k], idx[j] = idx[j], idx[k]
            iterations += 1
            if (t > self.max_updates or last_t == t):
                break
        if t<= self.max_updates:
            convergence=1
        else:
            convergence=0
        return w, b, convergence, t, iterations, idx, s

    # IMA Algorithm
    def IM_algorithm(self, X, y):
        self.w = np.zeros(self.w_elm.shape[0])
        self.ws = []
        self.bs = []
        self.ws.append(self.w)
        self.bs.append(self.b)
        fixed_margin = 0#compute_margin(X, y, self.w_elm, self.b)
        t = 0
```

```
        convergence = 1
        updates=0
        iterations=0
        margin=[]
        margin.append(fixed_margin)
        idx = np.linspace(0, y.shape[0]-1, y.shape[0])
        s=0
        while convergence==1 and t<self.IMA_iterations:
            w, b, convergence, updates_, iterations_, idx, s = self.FMP_algorithm(X, y, self.w, self.b, fixed_margin, idx, s)
            if convergence == 1:
                self.w = w
                self.b = b
                self.ws.append(self.w)
                self.bs.append(self.b)
            updates += updates_
            iterations += iterations_
            norm_w = sqrt(sum(self.w**2))
            gamma1 = []
            gamma2 = []
            for i in range(0, y.shape[0]):
                if y[i] == 1:
                    gamma1.append((y[i]*(np.dot(X[i], self.w)+self.b))/norm_w)
                else:
                    gamma2.append((y[i]*(np.dot(X[i], self.w)+self.b))/norm_w)
            gamma1 = np.array(gamma1)
            gamma2 = np.array(gamma2)
            gamma1 = gamma1[gamma1>=0]
            gamma2 = gamma2[gamma2>=0]
            if len(gamma1) == 0:
                min_gamma1 = 0
            else:
                min_gamma1 = min(gamma1)
            if len(gamma2) == 0:
                min_gamma2 = 0
            else:
                min_gamma2 = min(gamma2)
            fixed_margin = max([(min_gamma1 + min_gamma2)/2, (1+self.delta_margin)*fixed_margin])
            margin.append(compute_margin(X, y, self.w, self.b))
            t += 1
        return t, updates, iterations, margin

    # Function that manage the training of IMA ELM
    def fit(self, X, y):
        X_new = np.ones((X.shape[0], X.shape[1]+1))
        X_new[:,1:] = X
```

```
        X = X_new
        n = X.shape[1]
        self.Z = np.array([random.uniform(-0.5, 0.5) for i in range(n*self.
↪n_neurons)]).reshape(n, self.n_neurons)
        self.H = np.tanh(np.dot(X, self.Z))
        w = np.dot(np.linalg.pinv(self.H), y)
        self.w_elm = w.reshape((w.shape[0],))
        iterations_IMA, updates, iterations, margin = self.IM_algorithm(self.H,␣
↪y)
        return iterations_IMA, updates, iterations, margin

    # Function to apply IMA ELM model
    def predict(self, X, use_IMA_w=True):
        X_new = np.ones((X.shape[0], X.shape[1]+1))
        X_new[:,1:] = X
        H = np.tanh(np.dot(X_new, self.Z))
        if use_IMA_w == True:
            y_predicted = np.sign(np.dot(H, self.w) + self.b)
        else:
            y_predicted = np.sign(np.dot(H,  self.w_elm))
        y_predicted[y_predicted==0]=-1
        return y_predicted
```

2.2 ELM with IM P 1

[5]:
```
class IM_ELM_p1(BaseEstimator, ClassifierMixin):

    # Inicialization of important parameters
    def __init__(self, n_neurons, eta=0.1, lambda_param=0.01,␣
↪delta_margin=10^-8,
                 IMA_iterations=10, max_updates=10000):
        self.n_neurons = n_neurons              # Neurons of hidden layer osf␣
↪ELM
        self.eta = eta                          # Learning rate
        self.lambda_param = lambda_param        # Param important of soft␣
↪margin
        self.delta_margin = delta_margin        # (1 + delta_margin) * fixed␣
↪margin defines the minimum next margin of IMA
        self.IMA_iterations = IMA_iterations    # Maximum number of iterations␣
↪of IMA
        self.max_updates = max_updates          # Maximum number of updates in␣
↪one execution of FMP
        self.w = np.array([])                   # Vector of weights of the last␣
↪layer of the ELM obtained after the training of the IMA
        self.w_elm = np.array([])               # Vector of weights of the last␣
↪layer of the ELM obtained after the normal training of ELM
```

```
        self.H = np.array([])                      # H matrix of ELM (obtained␣
↪with training data)
        self.Z = np.array([])                      # Z matrix of ELM
        self.b = 0

    # Fixed Margin Algorithm
    def FMP_algorithm(self, X, y, w_init, b_init, fixed_margin, idx, s):
        t = 0
        iterations = 0
        w = w_init
        b = b_init
        w_norm_inf = LA.norm(w, ord=np.inf)
        last_t = -1
        lambda_t = 0
        alpha = np.zeros((X.shape[0]))
        while True:
            last_t = t
            e=0
            for k in range(0, y.shape[0]):
                i = int(idx[k])
                if(y[i]*(np.dot(X[i,:], w)+b) <= fixed_margin * w_norm_inf -␣
↪self.lambda_param * alpha[i]):
                    if w_norm_inf != 0:
                        lambda_t = 1 - (self.eta*fixed_margin)/w_norm_inf
                    else:
                        lambda_t = 1
                    alpha = alpha * lambda_t
                    alpha[i] = alpha[i] + self.eta
                    for j in range(len(w)):
                        if abs(w[j]) == w_norm_inf:
                            w[j] = w[j] - self.eta * (fixed_margin * np.
↪sign(w[j])/sum(abs(w) == w_norm_inf) - y[i] * X[i,j])
                        elif abs(w[j]) < w_norm_inf:
                            w[j] = w[j] + self.eta * (y[i] * X[i,j])
                    w_norm_inf = LA.norm(w, ord=np.inf)
                    b = b + self.eta*y[i]
                    t += 1
                    e += 1
                    if k > s:
                        s += 1
                        j = s
                    else:
                        j=e
                    idx[k], idx[j] = idx[j], idx[k]
            iterations += 1
            if (t > self.max_updates or last_t == t):
                break
```

```python
        if t<= self.max_updates:
            convergence=1
        else:
            convergence=0
        return w, b, convergence, t, iterations, idx, s

    # IMA Algorithm
    def IM_algorithm(self, X, y):
        self.w = np.zeros(self.H.shape[1])
        w = deepcopy(self.w)
        fixed_margin = 0#compute_margin(X, y, self.w, self.b)
        t = 0
        convergence = 1
        updates=0
        iterations=0
        margin=[]
        l = 0
        idx = np.linspace(0, y.shape[0]-1, y.shape[0])
        s=0
        while convergence==1 and t<self.IMA_iterations:
            w, b, convergence, updates_, iterations_, idx, s = self.
↪FMP_algorithm(X, y, w, self.b, fixed_margin, idx, s)
            if convergence == 1:
                self.w = w
                self.b = b
            updates += updates_
            iterations += iterations_
            norm_w = LA.norm(w, ord=np.inf)
            gamma1 = []
            gamma2 = []
            for i in range(0, y.shape[0]):
                if y[i] == 1:
                    gamma1.append((y[i]*(np.dot(X[i], self.w)+self.b))/norm_w)
                else:
                    gamma2.append((y[i]*(np.dot(X[i], self.w)+self.b))/norm_w)
            if max(gamma1) < 0:
                gamma1.append(0)
            if max(gamma2) < 0:
                gamma2.append(0)
            gamma1 = np.array(gamma1)
            gamma2 = np.array(gamma2)
            gamma1 = gamma1[gamma1>=0]
            gamma2 = gamma2[gamma2>=0]
            if len(gamma1) == 0:
                min_gamma1 = 0
            else:
                min_gamma1 = min(gamma1)
```

```
            if len(gamma2) == 0:
                min_gamma2 = 0
            else:
                min_gamma2 = min(gamma2)
            fixed_margin = max([(min_gamma1 + min_gamma2)/2, (1+self.
↪delta_margin)*fixed_margin])
            #margin.append(compute_margin(X, y, self.w, self.b))
            t += 1
        return t, updates, iterations, margin

    # Function that manage the training of IMA ELM
    def fit(self, X, y, Z=[]):
        X_new = np.ones((X.shape[0], X.shape[1]+1))
        X_new[:,1:] = X
        X = X_new
        n = X.shape[1]
        if len(Z) == 0:
            self.Z = np.array([random.uniform(-0.5, 0.5) for i in range(n*self.
↪n_neurons)]).reshape(n, self.n_neurons)
        else:
            self.Z = Z
        self.H = np.tanh(np.dot(X, self.Z))
        #w = np.dot(np.linalg.pinv(self.H), y)
        #self.w_elm = w.reshape((w.shape[0],))
        iterations_IMA, updates, iterations, margin = self.IM_algorithm(self.H,␣
↪y)
        return iterations_IMA, updates, iterations, margin

    # Function to apply IMA ELM model
    def predict(self, X, use_IMA_w=True):
        X_new = np.ones((X.shape[0], X.shape[1]+1))
        X_new[:,1:] = X
        H = np.tanh(np.dot(X_new, self.Z))
        if use_IMA_w == True:
            y_predicted = np.sign(np.dot(H, self.w) + self.b)
        else:
            y_predicted = np.sign(np.dot(H,  self.w_elm))
        y_predicted[y_predicted==0]=-1
        return y_predicted
```

2.3 ELM with IM P inf

[6]:
```
class IM_ELM_pinf(BaseEstimator, ClassifierMixin):

    # Inicialization of important parameters
```

```
    def __init__(self, n_neurons, eta=0.1, lambda_param=0.01,␣
↪delta_margin=10^-3,
                 IMA_iterations=10, max_updates=10000):
        self.n_neurons = n_neurons                # Neurons of hidden layer osf␣
↪ELM
        self.eta = eta                            # Learning rate
        self.lambda_param = lambda_param          # Param important of soft␣
↪margin
        self.delta_margin = delta_margin          # (1 + delta_margin) * fixed␣
↪margin defines the minimum next margin of IMA
        self.IMA_iterations = IMA_iterations      # Maximum number of iterations␣
↪of IMA
        self.max_updates = max_updates            # Maximum number of updates in␣
↪one execution of FMP
        self.w = np.array([])                     # Vector of weights of the last␣
↪layer of the ELM obtained after the training of the IMA
        self.w_elm = np.array([])                 # Vector of weights of the last␣
↪layer of the ELM obtained after the normal training of ELM
        self.H = np.array([])                     # H matrix of ELM (obtained␣
↪with training data)
        self.Z = np.array([])                     # Z matrix of ELM
        self.b = 0

    # Fixed Margin Algorithm
    def FMP_algorithm(self, X, y, w_init, b_init, fixed_margin, idx, s):
        t = 0
        iterations = 0
        w = w_init
        b = b_init
        w_norm_1 = LA.norm(w, ord=1)
        last_t = -1
        lambda_t = 0
        alpha = np.zeros((X.shape[0]))
        while True:
            last_t = t
            e=0
            for k in range(0, y.shape[0]):
                i = int(idx[k])
                if(y[i]*(np.dot(X[i,:], w)+b) <= fixed_margin * w_norm_1 - self.
↪lambda_param * alpha[i]):
                    if w_norm_1 != 0:
                        lambda_t = 1 - (self.eta*fixed_margin)/w_norm_1
                    else:
                        lambda_t = 1
                    alpha = alpha * lambda_t
                    alpha[i] = alpha[i] + self.eta
```

