
بسم الله الرحمن الرحيم

پروژه درس روشهای چندمتغیری گسسته – دانشگاه اراک – نیم‌سال ۰۰۱

تشخیص نویسه اعداد فارسی با روش های

رگرسیون لجستیک چند جمله ای و ماشین های بردار پشتیبان

Persian Optical character recognition With

Support Vector Machines & Multinomial Logistic Regression

نویسندگان

محراب عتیقی

آتوسا رستمی

شماره دانشجویی

۳۹۷۱۲۱۳۱۱۲۵

۳۹۷۱۲۱۳۱۰۸۷

استاد راهنما

سیدجمال میرکمالی

چکیده

سابقه نویسه خوانی نوری به بیش از نیم قرن گذشته برمی گردد. از زمانی که سیستم‌های کامپیوتری، در تجارت و صنعت وارد شد، نیاز به جمع‌آوری و پردازش دستخط به وسیله سیستم به وجود آمد. اما پاسخ به این نیازها با میزان تکنولوژی سیستم‌های موجود، محدود می‌شدند. **نویسه خوان نوری** که با سرواژه‌ی OCR شناخته می‌شود، عبارت است از تشخیص (recognition) خودکار متون موجود درتصاویر اسناد و تبدیل آن‌ها به متون قابل جستجو و ویرایش توسط رایانه. در این پروژه به بررسی مجموعه ارقام دستنویس هدی که اولین مجموعه‌ی بزرگ ارقام دستنویس فارسی است می پردازیم و نتایج حاصل را در طی گزارش برای پیشبینی مدل و دسته بندی داده های جدید و دقت مدل های برازش داده شده را بیان میکنیم.

کلیات

این مجوعه داده مشتمل بر **۱۰۲۳۵۳نمونه دستنوشته** سیاه سفید است. این مجموعه طی انجام یک پروژه‌ی کارشناسی ارشد درباره بازشناسی فرمهای دستنویس تهیه شده است . داده های این مجموعه از حدود ۱۲۰۰۰ فرم ثبت نام آزمون سراسری کارشناسی ارشد سال ۱۳۸۴ و آزمون کاردانی پیوسته‌ی دانشگاه جامع علمی کاربردی سال ۱۳۸۳ استخراج شده است. خصوصیات این مجموعه داده به شرح زیر است:

درجه تفکیک نمونه‌ها :۲۰۰ نقطه بر اینچ

تعداد کل نمونه‌ها :۱۰۲۳۵۲ نمونه

تعداد نمونه‌های آموزش :۶۰۰۰ نمونه از هر کلاس

تعداد نمونه‌های آزمایش :۲۰۰۰ نمونه از هر کلاس

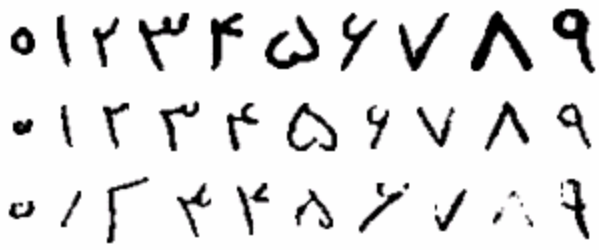
سایر نمونه‌ها :۲۲۳۵۲ نمونه

تعداد نمونه ها در هر کلاس

رقم ۰	رقم ۱	رقم۲	رقم ۳	رقم ۴	رقم ۵	رقم ۶	رقم ۷	رقم ۸	رقم ۹
۱۰۰۷۰	۱۰۳۳۰	۹۹۲۳	۱۰۳۳۴	۱۰۳۳۳	۱۰۱۱۰	۱۰۲۵۴	۱۰۳۶۳	۱۰۲۶۴	۱۰۳۷۱

