# Topic detection with MLP and ADALINE

## 1 استخراج ویژگی

قبل از هر چیز نیاز به خواندن داده داریم به صورت زیر

```
#dataset directory
folder='data'
#read the data using read_data function
Data=Read_data(folder)
```

تابع Read_data این کار را انجام میدهد . این تابع را میتوان به صورت زیر خلاصه کرد

پیدا کردن تمام پوشه ها در دایرکتوری ارسال شده

```
for filename in os.listdir(folder):
    #add a class to each file
    labels_name += 1
    path1 = os.path.join(folder, filename)
```

پیدا کردن تمام پوشه در هر کدام از پوشه های شناخته شده در خط بالا

```
    for i in os.listdir(path1):
```

محاسبه ادرس هر فایل متنی در هر پوشه

```
        path2text = os.path.join(path1, i)
        #read the text file and add it to Data
```

خواندن ان فایل به صورت یک رشته و اضافه کردن ان همراه کلاس ان به متغییر Data

```
        with open(path2text, "r",encoding="utf8") as
        myfile:
                    sample = myfile.read()
                    Data.append([ sample,labels_name])
```

پس از خواندن داده ها نوبت به استخراج ویژگی است برای استخراج ویژگی ابتدا هر متن را پیش پردازش کرده به صورتی که تمام اعداد و کارکتر های غیر فارسی از ان حذف می کنیم و ان را به کلمات سازنده ان می شکنیم. سپس یک دیکشنری از تمام لغات شاخته شده درست میکنیم و اسناد را با استفاده از این دیکشنری برداری میکنیم . این کار توسط تابع vectorizer انجام می شود که میتوان ان را به صورت زیر توضیح داد

برای هر متن در داخل corpus ابتدا ان را با استفاده از تابع tok پیش پردازش کرده و سپس برای تک تک لغات ان بررسی میکنیم که ایا این لغت در دیکشنری موجود است یا نه. اگه موجود نبود ان را اضافه و اگر بود به تعداد ان یکی اضافه میکنیم/

```
for i in Corpus:
    text = TOK(i)
    for j in text:
        #check if the word is in the vocabulary
        if j in dict:
            dict[j]+= 1
        else:
            dict[j] = 1
```

پس از ان تعداد تکرار هر کلمه را شمرده و سپس 1000 تای پر تکرار ان را برمیداریم

```
#count the number of accurance of each word and take
the 1000 most frequent words
Vocabulary = Counter(dict).most_common(1000)
#seperqate the words
Vocabulary = [x[0] for x in Vocabulary]
```

پس از ان برای هر سند یک بردار 100 تایی میسازیم و با استفاده از کلمات دیکشنری مطابق با توضیح سوال ان را برداری میکنیم

```
#define a feature vectors for each document
vector = np.zeros(1000, dtype=float)

#doing the same thing for test data
for i in Data:
```

پیش پردازش متن

```
    text = TOK(i[0])
    for j in text:
        if j in Vocabulary:
```

شمردن تعداد تکرار های هر لغت و ذخیره ان در جایگاه مناسب

```
            vector[Vocabulary.index(j)] += 1
```

گرفتن لگاریتم تعداد تکرار هر لغت مطابق صورت سوال

```
    vector = np.asanyarray([math.log(x + 1) for x in
vector])
```

دخیره هر بردار و کلاس ان در data_vectors

```
    data_vectors.append([vector, i[1]])
    vector = np.zeros(1000, dtype=float)
```

پس از پیش پردازش داده ها نوبت به جدا کردن داده ای تست و
آموزش است به صورتی که برای هر موضوع 2 سند برای تست و 8
تا برای اموزش داشته باشیم . این کار به صورت زیر انجام
میشود

```
# split the data so we will have 2 documents as test
data for each category
X_train,X_test,y_train,y_test=Split_data(X,Y)
```

که در ان X,Y ویژگی های استخراج شده در مرحله قبل است.
این تابع را میتوان به صورت زیر خلاصه کرد
برای هر 7 کلاس ابتدا داده های ان هارا پیدا کرده و سپس 2
تای اول ان را به عنوان تست و بقیه را برای اموزش جدا
میکنیم

```
#for each class we select the 2 sample to the test and
the rest to the train data
for i in range(7):
    #find class with lable equal to (i+1)
```

پیدا کردن اندیش های کلاس i+1
توجه :i از صفر شروع میشود برای همین ان را به علاوه 1
میکنیم

```
indecis = np.where(Y == i + 1)
```