```
                    w = w - self.eta * (fixed_margin * np.sign(w) - y[i] * X[i,:
↪])
                    w_norm_1 = LA.norm(w, ord=1)
                    b = b + self.eta*y[i]
                    t += 1
                    e += 1
                    if k > s:
                        s += 1
                        j = s
                    else:
                        j=e
                    idx[k], idx[j] = idx[j], idx[k]
            iterations += 1
            if (t > self.max_updates or last_t == t):
                break
        if t<= self.max_updates:
            convergence=1
        else:
            convergence=0
        return w, b, convergence, t, iterations, idx, s

    # IMA Algorithm
    def IM_algorithm(self, X, y):
        self.w = np.zeros(self.H.shape[1])
        self.ws = []
        self.bs = []
        self.ws.append(self.w)
        self.bs.append(self.b)
        fixed_margin = 0#compute_margin(X, y, self.w_elm, self.b)
        t = 0
        convergence = 1
        updates=0
        iterations=0
        margin=[]
        margin.append(fixed_margin)
        idx = np.linspace(0, y.shape[0]-1, y.shape[0])
        s=0
        l=0
        while convergence==1 and t<self.IMA_iterations:
            w, b, convergence, updates_, iterations_, idx, s = self.
↪FMP_algorithm(X, y, self.w, self.b, fixed_margin, idx, s)
            if convergence == 1:
                self.w = w
                self.b = b
                self.ws.append(self.w)
                self.bs.append(self.b)
            updates += updates_
```

```
            iterations += iterations_
            norm_w = LA.norm(w, ord=1)
            gamma1 = []
            gamma2 = []
            for i in range(0, y.shape[0]):
                if y[i] == 1:
                    gamma1.append((y[i]*(np.dot(X[i], self.w)+self.b))/norm_w)
                else:
                    gamma2.append((y[i]*(np.dot(X[i], self.w)+self.b))/norm_w)
            gamma1 = np.array(gamma1)
            gamma2 = np.array(gamma2)
            gamma1 = gamma1[gamma1>=0]
            gamma2 = gamma2[gamma2>=0]
            if len(gamma1) == 0:
                min_gamma1 = 0
            else:
                min_gamma1 = min(gamma1)
            if len(gamma2) == 0:
                min_gamma2 = 0
            else:
                min_gamma2 = min(gamma2)
            fixed_margin = max([(min_gamma1 + min_gamma2)/2, (1+self.
↪delta_margin)*fixed_margin])
            #margin.append(compute_margin(X, y, self.w, self.b))
            t += 1
        return t, updates, iterations, margin

    # Function that manage the training of IMA ELM
    def fit(self, X, y, Z=[]):
        X_new = np.ones((X.shape[0], X.shape[1]+1))
        X_new[:,1:] = X
        X = X_new
        n = X.shape[1]
        if len(Z) == 0:
            self.Z = np.array([random.uniform(-0.5, 0.5) for i in range(n*self.
↪n_neurons)]).reshape(n, self.n_neurons)
        else:
            self.Z = Z
        self.H = np.tanh(np.dot(X, self.Z))
        #w = np.dot(np.linalg.pinv(self.H), y)
        #self.w_elm = w.reshape((w.shape[0],))
        iterations_IMA, updates, iterations, margin = self.IM_algorithm(self.H,␣
↪y)
        return iterations_IMA, updates, iterations, margin

    # Function to apply IMA ELM model
    def predict(self, X, use_IMA_w=True):
```

```
        X_new = np.ones((X.shape[0], X.shape[1]+1))
        X_new[:,1:] = X
        H = np.tanh(np.dot(X_new, self.Z))
        if use_IMA_w == True:
            y_predicted = np.sign(np.dot(H, self.w) + self.b)
        else:
            y_predicted = np.sign(np.dot(H,  self.w_elm))
        y_predicted[y_predicted==0]=-1
        return y_predicted
```

2.4 Commom ELM

[7]:
```
class ELM(BaseEstimator, ClassifierMixin):

    def __init__(self, n_neurons):
        self.n_neurons = n_neurons

    def fit(self, X, y):
        # Adding polarization term
        X_new = np.ones((X.shape[0], X.shape[1]+1))
        X_new[:,1:] = X
        n = X_new.shape[1]
        self.Z = np.array([random.uniform(-0.5, 0.5) for i in range(n*self.
 ↪n_neurons)]).reshape(n, self.n_neurons)
        H = np.tanh(np.dot(X_new, self.Z))
        self.w = np.dot(np.linalg.pinv(H), y)
        return self.w, H, self.Z

    def predict(self, X):
        X_new = np.ones((X.shape[0], X.shape[1]+1))
        X_new[:,1:] = X
        H = np.tanh(np.dot(X_new, self.Z))
        y_predicted = np.sign(np.dot(H, self.w))
        y_predicted[y_predicted==0]=1
        return y_predicted
```

2.5 Function to Capture Results

[8]:
```
def results(X, y, n_splits, p, eta, IMA_iterations):
    # Normalizing data:
    normalizer = MinMaxScaler()
    X = normalizer.fit_transform(X)

    # GridSearch for lambda and learning rate of IMA ELM
    parameters = {'lambda_param':np.linspace(0.01, 10, 50)}
```

```
    clf = IM_ELM(n_neurons=p, delta_margin=10^-3, IMA_iterations=10,␣
↪max_updates=10000)
    clf = GridSearchCV(clf, parameters, scoring='accuracy', cv=5, verbose=0)
    clf.fit(X, y)
    lambda_param = clf.best_params_['lambda_param']

    print(f'Parameters: p={p}, eta={eta}, lambda={lambda_param}')
    # Stratified k fold cross validation
    kf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=72)
    i=0

    train_accuracy_IM_ELM = np.zeros(n_splits)
    test_accuracy_IM_ELM = np.zeros(n_splits)
    margin_IM_ELM = np.zeros(n_splits)
    updates = np.zeros(n_splits)
    iterations_FMP = np.zeros(n_splits)
    iterations_IMA = np.zeros(n_splits)
    norm_L0_IM_ELM_0 = np.zeros(n_splits)
    norm_L0_IM_ELM_1 = np.zeros(n_splits)
    norm_L0_IM_ELM_2 = np.zeros(n_splits)
    norm_L0_IM_ELM_3 = np.zeros(n_splits)
    norm_L0_IM_ELM_4 = np.zeros(n_splits)
    norm_L0_IM_ELM_5 = np.zeros(n_splits)

    train_accuracy_IM_ELM_p1 = np.zeros(n_splits)
    test_accuracy_IM_ELM_p1 = np.zeros(n_splits)
    margin_IM_ELM_p1 = np.zeros(n_splits)
    updates_p1 = np.zeros(n_splits)
    iterations_FMP_p1 = np.zeros(n_splits)
    iterations_IMA_p1 = np.zeros(n_splits)
    norm_L0_IM_ELM_p1_0 = np.zeros(n_splits)
    norm_L0_IM_ELM_p1_1 = np.zeros(n_splits)
    norm_L0_IM_ELM_p1_2 = np.zeros(n_splits)
    norm_L0_IM_ELM_p1_3 = np.zeros(n_splits)
    norm_L0_IM_ELM_p1_4 = np.zeros(n_splits)
    norm_L0_IM_ELM_p1_5 = np.zeros(n_splits)

    train_accuracy_IM_ELM_pinf = np.zeros(n_splits)
    test_accuracy_IM_ELM_pinf = np.zeros(n_splits)
    margin_IM_ELM_pinf = np.zeros(n_splits)
    updates_pinf = np.zeros(n_splits)
    iterations_FMP_pinf = np.zeros(n_splits)
    iterations_IMA_pinf = np.zeros(n_splits)
    norm_L0_IM_ELM_pinf = np.zeros(n_splits)
    norm_L0_IM_ELM_pinf_0 = np.zeros(n_splits)
    norm_L0_IM_ELM_pinf_1 = np.zeros(n_splits)
    norm_L0_IM_ELM_pinf_2 = np.zeros(n_splits)
```

```
    norm_L0_IM_ELM_pinf_3 = np.zeros(n_splits)
    norm_L0_IM_ELM_pinf_4 = np.zeros(n_splits)
    norm_L0_IM_ELM_pinf_5 = np.zeros(n_splits)

    train_accuracy_ELM = np.zeros(n_splits)
    test_accuracy_ELM = np.zeros(n_splits)
    margin_ELM = np.zeros(n_splits)
    norm_L0_ELM = np.zeros(n_splits)
    norm_L0_ELM_0 = np.zeros(n_splits)
    norm_L0_ELM_1 = np.zeros(n_splits)
    norm_L0_ELM_2 = np.zeros(n_splits)
    norm_L0_ELM_3 = np.zeros(n_splits)
    norm_L0_ELM_4 = np.zeros(n_splits)
    norm_L0_ELM_5 = np.zeros(n_splits)

    train_accuracy_SVM = np.zeros(n_splits)
    test_accuracy_SVM = np.zeros(n_splits)
    margin_SVM = np.zeros(n_splits)

    margins=[]
    margins_p1 = []
    margins_pinf = []

    for train_index, test_index in kf.split(X, y):
        X_train = X[train_index,:]
        X_test = X[test_index,:]
        y_train = y[train_index]
        y_test = y[test_index]

        # IM ELM
        clf = IM_ELM(n_neurons=p, eta=eta, lambda_param=lambda_param,␣
↪delta_margin=10^-3, IMA_iterations=IMA_iterations, max_updates=10000)
        iterations_IMA[i], updates[i], iterations_FMP[i], margin = clf.
↪fit(X_train, y_train)
        margins.append(margin)
        y_hat=clf.predict(X_test, use_IMA_w = True)
        y_hat_train=clf.predict(X_train, use_IMA_w = True)
        margin_IM_ELM[i] = compute_margin(clf.H[:,:], y_train, clf.w, clf.b)
        train_accuracy_IM_ELM[i] = accuracy_score(y_train, y_hat_train)
        test_accuracy_IM_ELM[i] = accuracy_score(y_test, y_hat)
        Z = clf.Z
        w = clf.w
        norm_w = LA.norm(w, ord=2)
        w = w/norm_w
        norm_L0_IM_ELM_0[i] = L0_norm(w, 0.2 * w.max())
        norm_L0_IM_ELM_1[i] = L0_norm(w, 0.1 * w.max())
        norm_L0_IM_ELM_2[i] = L0_norm(w, 0.01 * w.max())
```

```
        norm_L0_IM_ELM_3[i] = L0_norm(w, 0.001 * w.max())
        norm_L0_IM_ELM_4[i] = L0_norm(w, 0.0001 * w.max())
        norm_L0_IM_ELM_5[i] = L0_norm(w, 0.00001 * w.max())