نمونه هایی از دستخط های مختلف موجود در مجموعه ارقام دستنویس

نمونه هایی از کیفیتهای مختلف موجود در مجموعه ارقام دستنویس



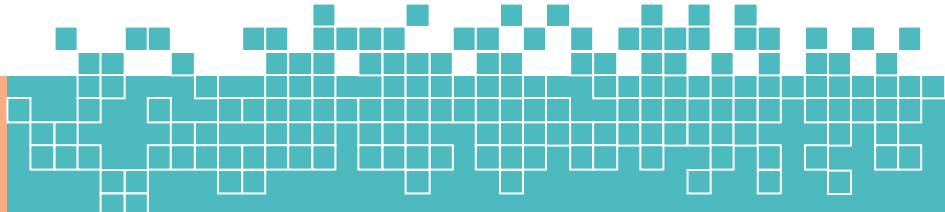
```
# Hoda Dataset Reader
# Python code for reading Hoda farsi digit dataset.
# import struct
# import numpy as np
# import cv2
def __convert_to_one_hot(vector, num_classes):
    result = np.zeros(shape=[len(vector), num_classes])
    result[np.arange(len(vector)), vector] = 1
    return result


def __resize_image(src_image, dst_image_height, dst_image_width):
    src_image_height = src_image.shape[0]
    src_image_width = src_image.shape[1]

    if src_image_height > dst_image_height or src_image_width > dst_ima
ge_width:
        height_scale = dst_image_height / src_image_height
        width_scale = dst_image_width / src_image_width
        scale = min(height_scale, width_scale)
        img = cv2.resize(src=src_image, dsize=(0, 0), fx=scale, fy=scal
e, interpolation=cv2.INTER_CUBIC)
    else:
        img = src_image

    img_height = img.shape[0]
    img_width = img.shape[1]

    dst_image = np.zeros(shape=[dst_image_height, dst_image_width], dty
pe=np.uint8)

    y_offset = (dst_image_height - img_height) // 2
    x_offset = (dst_image_width - img_width) // 2

    dst_image[y_offset:y_offset+img_height, x_offset:x_offset+img_width
] = img

    return dst_image


def read_hoda_cdb(file_name):
    with open(file_name, 'rb') as binary_file:
```

```
        data = binary_file.read()

        offset = 0

        # read private header

        yy = struct.unpack_from('H', data, offset)[0]
        offset += 2

        m = struct.unpack_from('B', data, offset)[0]
        offset += 1

        d = struct.unpack_from('B', data, offset)[0]
        offset += 1

        H = struct.unpack_from('B', data, offset)[0]
        offset += 1

        W = struct.unpack_from('B', data, offset)[0]
        offset += 1

        TotalRec = struct.unpack_from('I', data, offset)[0]
        offset += 4

        LetterCount = struct.unpack_from('128I', data, offset)
        offset += 128 * 4

        imgType = struct.unpack_from('B', data, offset)[0]  # 0: binary
, 1: gray
        offset += 1

        Comments = struct.unpack_from('256c', data, offset)
        offset += 256 * 1

        Reserved = struct.unpack_from('245c', data, offset)
        offset += 245 * 1

        if (W > 0) and (H > 0):
            normal = True
        else:
            normal = False

        images = []
        labels = []

        for i in range(TotalRec):

            StartByte = struct.unpack_from('B', data, offset)[0]  # mus
```

```
t be 0xff
            offset += 1

            label = struct.unpack_from('B', data, offset)[0]
            offset += 1

            if not normal:
                W = struct.unpack_from('B', data, offset)[0]
                offset += 1

                H = struct.unpack_from('B', data, offset)[0]
                offset += 1

            ByteCount = struct.unpack_from('H', data, offset)[0]
            offset += 2

            image = np.zeros(shape=[H, W], dtype=np.uint8)

            if imgType == 0:
                # Binary
                for y in range(H):
                    bWhite = True
                    counter = 0
                    while counter < W:
                        WBcount = struct.unpack_from('B', data, offset)
[0]
                        offset += 1