جدا کردن داده های تست و اموزش از هم و ذخیره ان ها

```
    test_index = indecis[0][0:2]
    train_index = indecis[0][2:]
    #append the test data
    for te in test_index:
        X_test.append(X[te, :])
        y_test.append(Y[te])
    # append the train data
    for tr in train_index:
        X_train.append(X[tr, :])
        y_train.append(Y[tr])
```

تا این مرحله برنامه در فایل main این توابع را اجرا کرده است

```
#dataset directory
folder='data'
#read the data using read_data function
Data=Read_data(folder)

#extracting features using vectorizer function
X,Y=vectorizer(Data)

# split the data so we will have 2 documents as test
data for each category
X_train,X_test,y_train,y_test=Split_data(X,Y)
```

## 2 پیاده سازی ADALINE

ابتدا یک کلاس ساخته که بتوان خیلی سریع به اندازه دلخواه از این کلاسیفایر ساخت

```
#define a class
class Adaline():
```

تعیین نرخ یادگیری در موقع فراخوانی کلاس

```
    #set the learniing rate as the classifer is created
    def __init__(self, eta):
        self.eta = eta
```

یادگیری

```
    #traing ADALINE
    def fit(self, X, y):
```

مقدار دهی وزن های اولیه با مقادیر کوچک و تصادفی

```
        np.random.seed(16)
        #assgning small randoms numbers to weights
        self.weight_ = np.random.uniform(-1, 1,
X.shape[1] + 1)
        self.error_ = []
```

```
        #stop criteria
        update=True
        while(update):
```

برای هر جفت داده و کلاس ان وزن ها را اپدیت میکنیم

```
  #for each sample we update the weight and
check if we have to stop
 for xi, target in zip(X, y):
                p_weight = self.weight_
                output = self.feed_forward(xi)
```

محاسبه خطلا برای هر نمونه

```
 error = (target - output)
 #updating th weights
 self.weight_[1:] += self.eta * xi.dot(error)
 self.weight_[0] += self.eta * error
 #calculate the diffference between weights

 weight_diff=sum((self.weight_-p_weight)**2)
#check if we have to stop training
```

بررسی شرایط توقف یعنی اگر وزن ها بی تغییر شدن یا دقت به حد کافی رسید

```
 if accuracy_score(y,self.predict_int(X))>99 or
weight_diff<.0000000000001 :
                    update=False
                    break

        return self
```

گرفتن خروجی در هر مرحله برای اپدیت وزدن ها

```
    #pass the data through the layer to get a ouptput
    def feed_forward(self, X):
        return np.dot(X, self.weight_[1:]) +
self.weight_[0]
```

گرفتن خروجی به صورت اینتجر یعنی یا 1 یا منفی 1

```
    #predict each class
    def predict_int(self, X):
        return np.where(self.feed_forward(X) >= 0.0, 1,
```

```
-1)
```

گرفتن احتمال برای هر نمونه . این کار با محاسبه فاصله هر نقطه از HYPERPLANE ایجاد شده بدست می اید.

```
#predict the probability of each class
    def predict(self, X):
        output=self.feed_forward(X)
        output=output/np.linalg.norm(output,ord=2)
        return output
```

برای استفاده از adaline به صورت زیر عمل میکنم
ابتدا از دیتا ست 7 تا دیتا ست binary میسازیم هر کدام برای یک کلاس. مثلا برای کلاس یک ابتدا labelsهای این کلاس را 1 و بقیه labels ها را 1- میکنیم. پس از ساختن این دیتا ست 7 تا شبکه adaline را جداگانه اموزش میدهیم

```
for i in range(7):
    #create a Adaline classifer with larning rate =
0.0000001
```

تعریف adaline

```
    ada = Adaline(eta=0.0000001)
    #make a binay dataset
```

ایجاد یک دیتا ست دوتایی

```
    y[y!=i+1]=-1
    y[y==i+1]=1
    #train the classifer
```

اموزش adaline و محاسبه دقت اموزش

```
    ada.fit(X_train,y)
    #predict the traning data
    y_pred=ada.predict_int(X_train)
    print('accuracy of classifer',str(i+1),' : '
,accuracy_score(y,y_pred))
    #assign th ogirinal dataset to y for the next loop
    y = copy.copy(y_train)
    #add the classifier to the list
```

اضافه کردن کلاسیفایر ایجاد شده به یک لیست برای محاسبه های بعدی

```
    classifiers.append(ada)
```