        # ELM
        y_hat=clf.predict(X_test, use_IMA_w = False)
        y_hat_train=clf.predict(X_train, use_IMA_w = False)
        margin_ELM[i] = compute_margin(clf.H[:,:], y_train, clf.w_elm, 0)
        train_accuracy_ELM[i] = accuracy_score(y_train, y_hat_train)
        test_accuracy_ELM[i] = accuracy_score(y_test, y_hat)
        w = clf.w_elm
        norm_w = LA.norm(w, ord=2)
        w = w/norm_w
        norm_L0_ELM_0[i] = L0_norm(w, 0.2 * w.max())
        norm_L0_ELM_1[i] = L0_norm(w, 0.1 * w.max())
        norm_L0_ELM_2[i] = L0_norm(w, 0.01 * w.max())
        norm_L0_ELM_3[i] = L0_norm(w, 0.001 * w.max())
        norm_L0_ELM_4[i] = L0_norm(w, 0.0001 * w.max())
        norm_L0_ELM_5[i] = L0_norm(w, 0.00001 * w.max())

         # IM ELM p1
        clf = IM_ELM_p1(n_neurons=p, eta=eta, lambda_param=lambda_param,
 delta_margin=10^-3, IMA_iterations=IMA_iterations, max_updates=10000)
        iterations_IMA_p1[i], updates_p1[i], iterations_FMP_p1[i], margin_p1 =
 clf.fit(X_train, y_train, Z)
        margins_p1.append(margin_p1)
        y_hat=clf.predict(X_test, use_IMA_w = True)
        y_hat_train=clf.predict(X_train, use_IMA_w = True)
        margin_IM_ELM_p1[i] = compute_margin(clf.H[:,:], y_train, clf.w, clf.b)
        train_accuracy_IM_ELM_p1[i] = accuracy_score(y_train, y_hat_train)
        test_accuracy_IM_ELM_p1[i] = accuracy_score(y_test, y_hat)
        w = clf.w
        norm_w = LA.norm(w, ord=2)
        w = w/norm_w
        norm_L0_IM_ELM_p1_0[i] = L0_norm(w, 0.2 * w.max())
        norm_L0_IM_ELM_p1_1[i] = L0_norm(w, 0.1 * w.max())
        norm_L0_IM_ELM_p1_2[i] = L0_norm(w, 0.01 * w.max())
        norm_L0_IM_ELM_p1_3[i] = L0_norm(w, 0.001 * w.max())
        norm_L0_IM_ELM_p1_4[i] = L0_norm(w, 0.0001 * w.max())
        norm_L0_IM_ELM_p1_5[i] = L0_norm(w, 0.00001 * w.max())

        # IM ELM p inf
        clf = IM_ELM_pinf(n_neurons=p, eta=eta, lambda_param=lambda_param,
 delta_margin=10^-3, IMA_iterations=IMA_iterations, max_updates=10000)
        iterations_IMA_pinf[i], updates_pinf[i], iterations_FMP_pinf[i],
 margin_pinf = clf.fit(X_train, y_train, Z)
        margins_pinf.append(margin_pinf)
```

```
        y_hat=clf.predict(X_test, use_IMA_w = True)
        y_hat_train=clf.predict(X_train, use_IMA_w = True)
        margin_IM_ELM_pinf[i] = compute_margin(clf.H[:,:], y_train, clf.w, clf.b)
        train_accuracy_IM_ELM_pinf[i] = accuracy_score(y_train, y_hat_train)
        test_accuracy_IM_ELM_pinf[i] = accuracy_score(y_test, y_hat)
        w = clf.w
        norm_w = LA.norm(w, ord=2)
        w = w/norm_w
        norm_L0_IM_ELM_pinf_0[i] = L0_norm(w, 0.2 * w.max())
        norm_L0_IM_ELM_pinf_1[i] = L0_norm(w, 0.1 * w.max())
        norm_L0_IM_ELM_pinf_2[i] = L0_norm(w, 0.01 * w.max())
        norm_L0_IM_ELM_pinf_3[i] = L0_norm(w, 0.001 * w.max())
        norm_L0_IM_ELM_pinf_4[i] = L0_norm(w, 0.0001 * w.max())
        norm_L0_IM_ELM_pinf_5[i] = L0_norm(w, 0.00001 * w.max())
        i+=1

    print("********* Results IM ELM p=2 **************")
    print("Acc train: " + '{:.4f}'.format(train_accuracy_IM_ELM.mean())+ "+/-" + '{:.4f}'.format(train_accuracy_IM_ELM.std()))
    print("Acc test: " + '{:.4f}'.format(test_accuracy_IM_ELM.mean()) + "+/-" + '{:.4f}'.format(test_accuracy_IM_ELM.std()))
    print("Iterations: " + '{:.4f}'.format(iterations_FMP.mean())+ "+/-" + '{:.4f}'.format(iterations_FMP.std()))
    print("Iterations IMA: " + '{:.4f}'.format(iterations_IMA.mean())+ "+/-" + '{:.4f}'.format(iterations_IMA.std()))
    print("Updates: " + '{:.4f}'.format(updates.mean())+ "+/-" + '{:.4f}'.format(updates.std()))
    print("Margin: " + '{:.9f}'.format(margin_IM_ELM.mean())+ "+/-" + '{:.9f}'.format(margin_IM_ELM.std()))
    print("Norm L0 (20%): " + '{:.9f}'.format(norm_L0_IM_ELM_0.mean())+ "+/-" + '{:.9f}'.format(norm_L0_IM_ELM_0.std()))
    print("Norm L0 (10%): " + '{:.9f}'.format(norm_L0_IM_ELM_1.mean())+ "+/-" + '{:.9f}'.format(norm_L0_IM_ELM_1.std()))
    print("Norm L0 (1%): " + '{:.9f}'.format(norm_L0_IM_ELM_2.mean())+ "+/-" + '{:.9f}'.format(norm_L0_IM_ELM_2.std()))
    print("Norm L0 (0.1%): " + '{:.9f}'.format(norm_L0_IM_ELM_3.mean())+ "+/-" + '{:.9f}'.format(norm_L0_IM_ELM_3.std()))
    print("Norm L0 (0.01%): " + '{:.9f}'.format(norm_L0_IM_ELM_4.mean())+ "+/-" + '{:.9f}'.format(norm_L0_IM_ELM_4.std()))
    print("Norm L0 (0.001%): " + '{:.9f}'.format(norm_L0_IM_ELM_5.mean())+ "+/-" + '{:.9f}'.format(norm_L0_IM_ELM_5.std()))

    print("********* Results IM ELM p=1**************")
    print("Acc train: " + '{:.4f}'.format(train_accuracy_IM_ELM_p1.mean())+ "+/-" + '{:.4f}'.format(train_accuracy_IM_ELM_p1.std()))
```

```
    print("Acc test: " + '{:.4f}'.format(test_accuracy_IM_ELM_p1.mean()) + "+/-" + '{:.4f}'.format(test_accuracy_IM_ELM_p1.std()))
    print("Iterations: " + '{:.4f}'.format(iterations_FMP_p1.mean())+ "+/-" + '{:.4f}'.format(iterations_FMP_p1.std()))
    print("Iterations IMA: " + '{:.4f}'.format(iterations_IMA_p1.mean())+ "+/-" + '{:.4f}'.format(iterations_IMA_p1.std()))
    print("Updates: " + '{:.4f}'.format(updates_p1.mean())+ "+/-" + '{:.4f}'.format(updates_p1.std()))
    print("Margin: " + '{:.9f}'.format(margin_IM_ELM_p1.mean())+ "+/-" + '{:.9f}'.format(margin_IM_ELM_p1.std()))
    print("Norm L0 (20%): " + '{:.9f}'.format(norm_L0_IM_ELM_p1_0.mean())+ "+/-" + '{:.9f}'.format(norm_L0_IM_ELM_p1_0.std()))
    print("Norm L0 (10%): " + '{:.9f}'.format(norm_L0_IM_ELM_p1_1.mean())+ "+/-" + '{:.9f}'.format(norm_L0_IM_ELM_p1_1.std()))
    print("Norm L0 (1%): " + '{:.9f}'.format(norm_L0_IM_ELM_p1_2.mean())+ "+/-" + '{:.9f}'.format(norm_L0_IM_ELM_p1_2.std()))
    print("Norm L0 (0.1%): " + '{:.9f}'.format(norm_L0_IM_ELM_p1_3.mean())+ "+/-" + '{:.9f}'.format(norm_L0_IM_ELM_p1_3.std()))
    print("Norm L0 (0.01%): " + '{:.9f}'.format(norm_L0_IM_ELM_p1_4.mean())+ "+/-" + '{:.9f}'.format(norm_L0_IM_ELM_p1_4.std()))
    print("Norm L0 (0.001%): " + '{:.9f}'.format(norm_L0_IM_ELM_p1_5.mean())+ "+/-" + '{:.9f}'.format(norm_L0_IM_ELM_p1_5.std()))

    print("********* Results IM ELM p=inf**************")
    print("Acc train: " + '{:.4f}'.format(train_accuracy_IM_ELM_pinf.mean())+ "+/-" + '{:.4f}'.format(train_accuracy_IM_ELM_pinf.std()))
    print("Acc test: " + '{:.4f}'.format(test_accuracy_IM_ELM_pinf.mean()) + "+/-" + '{:.4f}'.format(test_accuracy_IM_ELM_pinf.std()))
    print("Iterations: " + '{:.4f}'.format(iterations_FMP_pinf.mean())+ "+/-" + '{:.4f}'.format(iterations_FMP_pinf.std()))
    print("Iterations IMA: " + '{:.4f}'.format(iterations_IMA_pinf.mean())+ "+/-" + '{:.4f}'.format(iterations_IMA_pinf.std()))
    print("Updates: " + '{:.4f}'.format(updates_pinf.mean())+ "+/-" + '{:.4f}'.format(updates_pinf.std()))
    print("Margin: " + '{:.9f}'.format(margin_IM_ELM_pinf.mean())+ "+/-" + '{:.9f}'.format(margin_IM_ELM_pinf.std()))
    print("Norm L0 (20%): " + '{:.9f}'.format(norm_L0_IM_ELM_pinf_0.mean())+ "+/-" + '{:.9f}'.format(norm_L0_IM_ELM_pinf_0.std()))
    print("Norm L0 (10%): " + '{:.9f}'.format(norm_L0_IM_ELM_pinf_1.mean())+ "+/-" + '{:.9f}'.format(norm_L0_IM_ELM_pinf_1.std()))
    print("Norm L0 (1%): " + '{:.9f}'.format(norm_L0_IM_ELM_pinf_2.mean())+ "+/-" + '{:.9f}'.format(norm_L0_IM_ELM_pinf_2.std()))
    print("Norm L0 (0.1%): " + '{:.9f}'.format(norm_L0_IM_ELM_pinf_3.mean())+ "+/-" + '{:.9f}'.format(norm_L0_IM_ELM_pinf_3.std()))
```