                        # x = 0
                        # while x < WBcount:
                        #     if bWhite:
                        #         image[y, x + counter] = 0  # Backgrou
nd
                        #     else:
                        #         image[y, x + counter] = 255  # ForeGr
ound
                        #     x += 1
                        if bWhite:
                            image[y, counter:counter + WBcount] = 0  #
Background
                        else:
                            image[y, counter:counter + WBcount] = 255
# ForeGround
                        bWhite = not bWhite  # black white black white
...
                        counter += WBcount
            else:
                # GrayScale mode
                data = struct.unpack_from('{}B'.format(W * H), data, of
```

```
fset)
                offset += W * H
                image = np.asarray(data, dtype=np.uint8).reshape([W, H]
).T

            images.append(image)
            labels.append(label)

        return images, labels


def read_hoda_dataset(dataset_path, images_height=32, images_width=32,
one_hot=False, reshape=True):
    images, labels = read_hoda_cdb(dataset_path)
    assert len(images) == len(labels)

    X = np.zeros(shape=[len(images), images_height, images_width], dtyp
e=np.float32)
    Y = np.zeros(shape=[len(labels)], dtype=np.int)

    for i in range(len(images)):
        image = images[i]
        # Image resizing.
        image = __resize_image(src_image=image, dst_image_height=images
_height, dst_image_width=images_width)
        # Image normalization.
        image = image / 255
        # Image binarization.
        image = np.where(image >= 0.5, 1, 0)
        # Image.
        X[i] = image
        # Label.
        Y[i] = labels[i]

    if one_hot:
        Y = __convert_to_one_hot(Y, 10).astype(dtype=np.float32)
    else:
        Y = Y.astype(dtype=np.float32)

    if reshape:
        X = X.reshape(-1, images_height * images_width)
    else:
        X = X.reshape(-1, images_height, images_width, 1)

    return X, Y
```

```
# *-* coding: utf-8 *-*

from matplotlib import pyplot as plt
from HodaDatasetReader import read_hoda_cdb, read_hoda_dataset

print('########################################################################
################')
print()

# type(train_images):  <class 'list'>
# len(train_images):  60000
#
# type(train_images[ i ]): <class 'numpy.ndarray'>
# train_images[ i ].dtype: uint8
# train_images[ i ].min(): 0
# train_images[ i ].max(): 255
# train_images[ i ].shape: (HEIGHT, WIDTH)
#
# type(train_labels):  <class 'list'>
# len(train_labels):  60000
#
# type(train_labels[ i ]): <class 'int'>
# train_labels[ i ]: 0...9
print('Reading Train 60000.cdb ...')
train_images, train_labels = read_hoda_cdb('./DigitDB/Train 60000.cdb')


# type(test_images):  <class 'list'>
# len(test_images):  20000
#
# type(test_images[ i ]): <class 'numpy.ndarray'>
# test_images[ i ].dtype: uint8
# test_images[ i ].min(): 0
# test_images[ i ].max(): 255
# test_images[ i ].shape: (HEIGHT, WIDTH)
#
# type(test_labels):  <class 'list'>
# len(test_labels):  20000
#
# type(test_labels[ i ]): <class 'int'>
# test_labels[ i ]: 0...9
print('Reading Test 20000.cdb ...')
test_images, test_labels = read_hoda_cdb('./DigitDB/Test 20000.cdb')


# type(remaining_images):  <class 'list'>
# len(remaining_images):  22352
#
```