حال داده های تست و اموزش را به تک تک کلاسیفایر های ایجاد شده در مرحله قبل داده و نتیجه همه ی انها را ذخیره میکنم یعنی خروجی ما برای داده ها اموزش یک ماتریس 56*7 است 7 برای هر کلاسیفایر و 56 که تعداد داده های اموزشی است. سپس برای تصمیم گیری نهایی برای هر کدام از 56 تا داده ان کلایسفایری که با احتمال بیشتری خروجی را پیشبینی کرده بود انتخاب میشود

پیش بینی تمام داده ها

```
#predict the test and the traing data using all th
classifers
predicted_labels_train=[]
predicted_labels_test=[]
for clf in classifiers:
    predicted_labels_train.append(clf.predict(X_train))
    predicted_labels_test.append(clf.predict(X_test))
```

استفاده از تکنیک توضیح داده شده در بالا و ذخیره کلاس نهایی در متغییر final_class_testو final_class_train

```
using one versus all technique to classify each sample
in train data
final_classes_test=[]
final_classes_train=[]
for i in range(len(predicted_labels_train[0])):
    col=[]
    for j in range(7):
        col.append(predicted_labels_train[j][i])
        #assign the max probability of each class to
the final labels
    final_classes_train.append(col.index(max(col))+1)
```

و در اخر نمایش دقت برای تست و اموزش

```
#display the final result using ADALINE
print('final accuracy on train data: ',
accuracy_score(final_classes_train,y_train))
print('final accuracy on test data: ',
accuracy_score(final_classes_test,y_test))
```

## 3 شبکه عصبی پرسپترون

از کتابخانه sklearn استفاده شده است

تعریف mlp

```
print("\n\nRunning neural network...")
nn_clf = MLPClassifier()
#define MLP and set learning rate to 0.01
```

تنظیم پارامتر ها ( نرخ یادگیری برابر .01 طبق گفته سوال)

```
MLPClassifier(solver='adam', activation='tanh',
early_stopping=True,learning_rate=.01, alpha=1e-5,
            hidden_layer_sizes=(200, 100))
```

اموزش شبکه

```
#traing MLP
nn_clf.fit(X_train, y_train)
#predict the test data
```

پیش بینی داده های تست و ترین و نمایش دقت ان ها

```
y_pred = nn_clf.predict(X_test)
print('accuracy on test data :',accuracy_score(y_test,
y_pred))
# predict the train data
y_pred = nn_clf.predict(X_train)
print("accuracy on train data :
",accuracy_score(y_train, y_pred))
```

برای نمایش خطای مدل در حین یادگیری از کد های اماده پایتون در سایت sklearn استفاده شده است که در فایل plot در این پروژه موجود است . تنها تغییر انجام شده تغییر دقت کلاسیفایر به خطا است چون در صورت سوال خطا ی یادگیری خواسته شده است

نحوه استفاده از این تابع :

X,Y داده های ما هستند. و cv مقدار crossvalidation است که برای تست مدل به کار میرود. هر چند که در صورت سوال مطرح نشده است اما ما در اینجا نمودار تغییرات را برای داده های تست زیر نشان میدهیم برای درک بهتر. چون داده های تست ما 14 تا هستند پس cv را 5 انتخاب کرده که 70/5 برابر 14 شود.

```
title = "Learning Curves (neural network)"
```

```
#plot the learning curve
plot_learning_curve(nn_clf, title, X, Y, ylim=(-0.1,
1.01), cv=5)
plt.show()
```

nn_clf همان کلاسیفایر mlpاست.

کار های بالا برای کلاسیفایر naïve bayes نیز انجام شده است

```
# _____ using naive
bayes_____#
print("\n\nRunning Naive bayes...")
NB_model = GaussianNB()
NB_model = NB_model.fit(X=X_train, y=y_train)
NB_model.fit(X_train, y_train)
y_pred = NB_model.predict(X_test)
print('accuracy on test data :', accuracy_score(y_test,
y_pred))

y_pred = NB_model.predict(X_train)
print("accuracy on train data : ",
accuracy_score(y_train, y_pred))

title = "Learning Curves (Naive bayes)"

plot_learning_curve(NB_model, title, X, Y, ylim=(-0.1,
1.01), cv=5)
plt.show()
```

## 4 نتایج

- نتایج برای شبکه adaline و استفاده از تکنیک OVA

```
running ADALINE classifer using OVA technique...

accuracy of classifer 1  :  0.857142857143
accuracy of classifer 2  :  0.821428571429
accuracy of classifer 3  :  0.785714285714
accuracy of classifer 4  :  0.785714285714
accuracy of classifer 5  :  0.821428571429
accuracy of classifer 6  :  0.785714285714
accuracy of classifer 7  :  0.785714285714
final accuracy on train data:  0.178571428571
final accuracy on test data:  0.142857142857
```

دقت هر کلاسیفایر را برای دیتا ست مربطه ملاحضه میکنید و همچنین دقت کلی برای داده ای اموزش و تست.