```
    print("Norm L0 (0.01%): " + '{:.9f}'.format(norm_L0_IM_ELM_pinf_4.mean())+␣
↪"+/-" + '{:.9f}'.format(norm_L0_IM_ELM_pinf_4.std()))
    print("Norm L0 (0.001%): " + '{:.9f}'.format(norm_L0_IM_ELM_pinf_5.mean())+␣
↪"+/-" + '{:.9f}'.format(norm_L0_IM_ELM_pinf_5.std()))

    print("********* Results ELM **************")
    print("Acc train: " + '{:.4f}'.format(train_accuracy_ELM.mean())+ "+/-" +␣
↪'{:.4f}'.format(train_accuracy_ELM.std()))
    print("Acc test: " + '{:.4f}'.format(test_accuracy_ELM.mean()) + "+/-" + '{:
↪.4f}'.format(test_accuracy_ELM.std()))
    print("Margin: " + '{:.9f}'.format(margin_ELM.mean())+ "+/-" + '{:.9f}'.
↪format(margin_ELM.std()))
    print("Norm L0 (20%): " + '{:.9f}'.format(norm_L0_ELM_0.mean())+ "+/-" + '{:
↪.9f}'.format(norm_L0_ELM_0.std()))
    print("Norm L0 (10%): " + '{:.9f}'.format(norm_L0_ELM_1.mean())+ "+/-" + '{:
↪.9f}'.format(norm_L0_ELM_1.std()))
    print("Norm L0 (1%): " + '{:.9f}'.format(norm_L0_ELM_2.mean())+ "+/-" + '{:.
↪9f}'.format(norm_L0_ELM_2.std()))
    print("Norm L0 (0.1%): " + '{:.9f}'.format(norm_L0_ELM_3.mean())+ "+/-" +␣
↪'{:.9f}'.format(norm_L0_ELM_3.std()))
    print("Norm L0 (0.01%): " + '{:.9f}'.format(norm_L0_ELM_4.mean())+ "+/-" +␣
↪'{:.9f}'.format(norm_L0_ELM_4.std()))
    print("Norm L0 (0.001%): " + '{:.9f}'.format(norm_L0_ELM_5.mean())+ "+/-" +␣
↪'{:.9f}'.format(norm_L0_ELM_5.std()))
    return margins
```

2.6 Function to Plot Margin Evolution

[24]:
```
def plot_margin_evolution(m):
    avg_margins = []
    sem_margins = []
    for i in range(len(m[0])):
        margins = []
        for j in range(len(m)):
            margins.append(m[j][i])
        avg_margins.append(np.mean(margins))
        sem_margins.append(sem(margins))
    x = np.array(range(len(avg_margins)))
    plt.figure(1)
    plt.plot(x, avg_margins)
    plt.grid()
    plt.figure(2)
    plt.errorbar(x, avg_margins, sem_margins, color = 'blue', marker='s',␣
↪capsize=5)
    plt.grid()
```

2.7 Application on Iris Dataset

[25]:
```
iris = datasets.load_iris()
X = iris.data
# setosa - 0, versicolor - 1, virginica - 2
y = iris.target
# O problema agora possui apenas as classes y=-1 e y=1
y[y>0] = 1
y[y==0] = -1
m = results(X, y, 10, 150, 0.1, 20)
#plot_margin_evolution(m)
```

```
Parameters: p=150, eta=0.1, lambda=0.01
********* Results IM ELM p=2 **************
Acc train: 1.0000+/-0.0000
Acc test: 1.0000+/-0.0000
Iterations: 2437.1000+/-1835.1254
Iterations IMA: 18.5000+/-4.5000
Updates: 6609.8000+/-4835.3612
Margin: 0.690434230+/-0.032543713
Norm L0 (20%): 97.100000000+/-4.887739764
Norm L0 (10%): 121.900000000+/-4.060788101
Norm L0 (1%): 147.600000000+/-1.496662955
Norm L0 (0.1%): 149.500000000+/-0.806225775
Norm L0 (0.01%): 150.000000000+/-0.000000000
Norm L0 (0.001%): 150.000000000+/-0.000000000
********* Results IM ELM p=1**************
Acc train: 1.0000+/-0.0000
Acc test: 1.0000+/-0.0000
Iterations: 804.4000+/-247.9569
Iterations IMA: 20.0000+/-0.0000
Updates: 1934.3000+/-535.4497
Margin: 0.583975169+/-0.032138315
Norm L0 (20%): 145.100000000+/-1.135781669
Norm L0 (10%): 147.400000000+/-1.562049935
Norm L0 (1%): 149.800000000+/-0.600000000
Norm L0 (0.1%): 150.000000000+/-0.000000000
Norm L0 (0.01%): 150.000000000+/-0.000000000
Norm L0 (0.001%): 150.000000000+/-0.000000000
********* Results IM ELM p=inf**************
Acc train: 1.0000+/-0.0000
Acc test: 1.0000+/-0.0000
Iterations: 1156.2000+/-1813.7026
Iterations IMA: 20.0000+/-0.0000
Updates: 3634.4000+/-6300.7749
Margin: 0.362088676+/-0.068720334
Norm L0 (20%): 10.500000000+/-4.432832052
Norm L0 (10%): 13.000000000+/-5.656854249
```

```
Norm L0 (1%): 24.000000000+/-14.839137441
Norm L0 (0.1%): 106.800000000+/-27.805754800
Norm L0 (0.01%): 144.700000000+/-5.848931526
Norm L0 (0.001%): 149.100000000+/-1.513274595
********* Results ELM **************
Acc train: 1.0000+/-0.0000
Acc test: 0.9867+/-0.0267
Margin: 0.000000550+/-0.000000139
Norm L0 (20%): 59.600000000+/-10.442221986
Norm L0 (10%): 89.700000000+/-9.434511116
Norm L0 (1%): 141.000000000+/-4.289522118
Norm L0 (0.1%): 148.700000000+/-1.417744688
Norm L0 (0.01%): 150.000000000+/-0.000000000
Norm L0 (0.001%): 150.000000000+/-0.000000000
```

2.8 Application on Synthetic Dataset

[26]:
```python
synthetic_dataset = pd.read_csv('data/synthetic_dataset/synthetic_control.
↪data', sep="\s+",  header=None, engine='python')
X = synthetic_dataset.to_numpy()
y = np.concatenate((np.ones(100), np.ones(200)*-1, np.ones(100), np.
↪ones(100)*-1,np.ones(100)))
m = results(X, y, 10, 600, 0.1, 20)
#plot_margin_evolution(m)
```

```
Parameters: p=600, eta=0.1, lambda=0.01
********* Results IM ELM p=2 **************
Acc train: 1.0000+/-0.0000
Acc test: 0.9950+/-0.0107
Iterations: 1320.0000+/-517.3517
Iterations IMA: 19.4000+/-1.2806
Updates: 14008.7000+/-5452.8292
Margin: 0.432481084+/-0.020309761
Norm L0 (20%): 253.300000000+/-49.091852685
Norm L0 (10%): 417.200000000+/-36.138068570
Norm L0 (1%): 580.500000000+/-4.588027899
Norm L0 (0.1%): 597.800000000+/-1.248999600
Norm L0 (0.01%): 600.000000000+/-0.000000000
Norm L0 (0.001%): 600.000000000+/-0.000000000
********* Results IM ELM p=1**************
Acc train: 1.0000+/-0.0000
Acc test: 0.9950+/-0.0107
Iterations: 478.0000+/-277.0498
Iterations IMA: 20.0000+/-0.0000
Updates: 4479.0000+/-2345.7443
Margin: 0.335656036+/-0.019141857
Norm L0 (20%): 532.500000000+/-15.692354826
```

```
Norm L0 (10%): 566.900000000+/-8.080222769
Norm L0 (1%): 595.900000000+/-2.071231518
Norm L0 (0.1%): 600.000000000+/-0.000000000
Norm L0 (0.01%): 600.000000000+/-0.000000000
Norm L0 (0.001%): 600.000000000+/-0.000000000
********* Results IM ELM p=inf**************
Acc train: 1.0000+/-0.0000
Acc test: 0.9967+/-0.0067
Iterations: 439.3000+/-144.3579
Iterations IMA: 20.0000+/-0.0000
Updates: 4421.5000+/-1290.3511
Margin: 0.312057376+/-0.016792737
Norm L0 (20%): 76.400000000+/-18.078716769
Norm L0 (10%): 107.500000000+/-17.517134469
Norm L0 (1%): 184.100000000+/-30.768327871
Norm L0 (0.1%): 520.100000000+/-20.767522722
Norm L0 (0.01%): 591.400000000+/-4.176122604
Norm L0 (0.001%): 598.900000000+/-1.135781669
********* Results ELM **************
Acc train: 1.0000+/-0.0000
Acc test: 0.7733+/-0.0624
Margin: 0.014480311+/-0.001484165
Norm L0 (20%): 244.100000000+/-52.947993352
Norm L0 (10%): 399.600000000+/-38.134498817
Norm L0 (1%): 578.700000000+/-6.372597587
Norm L0 (0.1%): 597.600000000+/-1.200000000
Norm L0 (0.01%): 599.800000000+/-0.600000000
Norm L0 (0.001%): 600.000000000+/-0.000000000
```

2.9 Application on Robot Dataset

[27]:
```python
robot_dataset = pd.read_csv('data/robot/lp4_data.csv', delimiter =',')
X = robot_dataset.to_numpy().reshape([117,90])
y = np.concatenate((np.ones(24), np.ones(117-24)*-1))
m = results(X, y, 10, 117, 0.1, 20)
#plot_margin_evolution(m)
```

```
Parameters: p=117, eta=0.1, lambda=1.437142857142857
********* Results IM ELM p=2 **************
Acc train: 0.8082+/-0.0452
Acc test: 0.8205+/-0.0690
Iterations: 370.2000+/-85.4854
Iterations IMA: 7.5000+/-6.5154
Updates: 11500.6000+/-3235.4924
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 54.700000000+/-15.912573645
Norm L0 (10%): 91.600000000+/-7.459222480
```

```
Norm L0 (1%): 113.900000000+/-1.813835715
Norm L0 (0.1%): 116.800000000+/-0.600000000
Norm L0 (0.01%): 117.000000000+/-0.000000000
Norm L0 (0.001%): 117.000000000+/-0.000000000
********* Results IM ELM p=1**************
Acc train: 0.8233+/-0.0577
Acc test: 0.8114+/-0.0745
Iterations: 337.0000+/-85.9511
Iterations IMA: 6.7000+/-7.0718
Updates: 10677.2000+/-3199.6270
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 101.400000000+/-12.167168939
Norm L0 (10%): 109.500000000+/-5.315072906
Norm L0 (1%): 116.300000000+/-0.900000000
Norm L0 (0.1%): 116.900000000+/-0.300000000
Norm L0 (0.01%): 117.000000000+/-0.000000000
Norm L0 (0.001%): 117.000000000+/-0.000000000
********* Results IM ELM p=inf**************
Acc train: 0.8063+/-0.0362
Acc test: 0.7689+/-0.0570
Iterations: 432.0000+/-179.4252
Iterations IMA: 13.6000+/-6.6663
Updates: 13534.7000+/-5613.1441
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 13.400000000+/-10.983624174
Norm L0 (10%): 27.000000000+/-19.884667460
Norm L0 (1%): 97.200000000+/-9.303762680
Norm L0 (0.1%): 115.000000000+/-1.264911064
Norm L0 (0.01%): 116.800000000+/-0.400000000
Norm L0 (0.001%): 117.000000000+/-0.000000000
********* Results ELM **************
Acc train: 1.0000+/-0.0000
Acc test: 0.5273+/-0.2028
Margin: 0.000141341+/-0.000073803
Norm L0 (20%): 17.000000000+/-4.939635614
Norm L0 (10%): 31.900000000+/-8.030566605
Norm L0 (1%): 96.900000000+/-9.267685795
Norm L0 (0.1%): 114.400000000+/-1.959591794
Norm L0 (0.01%): 116.800000000+/-0.400000000
Norm L0 (0.001%): 117.000000000+/-0.000000000
```

2.10 Application on Mushroom Dataset

[13]:
```
df = pd.read_csv('data/Mushroom/agaricus-lepiota.data', delimiter =',',␣
 ↪header=None)
df = df.replace("?", np.nan)
df = df.dropna()
```