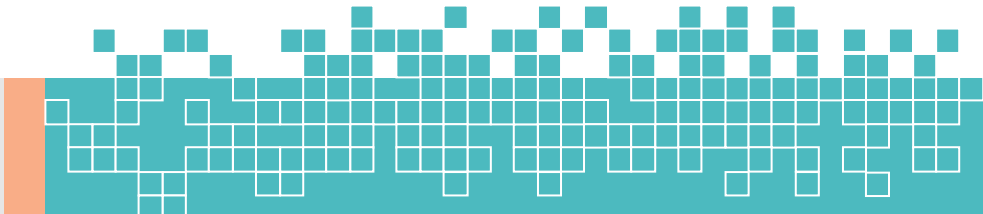
```
# type(remaining_images[ i ]): <class 'numpy.ndarray'>
# remaining_images[ i ].dtype: uint8
# remaining_images[ i ].min(): 0
# remaining_images[ i ].max(): 255
# remaining_images[ i ].shape: (HEIGHT, WIDTH)
#
# type(remaining_labels):  <class 'list'>
# len(remaining_labels):  22352
#
# type(remaining_labels[ i ]): <class 'int'>
# remaining_labels[ i ]: 0...9
print('Reading RemainingSamples.cdb ...')
remaining_images, remaining_labels = read_hoda_cdb('./DigitDB/Remaining
Samples.cdb')

print()

# ********************************************************************
*********

print('type(train_images): ', type(train_images))
print('len(train_images): ', len(train_images))
print()

print('type(train_labels): ', type(train_labels))
print('len(train_labels): ', len(train_labels))
print()

fig = plt.figure(figsize=(15, 4))
for i in range(4):

    print('---------------------------------------')
    print()

    print('type(train_images[', i, ']):', type(train_images[i]))
    print('train_images[', i, '].dtype:', train_images[i].dtype)
    print('train_images[', i, '].min():', train_images[i].min())
    print('train_images[', i, '].max():', train_images[i].max())
    print('train_images[', i, '].shape = (HEIGHT, WIDTH):', train_image
s[i].shape)
    print()

    print('type(train_labels[', i, ']):', type(train_labels[i]))
    print('train_labels[', i, ']:', train_labels[i])
    print()

    fig.add_subplot(1, 4, i + 1)
    plt.title('train_labels[' + str(i) + '] = ' + str(train_labels[i]))
```

```
    plt.imshow(train_images[i], cmap='gray')

plt.show()

print('##################################################################
################')
print()

# type(X_train):  <class 'numpy.ndarray'>
# X_train.dtype: float32
# X_train.shape: (reshape=True),  (60000, 1024)
#
# type(X_train[ i ]): <class 'numpy.ndarray'>
# X_train[ i ].dtype: float32
# X_train[ i ].min(): 0.0
# X_train[ i ].max(): 1.0
# X_train[ i ].shape = (HEIGHT*WIDTH,): (reshape=True),  (1024,)
#
# type(Y_train):  <class 'numpy.ndarray'>
# Y_train.dtype: float32
# Y_train.shape: (one_hot=False),  (60000,)
#
# type(Y_train[ i ]): <class 'numpy.float32'>
# Y_train[ i ].dtype: float32
# Y_train[ i ]: (one_hot=False),  0...9
print('Reading train dataset (Train 60000.cdb)...')
X_train, Y_train = read_hoda_dataset(dataset_path='./DigitDB/Train 6000
0.cdb',
                                     images_height=32,
                                     images_width=32,
                                     one_hot=False,
                                     reshape=True)


# type(X_test):  <class 'numpy.ndarray'>
# X_test.dtype: float32
# X_test.shape: (reshape=False),  (20000, 32, 32, 1)
#
# type(X_test[ i ]): <class 'numpy.ndarray'>
# X_test[ i ].dtype: float32
# X_test[ i ].min(): 0.0
# X_test[ i ].max(): 1.0
# X_test[ i ].shape = (HEIGHT, WIDTH, CHANNEL): (reshape=False),  (32,
32, 1)
#
# type(Y_test):  <class 'numpy.ndarray'>
# Y_test.dtype: float32
# Y_test.shape: (one_hot=True),  (20000, 10)
```

```
#
# type(Y_test[ i ]): <class 'numpy.ndarray'>
# Y_test[ i ].dtype: float32
# Y_test[ i ].min(): 0.0
# Y_test[ i ].max(): 1.0
# Y_test[ 0 ]: (one_hot=True),  [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
print('Reading test dataset (Test 20000.cdb)...')
X_test, Y_test = read_hoda_dataset(dataset_path='./DigitDB/Test 20000.c
db',
                                   images_height=32,
                                   images_width=32,
                                   one_hot=True,
                                   reshape=False)


# type(X_remaining):  <class 'numpy.ndarray'>
# X_remaining.dtype: float32
# X_remaining.shape: (reshape=True),  (22352, 1024)
#
# type(X_remaining[ i ]): <class 'numpy.ndarray'>
# X_remaining[ i ].dtype: float32
# X_remaining[ i ].min(): 0.0
# X_remaining[ i ].max(): 1.0
# X_remaining[ i ].shape = (HEIGHT*WIDTH,): (reshape=True),  (1024,)
#
# type(Y_remaining):  <class 'numpy.ndarray'>
# Y_remaining.dtype: float32
# Y_remaining.shape: (one_hot=True),  (22352, 10)
#
# type(Y_remaining[ i ]): <class 'numpy.ndarray'>
# Y_remaining[ i ].dtype: float32
# Y_remaining[ i ].min(): 0.0
# Y_remaining[ i ].max(): 1.0
# Y_remaining[ 0 ]: (one_hot=True),  [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
print('Reading remaining samples dataset (RemainingSamples.cdb)...')
X_remaining, Y_remaining = read_hoda_dataset('./DigitDB/RemainingSample
s.cdb',
                                             images_height=32,
                                             images_width=32,
                                             one_hot=True,
                                             reshape=True)

print()

# ****************************************************************
*********

print('type(X_train): ', type(X_train))
```