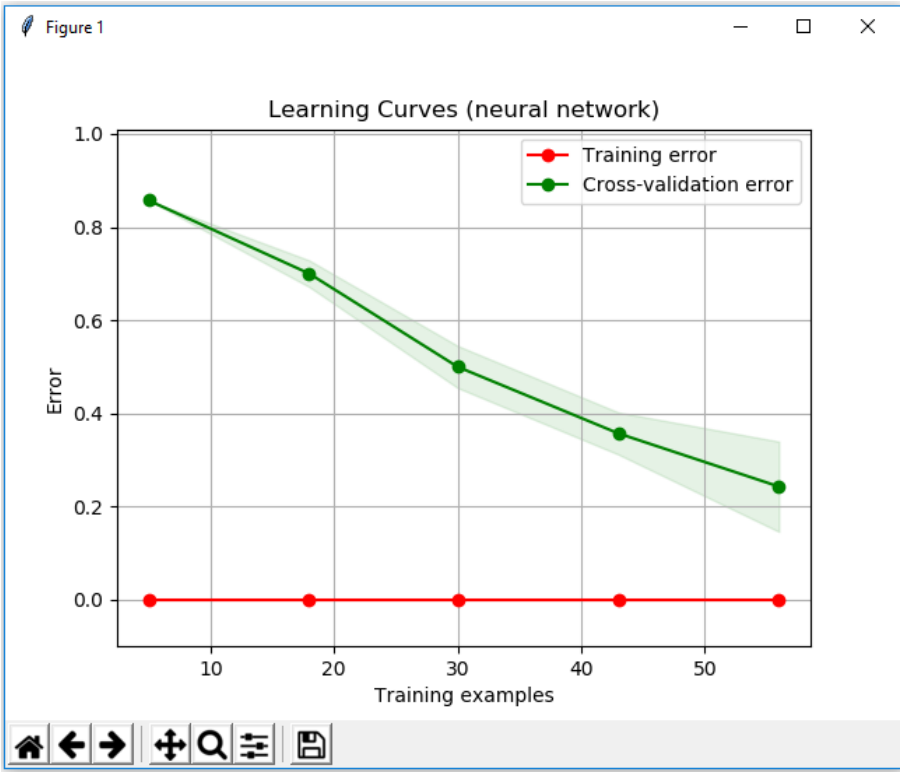
پس از امتحان نرخ های یادگیری مختلف نرخ یادگیری 0.0000001 انتخاب شده است که بهترین دقت را نسبت به بقیه دارد.

- نتایج برای mlp با ساختار لایه های 200و 100 نورون هر کدام متعلق به لایه اول و دوم

```
MLPClassifier(solver='adam', activation='tanh',
early_stopping=True,learning_rate=.01, alpha=1e-5,
            hidden_layer_sizes=(300, 200))
```

```
Running neural network...
accuracy on test data : 0.714285714286
accuracy on train data :  1.0
```

همانطور که مشاهده میکنید دقت خیلی نسبت به روش قبل بهتر است.
نمودار تغییرات خطای این شبکه:



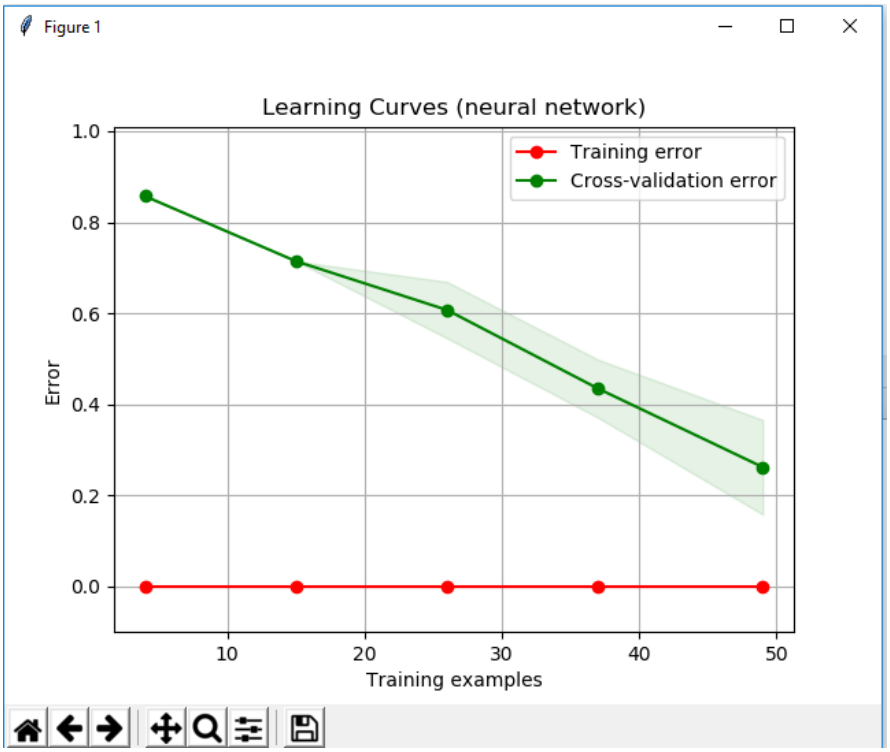

همانطور که مشاهده میکنین خطای یادگیری با زیاد شده داده های اموزش ثابت و برابر 0 بوده اما با زیاد شدن داده های اموطش خطا برای روی داده های تست کم شده است که این نشان میدهد افزایش داده های اموزش بیشتر موجد یادگیری بهتر میشود.

- نتایج برای mlp با ساختار 50 نورون برای لایه میانی

```
MLPClassifier(solver='adam', activation='tanh',
early_stopping=True,learning_rate=.01, alpha=1e-5,
            hidden_layer_sizes=(50))
```

```
Running neural network...
accuracy on test data : 0.571428571429
accuracy on train data :  1.0
```

همان طور که مشاهده میکنید دقت نست به حالت قبل کم تر شده است چون تنها یک لایه داریم و برای این مسئله کافی نیست.
نمودار ان را در زیر مشاهده میکنید.



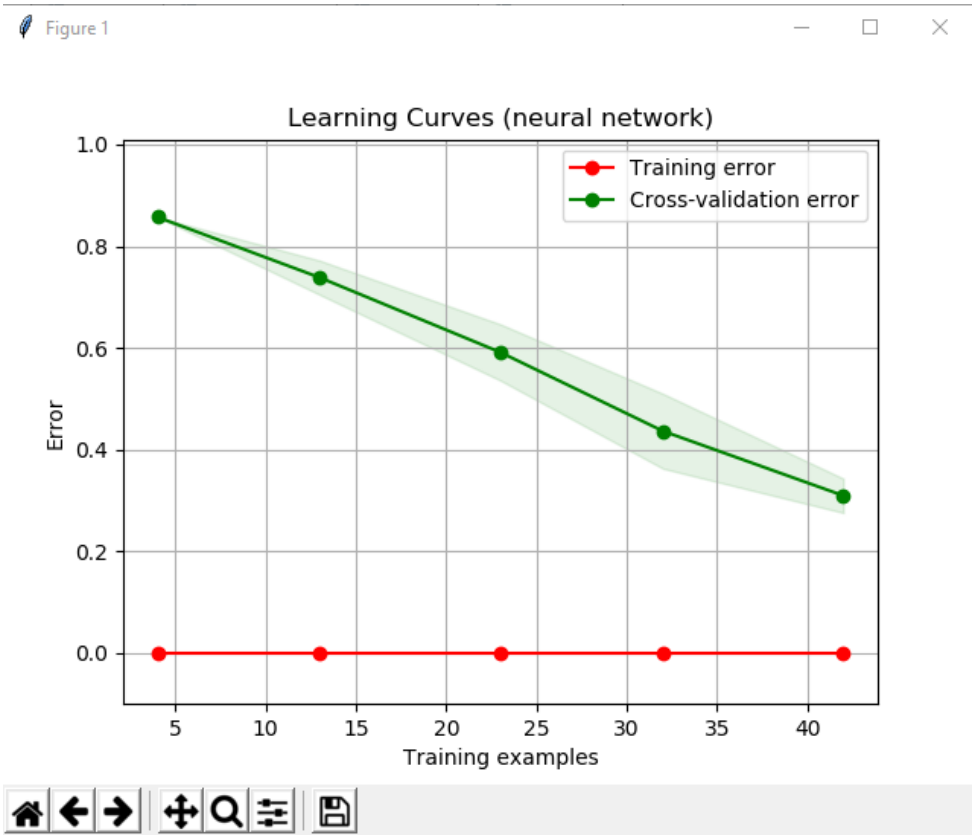

همان طور که واضح است شیب یادگیری داده های تست کمتر است نسبت به حالت قبل .