```python
y = df[0].to_numpy()
X = df.drop([0], axis='columns')
X = pd.get_dummies(X).to_numpy()
y[np.where(y=='e')] = -1
y[np.where(y=='p')] = 1
y = np.array(y.tolist())
run(X, y)
```

```
Experimento com 1000 neurônios:
Parameters: p=1000, eta=0.1, lambda=4.0875510204081635, C=0.04291934260128778
********* Results ELM-IMA **************
Acc train: 1.0000+/-0.0000
Acc test: 0.9998+/-0.0005
Iterations: 62.9000+/-7.7389
Iterations IMA: 20.0000+/-0.0000
Updates: 1357.7000+/-397.6848
Margin: 0.049897723+/-0.022380742
********* Results ELM-IMA Beta **************
Acc train: 1.0000+/-0.0000
Acc test: 0.9998+/-0.0005
Iterations: 60.0000+/-4.7117
Iterations IMA: 20.0000+/-0.0000
Updates: 1504.2000+/-473.8898
Margin: 0.065234755+/-0.007324467
********* Results ELM **************
Acc train: 1.0000+/-0.0000
Acc test: 1.0000+/-0.0000
Margin: 0.058185186+/-0.002137417
********* Results SVM **************
Acc train: 0.9990+/-0.0005
Acc test: 0.9972+/-0.0016
Margin: 0.000000+/-0.000000



Experimento com 500 neurônios:
Parameters: p=500, eta=0.1, lambda=4.291428571428571, C=0.04291934260128778
********* Results ELM-IMA **************
Acc train: 1.0000+/-0.0001
Acc test: 0.9998+/-0.0005
Iterations: 87.5000+/-50.2678
Iterations IMA: 20.0000+/-0.0000
Updates: 1629.0000+/-1125.0265
Margin: 0.039336680+/-0.022962223
********* Results ELM-IMA Beta **************
Acc train: 1.0000+/-0.0000
Acc test: 1.0000+/-0.0000
```

```
Iterations: 77.3000+/-48.9245
Iterations IMA: 20.0000+/-0.0000
Updates: 1302.6000+/-873.6290
Margin: 0.040345545+/-0.010903189
********* Results ELM **************
Acc train: 1.0000+/-0.0000
Acc test: 1.0000+/-0.0000
Margin: 0.033250171+/-0.002345337
********* Results SVM **************
Acc train: 0.9990+/-0.0005
Acc test: 0.9972+/-0.0016
Margin: 0.000000+/-0.000000



Experimento com 333 neurônios:
Parameters: p=333, eta=0.1, lambda=4.495306122448979, C=0.04291934260128778
********* Results ELM-IMA **************
Acc train: 1.0000+/-0.0000
Acc test: 1.0000+/-0.0000
Iterations: 78.9000+/-42.8450
Iterations IMA: 20.0000+/-0.0000
Updates: 1911.3000+/-1406.6034
Margin: 0.028549551+/-0.011459962
********* Results ELM-IMA Beta **************
Acc train: 1.0000+/-0.0000
Acc test: 1.0000+/-0.0000
Iterations: 63.1000+/-21.5149
Iterations IMA: 20.0000+/-0.0000
Updates: 1272.6000+/-945.6805
Margin: 0.034326184+/-0.006097951
********* Results ELM **************
Acc train: 1.0000+/-0.0000
Acc test: 1.0000+/-0.0000
Margin: 0.032539105+/-0.003072349
********* Results SVM **************
Acc train: 0.9990+/-0.0005
Acc test: 0.9972+/-0.0016
Margin: 0.000000+/-0.000000
```

[20]:

```
df = pd.read_csv('data/Mushroom/agaricus-lepiota.data', delimiter =',',␣
  ↪header=None)
df = df.replace("?", np.nan)
df = df.dropna()
```

```python
y = df[0].to_numpy()
X = df.drop([0], axis='columns')
X = pd.get_dummies(X).to_numpy()
y[np.where(y=='e')] = -1
y[np.where(y=='p')] = 1
y = np.array(y.tolist())
m = results(X, y, 10, 1000, 0.1, 20)
#plot_margin_evolution(m)
```

```
Parameters: p=1000, eta=0.1, lambda=2.4565306122448978
********* Results IM ELM p=2 **************
Acc train: 1.0000+/-0.0000
Acc test: 1.0000+/-0.0000
Iterations: 309.9000+/-63.1798
Iterations IMA: 20.0000+/-0.0000
Updates: 6654.8000+/-2022.3394
Margin: 1.706531914+/-0.074724859
Norm L0 (20%): 450.800000000+/-57.606944026
Norm L0 (10%): 696.800000000+/-35.176128269
Norm L0 (1%): 969.100000000+/-5.262128847
Norm L0 (0.1%): 996.800000000+/-2.088061302
Norm L0 (0.01%): 999.700000000+/-0.458257569
Norm L0 (0.001%): 1000.000000000+/-0.000000000
********* Results IM ELM p=1**************
Acc train: 1.0000+/-0.0000
Acc test: 1.0000+/-0.0000
Iterations: 138.1000+/-32.9983
Iterations IMA: 20.0000+/-0.0000
Updates: 1809.8000+/-736.4398
Margin: 1.412366507+/-0.055633771
Norm L0 (20%): 876.500000000+/-18.023595646
Norm L0 (10%): 936.700000000+/-10.668176976
Norm L0 (1%): 992.400000000+/-2.289104628
Norm L0 (0.1%): 999.300000000+/-0.900000000
Norm L0 (0.01%): 1000.000000000+/-0.000000000
Norm L0 (0.001%): 1000.000000000+/-0.000000000
********* Results IM ELM p=inf**************
Acc train: 1.0000+/-0.0000
Acc test: 1.0000+/-0.0000
Iterations: 165.3000+/-32.1685
Iterations IMA: 20.0000+/-0.0000
Updates: 3014.3000+/-981.1604
Margin: 1.077656298+/-0.062419271
Norm L0 (20%): 77.600000000+/-16.704490414
Norm L0 (10%): 116.300000000+/-19.334166649
Norm L0 (1%): 196.500000000+/-35.180250141
Norm L0 (0.1%): 779.500000000+/-87.091044316
```

```
Norm L0 (0.01%): 976.300000000+/-11.234322409
Norm L0 (0.001%): 997.700000000+/-1.676305461
********* Results ELM **************
Acc train: 1.0000+/-0.0000
Acc test: 1.0000+/-0.0000
Margin: 0.887007038+/-0.060240913
Norm L0 (20%): 202.900000000+/-99.442898188
Norm L0 (10%): 461.000000000+/-135.871998587
Norm L0 (1%): 936.900000000+/-21.177582487
Norm L0 (0.1%): 993.900000000+/-3.300000000
Norm L0 (0.01%): 999.400000000+/-0.489897949
Norm L0 (0.001%): 1000.000000000+/-0.000000000
```

2.11 Application on Ionosphere Dataset

[9]:
```
ionosphere_dataset = pd.read_csv('data/Ionosphere/ionosphere.data',␣
 ↪names=list(range(0,35)), sep=',')
y = ionosphere_dataset[34].to_numpy()
X = ionosphere_dataset.drop([34], axis='columns').to_numpy()
y[np.where(y=='g')] = 1
y[np.where(y=='b')] = -1
y = np.array(y.tolist())
m = results(X, y, 10, 351, 0.1, 20)
#plot_margin_evolution(m)
```

```
Parameters: p=351, eta=0.1, lambda=3.2720408163265304
********* Results IM ELM p=2 **************
Acc train: 0.9883+/-0.0058
Acc test: 0.9116+/-0.0606
Iterations: 461.9000+/-54.8606
Iterations IMA: 20.0000+/-0.0000
Updates: 6004.5000+/-707.7275
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 205.500000000+/-22.983689869
Norm L0 (10%): 277.300000000+/-15.691080269
Norm L0 (1%): 342.200000000+/-4.237924020
Norm L0 (0.1%): 350.100000000+/-0.943398113
Norm L0 (0.01%): 350.800000000+/-0.400000000
Norm L0 (0.001%): 351.000000000+/-0.000000000
********* Results IM ELM p=1**************
Acc train: 0.9921+/-0.0035
Acc test: 0.9087+/-0.0584
Iterations: 918.3000+/-322.6931
Iterations IMA: 18.5000+/-4.5000
Updates: 10594.7000+/-2938.0436
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 286.700000000+/-12.594046212
```

```
Norm L0 (10%): 319.000000000+/-8.786353055
Norm L0 (1%): 348.000000000+/-1.732050808
Norm L0 (0.1%): 350.700000000+/-0.458257569
Norm L0 (0.01%): 350.800000000+/-0.400000000
Norm L0 (0.001%): 351.000000000+/-0.000000000
********* Results IM ELM p=inf**************
Acc train: 0.9880+/-0.0042
Acc test: 0.9058+/-0.0689
Iterations: 494.2000+/-58.0772
Iterations IMA: 20.0000+/-0.0000
Updates: 6200.0000+/-716.6035
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 175.500000000+/-23.165707414
Norm L0 (10%): 245.300000000+/-17.441616898
Norm L0 (1%): 332.500000000+/-6.020797289
Norm L0 (0.1%): 348.800000000+/-1.469693846
Norm L0 (0.01%): 350.700000000+/-0.640312424
Norm L0 (0.001%): 351.000000000+/-0.000000000
********* Results ELM **************
Acc train: 1.0000+/-0.0000
Acc test: 0.7010+/-0.0554
Margin: 0.002498547+/-0.000764023
Norm L0 (20%): 166.200000000+/-27.392699757
Norm L0 (10%): 249.700000000+/-19.360010331
Norm L0 (1%): 338.400000000+/-3.382306905
Norm L0 (0.1%): 350.000000000+/-1.000000000
Norm L0 (0.01%): 350.800000000+/-0.400000000
Norm L0 (0.001%): 351.000000000+/-0.000000000
```

2.12 Application on Banknote Dataset

[ ]:
```
# read in banknote authentication set
banknotes = pd.read_csv('data/banknote/data_banknote_authentication.txt',␣
 ↪names=['variance', 'skewness', 'curtosis', 'entropy', 'class'])

# convert to array
X = banknotes[['variance', 'skewness', 'curtosis', 'entropy']].to_numpy()
y = banknotes[['class']].to_numpy()
y[np.where(y==0)] = -1
m = results(X, y, 10, 1000, 0.1, 20)
#plot_margin_evolution(m)
```

2.13 Application on Wine Dataset

[10]:
```
wine_dataset = pd.read_csv('data/wine/wine.data', names=['Class', 'Alcohol',␣
↪'Malic acid', 'Ash', 'Alcalinity of ash', 'Magnesium', 'Total phenols',␣
↪'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins', 'Color intensity',␣
↪'Hue', 'OD280/OD315', 'Proline'])
# convert to array
y = wine_dataset[['Class']].to_numpy()
X = wine_dataset.drop("Class",axis='columns').to_numpy()
y[np.where(y==3)] = 1
y[np.where(y==2)] = -1
m = results(X, y, 10, 178, 1, 20)
#plot_margin_evolution(m)
```

```
Parameters: p=178, eta=1, lambda=3.4759183673469387
********* Results IM ELM p=2 **************
Acc train: 0.9988+/-0.0025
Acc test: 0.9775+/-0.0276
Iterations: 101.7000+/-14.3600
Iterations IMA: 20.0000+/-0.0000
Updates: 498.1000+/-111.4293
Margin: 0.040775414+/-0.043580605
Norm L0 (20%): 100.400000000+/-8.742997198
Norm L0 (10%): 136.900000000+/-4.323193264
Norm L0 (1%): 173.400000000+/-1.854723699
Norm L0 (0.1%): 177.500000000+/-0.500000000
Norm L0 (0.01%): 177.900000000+/-0.300000000
Norm L0 (0.001%): 178.000000000+/-0.000000000
********* Results IM ELM p=1**************
Acc train: 1.0000+/-0.0000
Acc test: 0.9830+/-0.0260
Iterations: 113.5000+/-14.9482
Iterations IMA: 20.0000+/-0.0000
Updates: 572.0000+/-136.0529
Margin: 0.057100118+/-0.015538195
Norm L0 (20%): 141.300000000+/-8.258934556
Norm L0 (10%): 159.200000000+/-4.445222154
Norm L0 (1%): 175.600000000+/-1.685229955
Norm L0 (0.1%): 177.600000000+/-0.489897949
Norm L0 (0.01%): 178.000000000+/-0.000000000
Norm L0 (0.001%): 178.000000000+/-0.000000000
********* Results IM ELM p=inf**************
Acc train: 0.9994+/-0.0019
Acc test: 0.9830+/-0.0260
Iterations: 90.9000+/-7.4088
Iterations IMA: 20.0000+/-0.0000
Updates: 418.6000+/-54.0226
Margin: 0.048188853+/-0.027594062
```

```
Norm L0 (20%): 74.100000000+/-14.466858678
Norm L0 (10%): 103.900000000+/-16.096272861
Norm L0 (1%): 162.200000000+/-7.124605252
Norm L0 (0.1%): 176.100000000+/-1.640121947
Norm L0 (0.01%): 177.700000000+/-0.458257569
Norm L0 (0.001%): 178.000000000+/-0.000000000
********* Results ELM **************
Acc train: 1.0000+/-0.0000
Acc test: 0.7807+/-0.0682
Margin: 0.001828082+/-0.000230393
Norm L0 (20%): 94.600000000+/-13.245376552
Norm L0 (10%): 132.300000000+/-8.706893820
Norm L0 (1%): 173.600000000+/-2.154065923
Norm L0 (0.1%): 177.100000000+/-0.830662386
Norm L0 (0.01%): 177.800000000+/-0.400000000
Norm L0 (0.001%): 177.900000000+/-0.300000000
```

2.14 Application on WDBC Dataset

[11]:
```
wdbc_dataset = pd.read_csv('data/WDBC/wdbc.data', names=list(range(0,32)))
# convert to array
y = wdbc_dataset[1].to_numpy()
X = wdbc_dataset.drop([0, 1],axis='columns').to_numpy()
y[np.where(y=='B')] = 1
y[np.where(y=='M')] = -1
y = np.array(y.tolist())
m = results(X, y, 10, 569, 0.1, 20)
```

```
Parameters: p=569, eta=0.1, lambda=4.291428571428571
********* Results IM ELM p=2 **************
Acc train: 0.9869+/-0.0029
Acc test: 0.9684+/-0.0131
Iterations: 525.1000+/-44.2774
Iterations IMA: 20.0000+/-0.0000
Updates: 6672.8000+/-961.3621
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 307.600000000+/-35.327609599
Norm L0 (10%): 435.300000000+/-22.463525992
Norm L0 (1%): 556.800000000+/-3.370459909
Norm L0 (0.1%): 567.600000000+/-1.200000000
Norm L0 (0.01%): 568.900000000+/-0.300000000
Norm L0 (0.001%): 569.000000000+/-0.000000000
********* Results IM ELM p=1**************
Acc train: 0.9881+/-0.0107
Acc test: 0.9649+/-0.0235
Iterations: 883.4000+/-357.3780
Iterations IMA: 4.8000+/-5.2498
```

```
Updates: 13723.4000+/-4140.3893
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 489.400000000+/-13.828955130
Norm L0 (10%): 529.900000000+/-7.634788799
Norm L0 (1%): 565.200000000+/-2.039607805
Norm L0 (0.1%): 568.200000000+/-0.979795897
Norm L0 (0.01%): 569.000000000+/-0.000000000
Norm L0 (0.001%): 569.000000000+/-0.000000000
********* Results IM ELM p=inf**************
Acc train: 0.9840+/-0.0056
Acc test: 0.9736+/-0.0162
Iterations: 524.7000+/-58.6055
Iterations IMA: 20.0000+/-0.0000
Updates: 6959.8000+/-1122.6149
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 150.500000000+/-42.979646346
Norm L0 (10%): 218.200000000+/-51.524363169
Norm L0 (1%): 410.300000000+/-62.758346059
Norm L0 (0.1%): 547.300000000+/-13.813399292
Norm L0 (0.01%): 566.400000000+/-2.416609195
Norm L0 (0.001%): 568.800000000+/-0.400000000
********* Results ELM **************
Acc train: 1.0000+/-0.0000
Acc test: 0.7416+/-0.0548
Margin: 0.000486659+/-0.000042358
Norm L0 (20%): 259.100000000+/-45.588266034
Norm L0 (10%): 396.300000000+/-30.741014947
Norm L0 (1%): 551.700000000+/-6.341135545
Norm L0 (0.1%): 567.100000000+/-1.972308292
Norm L0 (0.01%): 568.900000000+/-0.300000000
Norm L0 (0.001%): 568.900000000+/-0.300000000
```

```
        ␣
 ↪---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call␣
 ↪last)

        <ipython-input-11-4d50121f42a7> in <module>
          7 y = np.array(y.tolist())
          8 m = results(X, y, 10, 569, 0.1, 20)
    ----> 9 plot_margin_evolution(m)

        NameError: name 'plot_margin_evolution' is not defined
```

2.15 Application on Sonar Dataset

[12]:
```
sonar_dataset = pd.read_csv('data/sonar/sonar.all-data',␣
↪names=list(range(0,61)), sep=',')
y = sonar_dataset[60].to_numpy()
X = sonar_dataset.drop([60], axis='columns').to_numpy()
y[np.where(y=='R')] = 1
y[np.where(y=='M')] = -1
y = np.array(y.tolist())
m = results(X, y, 10, 208, 0.1, 20)
```

```
Parameters: p=208, eta=0.1, lambda=4.291428571428571
********* Results IM ELM p=2 **************
Acc train: 0.9968+/-0.0049
Acc test: 0.8064+/-0.0993
Iterations: 219.8000+/-20.9370
Iterations IMA: 20.0000+/-0.0000
Updates: 3111.9000+/-336.0594
Margin: 0.010489642+/-0.011653672
Norm L0 (20%): 118.300000000+/-12.050311199
Norm L0 (10%): 163.000000000+/-8.752142595
Norm L0 (1%): 203.300000000+/-2.609597670
Norm L0 (0.1%): 207.100000000+/-0.943398113
Norm L0 (0.01%): 208.000000000+/-0.000000000
Norm L0 (0.001%): 208.000000000+/-0.000000000
********* Results IM ELM p=1**************
Acc train: 0.9989+/-0.0021
Acc test: 0.8162+/-0.0935
Iterations: 244.7000+/-28.7613
Iterations IMA: 20.0000+/-0.0000
Updates: 3459.9000+/-561.2384
Margin: 0.018038346+/-0.013071084
Norm L0 (20%): 156.700000000+/-9.869650450
Norm L0 (10%): 182.700000000+/-6.634003316
Norm L0 (1%): 205.400000000+/-1.800000000
Norm L0 (0.1%): 207.800000000+/-0.400000000
Norm L0 (0.01%): 207.900000000+/-0.300000000
Norm L0 (0.001%): 207.900000000+/-0.300000000
********* Results IM ELM p=inf**************
Acc train: 0.9979+/-0.0035
Acc test: 0.7826+/-0.0843
Iterations: 228.6000+/-27.9757
Iterations IMA: 20.0000+/-0.0000
Updates: 3159.8000+/-477.8284
Margin: 0.012422409+/-0.011729535
Norm L0 (20%): 105.400000000+/-15.167069592
Norm L0 (10%): 144.400000000+/-11.473447607
Norm L0 (1%): 197.000000000+/-3.949683532
```

```
Norm L0 (0.1%): 207.000000000+/-0.632455532
Norm L0 (0.01%): 208.000000000+/-0.000000000
Norm L0 (0.001%): 208.000000000+/-0.000000000
********* Results ELM **************
Acc train: 1.0000+/-0.0000
Acc test: 0.7105+/-0.1292
Margin: 0.020768938+/-0.004707855
Norm L0 (20%): 113.700000000+/-14.014635208
Norm L0 (10%): 159.000000000+/-8.197560613
Norm L0 (1%): 202.300000000+/-2.758622845
Norm L0 (0.1%): 207.200000000+/-1.248999600
Norm L0 (0.01%): 207.900000000+/-0.300000000
Norm L0 (0.001%): 208.000000000+/-0.000000000
```

2.16 Application on Pima Dataset

[13]:
```python
pima_dataset = pd.read_csv('data/diabetes/diabetes.csv', sep=",",␣
 ↪engine='python')
y = pima_dataset['Outcome'].to_numpy()
X = pima_dataset.drop(['Outcome'], axis='columns').to_numpy()
y[np.where(y==0)] = -1
y = np.array(y.tolist())
m = results(X, y, 5, 768, 0.1, 20)
```

```
Parameters: p=768, eta=0.1, lambda=6.534081632653061
********* Results IM ELM p=2 **************
Acc train: 0.7835+/-0.0100
Acc test: 0.7370+/-0.0251
Iterations: 401.4000+/-23.7874
Iterations IMA: 20.0000+/-0.0000
Updates: 43226.2000+/-1518.9328
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 234.600000000+/-45.596491093
Norm L0 (10%): 440.800000000+/-43.641264876
Norm L0 (1%): 736.400000000+/-4.409081537
Norm L0 (0.1%): 764.200000000+/-1.469693846
Norm L0 (0.01%): 767.800000000+/-0.400000000
Norm L0 (0.001%): 768.000000000+/-0.000000000
********* Results IM ELM p=1**************
Acc train: 0.7217+/-0.0481
Acc test: 0.7162+/-0.0416
Iterations: 72.8000+/-19.4463
Iterations IMA: 2.2000+/-0.4000
Updates: 12790.8000+/-1039.2100
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 556.800000000+/-97.532353606
Norm L0 (10%): 659.200000000+/-52.632309469
```