- نتایج برای mlp با ساختار لایه های :

```
MLPClassifier(solver='adam', activation='tanh',
early_stopping=True,learning_rate=.01, alpha=1e-5,
            hidden_layer_sizes=(1000,500,300,200,
100,50))
```

```
Running neural network...
accuracy on test data : 0.642857142857
accuracy on train data :  1.0
```

دقت کم مدل نسبت به حالت قبل را می توان به خاطر پیچیدگی زیاد مدل و overfit شدن توجیح کرد.
نمودار ان به صورت زیر است

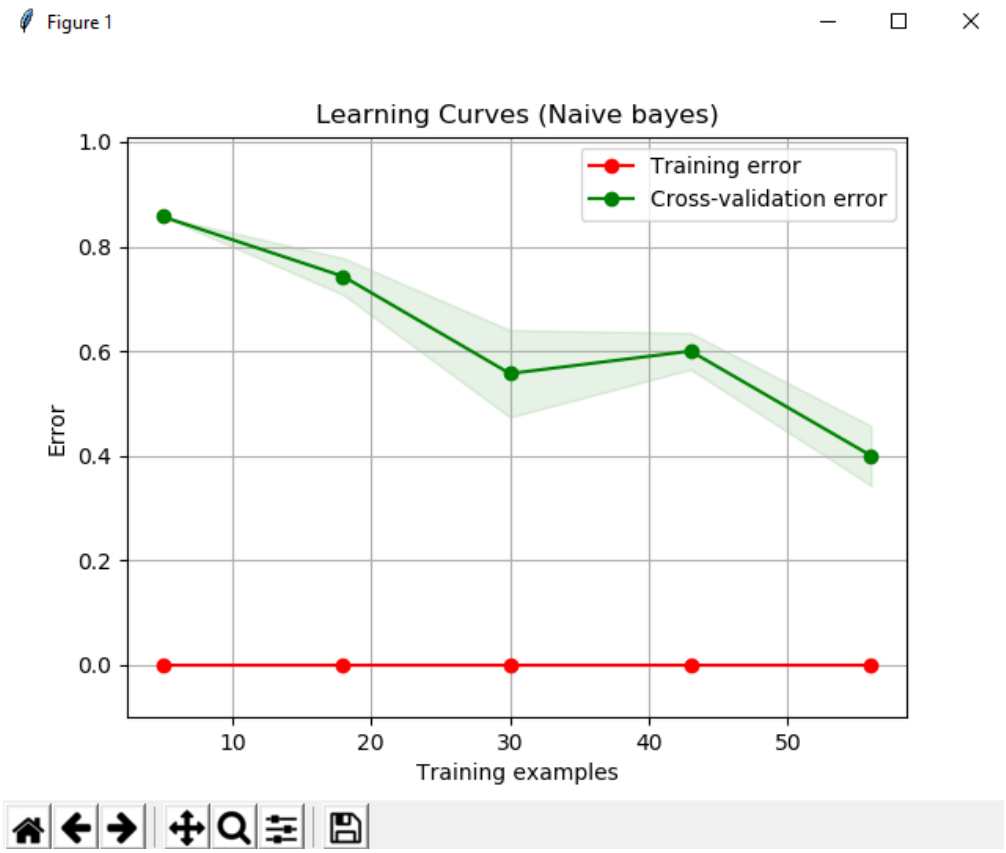



- نتایج برای naïve bayes

برای درک بهتر این کلاسیفایر نیز ازمایش شده است.

```
Running Naive bayes...
accuracy on test data : 0.5
accuracy on train data :  1.0
```

و نمودار ان :




و در اخر جدول مقایه نتایج :

| | MLP | Naïve bayes | Adaline |
|---|---|---|---|
| Train accuracy | 1 | 1 | .17 |
| Test accuracy | .71 | .5 | .14 |