```
Norm L0 (1%): 756.200000000+/-4.955804677
Norm L0 (0.1%): 767.600000000+/-0.489897949
Norm L0 (0.01%): 767.800000000+/-0.400000000
Norm L0 (0.001%): 768.000000000+/-0.000000000
********* Results IM ELM p=inf**************
Acc train: 0.7858+/-0.0077
Acc test: 0.7513+/-0.0286
Iterations: 400.8000+/-13.8910
Iterations IMA: 20.0000+/-0.0000
Updates: 42261.8000+/-1395.2444
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 138.600000000+/-35.651647928
Norm L0 (10%): 251.800000000+/-38.300913827
Norm L0 (1%): 612.000000000+/-34.380226875
Norm L0 (0.1%): 750.000000000+/-6.752777206
Norm L0 (0.01%): 767.200000000+/-0.748331477
Norm L0 (0.001%): 768.000000000+/-0.000000000
********* Results ELM **************
Acc train: 1.0000+/-0.0000
Acc test: 0.5587+/-0.0488
Margin: 0.000000833+/-0.000000076
Norm L0 (20%): 323.000000000+/-24.899799196
Norm L0 (10%): 518.600000000+/-20.185143051
Norm L0 (1%): 741.400000000+/-2.497999199
Norm L0 (0.1%): 766.200000000+/-0.979795897
Norm L0 (0.01%): 768.000000000+/-0.000000000
Norm L0 (0.001%): 768.000000000+/-0.000000000
```

2.17 Application on Statlog Dataset

[14]:
```
statlog_dataset = pd.read_csv('data/statlog/heart.dat', sep=" ", header=None,␣
 ↪engine='python')
y = statlog_dataset[13].to_numpy()
X = statlog_dataset.drop([13], axis='columns').to_numpy()
y[np.where(y==2)] = -1
y = np.array(y.tolist())
m = results(X, y, 10, 270, 0.1, 20)
```

```
Parameters: p=270, eta=0.1, lambda=7.553469387755102
********* Results IM ELM p=2 **************
Acc train: 0.8757+/-0.0249
Acc test: 0.7963+/-0.0727
Iterations: 299.1000+/-39.2822
Iterations IMA: 18.2000+/-5.4000
Updates: 7822.1000+/-1025.3015
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 130.400000000+/-16.378034070
```

```
Norm L0 (10%): 196.100000000+/-8.734414691
Norm L0 (1%): 260.800000000+/-2.785677655
Norm L0 (0.1%): 269.400000000+/-0.663324958
Norm L0 (0.01%): 270.000000000+/-0.000000000
Norm L0 (0.001%): 270.000000000+/-0.000000000
********* Results IM ELM p=1**************
Acc train: 0.8510+/-0.0453
Acc test: 0.7778+/-0.0597
Iterations: 223.4000+/-27.1522
Iterations IMA: 2.0000+/-0.0000
Updates: 10461.8000+/-47.3324
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 226.400000000+/-8.309031231
Norm L0 (10%): 249.800000000+/-5.230678732
Norm L0 (1%): 268.700000000+/-1.100000000
Norm L0 (0.1%): 269.900000000+/-0.300000000
Norm L0 (0.01%): 270.000000000+/-0.000000000
Norm L0 (0.001%): 270.000000000+/-0.000000000
********* Results IM ELM p=inf**************
Acc train: 0.8786+/-0.0171
Acc test: 0.7852+/-0.0825
Iterations: 310.5000+/-14.1510
Iterations IMA: 20.0000+/-0.0000
Updates: 7539.1000+/-390.8315
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 76.700000000+/-12.961867149
Norm L0 (10%): 122.200000000+/-13.854963010
Norm L0 (1%): 235.900000000+/-12.809761903
Norm L0 (0.1%): 265.900000000+/-2.426932220
Norm L0 (0.01%): 269.200000000+/-0.748331477
Norm L0 (0.001%): 269.900000000+/-0.300000000
********* Results ELM **************
Acc train: 1.0000+/-0.0000
Acc test: 0.5889+/-0.0785
Margin: 0.001355842+/-0.000368161
Norm L0 (20%): 128.100000000+/-11.246777316
Norm L0 (10%): 194.500000000+/-8.127115109
Norm L0 (1%): 262.100000000+/-2.467792536
Norm L0 (0.1%): 269.500000000+/-0.670820393
Norm L0 (0.01%): 269.800000000+/-0.600000000
Norm L0 (0.001%): 270.000000000+/-0.000000000
```

2.18 Application on Mammographic Dataset

[15]:
```python
mammo = pd.read_csv('data/mammographic/mammographic_masses.data', sep=",",␣
 ↪header=None, engine='python')
mammo = mammo.replace("?", np.nan)
mammo = mammo.dropna()
y = mammo[5].to_numpy()
X = mammo.drop([5], axis='columns').to_numpy()
y[np.where(y==0)] = -1
y = np.array(y.tolist())
m = results(X, y, 10, 830, 0.1, 20)
```

```
Parameters: p=830, eta=0.1, lambda=6.1263265306122445
********* Results IM ELM p=2 **************
Acc train: 0.8320+/-0.0045
Acc test: 0.8325+/-0.0518
Iterations: 521.7000+/-20.6787
Iterations IMA: 20.0000+/-0.0000
Updates: 52971.2000+/-1485.7010
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 496.100000000+/-36.231064020
Norm L0 (10%): 661.900000000+/-20.334453521
Norm L0 (1%): 812.600000000+/-5.407402334
Norm L0 (0.1%): 828.400000000+/-1.113552873
Norm L0 (0.01%): 829.900000000+/-0.300000000
Norm L0 (0.001%): 830.000000000+/-0.000000000
********* Results IM ELM p=1**************
Acc train: 0.8118+/-0.0177
Acc test: 0.8048+/-0.0567
Iterations: 81.4000+/-19.2676
Iterations IMA: 2.1000+/-0.3000
Updates: 13209.6000+/-1256.6015
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 660.200000000+/-27.942082957
Norm L0 (10%): 744.200000000+/-17.348198754
Norm L0 (1%): 821.000000000+/-3.033150178
Norm L0 (0.1%): 828.900000000+/-1.044030651
Norm L0 (0.01%): 829.900000000+/-0.300000000
Norm L0 (0.001%): 830.000000000+/-0.000000000
********* Results IM ELM p=inf**************
Acc train: 0.8293+/-0.0056
Acc test: 0.8229+/-0.0457
Iterations: 524.1000+/-21.8607
Iterations IMA: 20.0000+/-0.0000
Updates: 52423.8000+/-2089.2040
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 375.100000000+/-49.613405447
Norm L0 (10%): 510.300000000+/-48.503711198
```

```
Norm L0 (1%): 719.100000000+/-30.615192307
Norm L0 (0.1%): 819.500000000+/-3.853569774
Norm L0 (0.01%): 828.400000000+/-0.916515139
Norm L0 (0.001%): 830.000000000+/-0.000000000
********* Results ELM **************
Acc train: 0.9070+/-0.0048
Acc test: 0.7325+/-0.0493
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 267.400000000+/-38.252320191
Norm L0 (10%): 492.900000000+/-34.832312585
Norm L0 (1%): 792.200000000+/-6.144916598
Norm L0 (0.1%): 827.000000000+/-2.000000000
Norm L0 (0.01%): 829.900000000+/-0.300000000
Norm L0 (0.001%): 829.900000000+/-0.300000000
```

2.19 Application on Haberman Dataset

[16]:
```
haberman = pd.read_csv('data/haberman/haberman.data', sep=",", header=None,␣
 ↪engine='python')
y = haberman[3].to_numpy()
X = haberman.drop([3], axis='columns').to_numpy()
y[np.where(y==2)] = -1
y = np.array(y.tolist())
m = results(X, y, 10, 306, 0.1, 20)
```

```
Parameters: p=306, eta=0.1, lambda=4.6991836734693875
********* Results IM ELM p=2 **************
Acc train: 0.7360+/-0.0061
Acc test: 0.7451+/-0.0283
Iterations: 125.3000+/-35.8638
Iterations IMA: 2.4000+/-0.9165
Updates: 10894.7000+/-604.5360
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 152.900000000+/-29.857829794
Norm L0 (10%): 225.600000000+/-23.555041923
Norm L0 (1%): 297.600000000+/-3.168595904
Norm L0 (0.1%): 304.700000000+/-1.004987562
Norm L0 (0.01%): 305.900000000+/-0.300000000
Norm L0 (0.001%): 306.000000000+/-0.000000000
********* Results IM ELM p=1**************
Acc train: 0.7364+/-0.0155
Acc test: 0.7219+/-0.0491
Iterations: 111.5000+/-17.7158
Iterations IMA: 2.0000+/-0.0000
Updates: 10621.6000+/-101.6614
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 230.300000000+/-18.809837852
```

```
Norm L0 (10%): 265.900000000+/-13.714590770
Norm L0 (1%): 302.400000000+/-2.059126028
Norm L0 (0.1%): 305.900000000+/-0.300000000
Norm L0 (0.01%): 305.900000000+/-0.300000000
Norm L0 (0.001%): 306.000000000+/-0.000000000
********* Results IM ELM p=inf**************
Acc train: 0.7353+/-0.0011
Acc test: 0.7353+/-0.0094
Iterations: 307.8000+/-122.8265
Iterations IMA: 9.3000+/-3.2265
Updates: 16307.7000+/-3077.4727
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 154.400000000+/-25.892856158
Norm L0 (10%): 224.000000000+/-17.905306476
Norm L0 (1%): 297.700000000+/-3.436568055
Norm L0 (0.1%): 305.700000000+/-0.458257569
Norm L0 (0.01%): 306.000000000+/-0.000000000
Norm L0 (0.001%): 306.000000000+/-0.000000000
********* Results ELM **************
Acc train: 0.9684+/-0.0033
Acc test: 0.6632+/-0.0756
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 128.500000000+/-18.260613352
Norm L0 (10%): 195.200000000+/-15.387007506
Norm L0 (1%): 293.300000000+/-5.216320542
Norm L0 (0.1%): 304.900000000+/-1.135781669
Norm L0 (0.01%): 306.000000000+/-0.000000000
Norm L0 (0.001%): 306.000000000+/-0.000000000
```

2.20 Application on Transfusion Dataset

[17]:
```
transfusion = pd.read_csv('data/transfusion/transfusion.data', sep=",",␣
↪engine='python')
y = transfusion["whether he/she donated blood in March 2007"].to_numpy()
X = transfusion.drop(["whether he/she donated blood in March 2007"],␣
↪axis='columns').to_numpy()
y[np.where(y==0)] = -1
y = np.array(y.tolist())
m = results(X, y, 10, 748, 0.1, 20)
```

```
Parameters: p=748, eta=0.1, lambda=2.2526530612244895
********* Results IM ELM p=2 **************
Acc train: 0.7732+/-0.0061
Acc test: 0.7661+/-0.0131
Iterations: 83.0000+/-27.5608
Iterations IMA: 2.5000+/-0.9220
Updates: 14435.6000+/-2938.7719
```

```
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 227.700000000+/-56.076822307
Norm L0 (10%): 426.800000000+/-50.123447607
Norm L0 (1%): 714.300000000+/-7.483982897
Norm L0 (0.1%): 744.000000000+/-1.788854382
Norm L0 (0.01%): 747.800000000+/-0.400000000
Norm L0 (0.001%): 748.000000000+/-0.000000000
********* Results IM ELM p=1**************
Acc train: 0.7741+/-0.0099
Acc test: 0.7661+/-0.0260
Iterations: 65.8000+/-13.5263
Iterations IMA: 2.0000+/-0.0000
Updates: 12817.4000+/-667.0732
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 572.300000000+/-53.449134698
Norm L0 (10%): 659.100000000+/-29.170018855
Norm L0 (1%): 739.800000000+/-4.770744177
Norm L0 (0.1%): 747.500000000+/-0.670820393
Norm L0 (0.01%): 748.000000000+/-0.000000000
Norm L0 (0.001%): 748.000000000+/-0.000000000
********* Results IM ELM p=inf**************
Acc train: 0.7652+/-0.0027
Acc test: 0.7621+/-0.0090
Iterations: 397.8000+/-185.2943
Iterations IMA: 8.3000+/-3.3181
Updates: 43094.5000+/-16873.4779
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 178.400000000+/-187.353249238
Norm L0 (10%): 273.600000000+/-256.822584677
Norm L0 (1%): 602.100000000+/-144.711056938
Norm L0 (0.1%): 731.200000000+/-16.575886100
Norm L0 (0.01%): 746.400000000+/-1.685229955
Norm L0 (0.001%): 747.800000000+/-0.400000000
********* Results ELM **************
Acc train: 0.8690+/-0.0036
Acc test: 0.6950+/-0.0559
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 266.900000000+/-23.218311739
Norm L0 (10%): 457.500000000+/-14.527560015
Norm L0 (1%): 715.900000000+/-4.158124577
Norm L0 (0.1%): 744.300000000+/-1.900000000
Norm L0 (0.01%): 747.300000000+/-0.900000000
Norm L0 (0.001%): 747.900000000+/-0.300000000
```

2.21 Application on Australian Credit

[18]:
```
australian = pd.read_csv('data/australian_credit/australian.dat', header=None,␣
 ↪sep=" ", engine='python')
australian = australian.replace("?", np.nan)
australian = australian.dropna()
y = australian[14].to_numpy()
X = australian.drop([14], axis='columns').to_numpy()
y[np.where(y==0)] = -1
y = np.array(y.tolist())
m = results(X, y, 10, 1000, 0.1, 20)
```

```
Parameters: p=1000, eta=0.1, lambda=8.572857142857142
********* Results IM ELM p=2 **************
Acc train: 0.9047+/-0.0038
Acc test: 0.8609+/-0.0269
Iterations: 657.3000+/-25.2430
Iterations IMA: 20.0000+/-0.0000
Updates: 38250.3000+/-1765.9727
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 433.300000000+/-50.517422737
Norm L0 (10%): 692.300000000+/-30.070084802
Norm L0 (1%): 968.600000000+/-7.952358141
Norm L0 (0.1%): 997.200000000+/-1.536229150
Norm L0 (0.01%): 999.700000000+/-0.458257569
Norm L0 (0.001%): 1000.000000000+/-0.000000000
********* Results IM ELM p=1**************
Acc train: 0.8786+/-0.0114
Acc test: 0.8652+/-0.0337
Iterations: 151.0000+/-41.1971
Iterations IMA: 2.1000+/-0.3000
Updates: 13089.3000+/-2274.6447
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 675.000000000+/-61.736536994
Norm L0 (10%): 831.700000000+/-30.620418025
Norm L0 (1%): 984.200000000+/-4.069397990
Norm L0 (0.1%): 998.500000000+/-1.431782106
Norm L0 (0.01%): 999.900000000+/-0.300000000
Norm L0 (0.001%): 1000.000000000+/-0.000000000
********* Results IM ELM p=inf**************
Acc train: 0.9045+/-0.0072
Acc test: 0.8536+/-0.0246
Iterations: 650.1000+/-25.3947
Iterations IMA: 20.0000+/-0.0000
Updates: 37828.8000+/-1892.4502
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 352.400000000+/-38.851512197
Norm L0 (10%): 580.900000000+/-33.431871022
```

```
Norm L0 (1%): 918.200000000+/-18.529975715
Norm L0 (0.1%): 991.600000000+/-3.006659276
Norm L0 (0.01%): 999.500000000+/-0.670820393
Norm L0 (0.001%): 999.800000000+/-0.400000000
********* Results ELM **************
Acc train: 1.0000+/-0.0000
Acc test: 0.4768+/-0.0717
Margin: 0.000025978+/-0.000014173
Norm L0 (20%): 461.800000000+/-56.343233844
Norm L0 (10%): 714.600000000+/-41.250939383
Norm L0 (1%): 970.400000000+/-7.901898506
Norm L0 (0.1%): 997.400000000+/-2.009975124
Norm L0 (0.01%): 999.600000000+/-0.663324958
Norm L0 (0.001%): 999.900000000+/-0.300000000
```

2.22 Application on Breast Cancer Dataset

[19]:
```
breast = pd.read_csv('data/breast/breast.data', header=None, sep=",",␣
 ↪engine='python')
breast = breast.replace("?", np.nan)
breast = breast.dropna()
y = breast[10].to_numpy()
X = breast.drop([0, 10], axis='columns').to_numpy()
y[np.where(y==4)] = 1
y[np.where(y==2)] = -1
y = np.array(y.tolist())
m = results(X, y, 10, 699, 0.1, 20)
plot_margin_evolution(m)
```

```
Parameters: p=699, eta=0.1, lambda=1.437142857142857
********* Results IM ELM p=2 **************
Acc train: 0.9745+/-0.0046
Acc test: 0.9766+/-0.0188
Iterations: 897.4000+/-237.9324
Iterations IMA: 14.1000+/-7.4357
Updates: 18288.1000+/-3419.0049
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 231.800000000+/-33.331066590
Norm L0 (10%): 422.100000000+/-28.644196620
Norm L0 (1%): 669.000000000+/-7.000000000
Norm L0 (0.1%): 695.900000000+/-1.700000000
Norm L0 (0.01%): 698.800000000+/-0.400000000
Norm L0 (0.001%): 699.000000000+/-0.000000000
********* Results IM ELM p=1**************
Acc train: 0.9751+/-0.0036
Acc test: 0.9722+/-0.0250
Iterations: 414.4000+/-61.4804
```

```
Iterations IMA: 2.1000+/-0.3000
Updates: 11101.3000+/-563.7909
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 551.400000000+/-34.337151891
Norm L0 (10%): 622.100000000+/-17.952437160
Norm L0 (1%): 691.100000000+/-3.645545227
Norm L0 (0.1%): 698.400000000+/-0.663324958
Norm L0 (0.01%): 698.900000000+/-0.300000000
Norm L0 (0.001%): 699.000000000+/-0.000000000
********* Results IM ELM p=inf**************
Acc train: 0.9751+/-0.0046
Acc test: 0.9663+/-0.0187
Iterations: 1000.6000+/-35.5252
Iterations IMA: 20.0000+/-0.0000
Updates: 19541.8000+/-1927.8885
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 62.700000000+/-24.095850265
Norm L0 (10%): 102.900000000+/-40.692628325
Norm L0 (1%): 323.600000000+/-107.510185564
Norm L0 (0.1%): 646.400000000+/-24.054937123
Norm L0 (0.01%): 693.400000000+/-3.292415527
Norm L0 (0.001%): 698.400000000+/-0.800000000
********* Results ELM **************
Acc train: 1.0000+/-0.0000
Acc test: 0.8403+/-0.0497
Margin: 0.000462650+/-0.000032669
Norm L0 (20%): 338.000000000+/-43.566041822
Norm L0 (10%): 513.200000000+/-30.616335509
Norm L0 (1%): 679.200000000+/-4.190465368
Norm L0 (0.1%): 697.500000000+/-0.806225775
Norm L0 (0.01%): 699.000000000+/-0.000000000
Norm L0 (0.001%): 699.000000000+/-0.000000000
```

```
      ␣
↪---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call␣
↪last)

        <ipython-input-19-0dc868f2c333> in <module>
          8 y = np.array(y.tolist())
          9 m = results(X, y, 10, 699, 0.1, 20)
    ---> 10 plot_margin_evolution(m)


        NameError: name 'plot_margin_evolution' is not defined
```

2.23 Application on Spam Dataset

[21]:
```
spam = pd.read_csv('data/spam/spambase.data', header=None, sep=",",␣
 ↪engine='python')
y = spam[57].to_numpy()
X = spam.drop([57], axis='columns').to_numpy()
y[np.where(y==0)] = -1
y = np.array(y.tolist())
m = results(X, y, 10, 1000, 0.1, 20)
```

```
Parameters: p=1000, eta=0.1, lambda=3.679795918367347
********* Results IM ELM p=2 **************
Acc train: 0.7029+/-0.1481
Acc test: 0.7033+/-0.1487
Iterations: 247.5000+/-284.7178
Iterations IMA: 6.7000+/-8.7069
Updates: 53323.8000+/-66209.7456
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 131.200000000+/-201.557832892
Norm L0 (10%): 206.600000000+/-315.751547898
Norm L0 (1%): 290.800000000+/-444.205763132
Norm L0 (0.1%): 299.100000000+/-456.883015662
Norm L0 (0.01%): 300.000000000+/-458.257569496
Norm L0 (0.001%): 300.000000000+/-458.257569496
********* Results IM ELM p=1**************
Acc train: 0.6897+/-0.1287
Acc test: 0.6879+/-0.1260
Iterations: 69.8000+/-13.8477
Iterations IMA: 1.3000+/-0.4583
Updates: 13041.1000+/-4557.3034
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 182.200000000+/-279.763757481
Norm L0 (10%): 240.500000000+/-367.762219376
Norm L0 (1%): 293.700000000+/-448.635052130
Norm L0 (0.1%): 299.300000000+/-457.188374743
Norm L0 (0.01%): 300.000000000+/-458.257569496
Norm L0 (0.001%): 300.000000000+/-458.257569496
********* Results IM ELM p=inf**************
Acc train: 0.7019+/-0.1466
Acc test: 0.7010+/-0.1451
Iterations: 253.4000+/-293.5358
Iterations IMA: 6.7000+/-8.7069
Updates: 53464.0000+/-66333.4867
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 125.100000000+/-192.489194502
Norm L0 (10%): 198.500000000+/-303.561937667
Norm L0 (1%): 287.200000000+/-438.708513708
Norm L0 (0.1%): 299.000000000+/-456.731211984
```

```
Norm L0 (0.01%): 300.000000000+/-458.257569496
Norm L0 (0.001%): 300.000000000+/-458.257569496
********* Results ELM **************
Acc train: 0.9598+/-0.0008
Acc test: 0.9113+/-0.0125
Margin: 0.000000000+/-0.000000000
Norm L0 (20%): 347.800000000+/-57.525298782
Norm L0 (10%): 623.300000000+/-52.653679833
Norm L0 (1%): 960.100000000+/-9.224424101
Norm L0 (0.1%): 995.900000000+/-2.662705391
Norm L0 (0.01%): 999.300000000+/-1.004987562
Norm L0 (0.001%): 1000.000000000+/-0.000000000
```

[ ]:

```

```