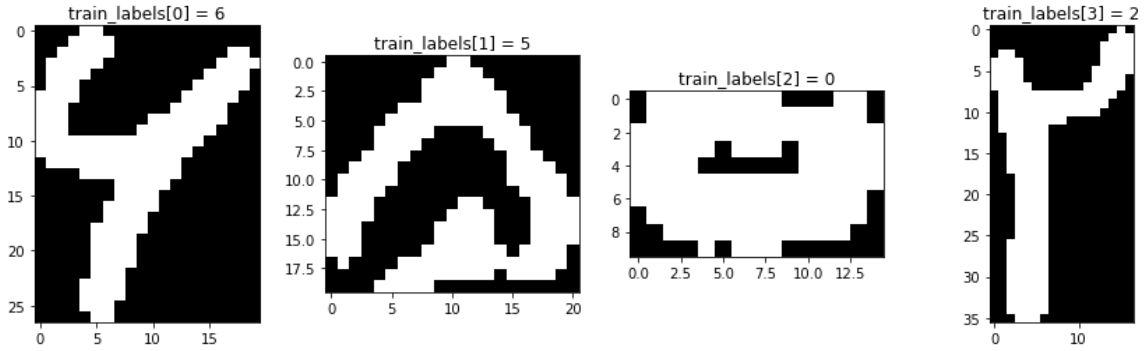
```
print('X_train.dtype:', X_train.dtype)
print('X_train.shape: (reshape=True), ', X_train.shape)
print()

print('type(Y_train): ', type(Y_train))
print('Y_train.dtype:', Y_train.dtype)
print('Y_train.shape: (one_hot=False), ', Y_train.shape)
print()

print('type(X_test): ', type(X_test))
print('X_test.dtype:', X_test.dtype)
print('X_test.shape: (reshape=False), ', X_test.shape)
print()

print('type(Y_test): ', type(Y_test))
print('Y_test.dtype:', Y_test.dtype)
print('Y_test.shape: (one_hot=True), ', Y_test.shape)
print()

print('type(X_remaining): ', type(X_remaining))
print('X_remaining.dtype:', X_remaining.dtype)
print('X_remaining.shape: (reshape=True), ', X_remaining.shape)
print()

print('type(Y_remaining): ', type(Y_remaining))
print('Y_remaining.dtype:', Y_remaining.dtype)
print('Y_remaining.shape: (one_hot=True), ', Y_remaining.shape)
print()

fig = plt.figure(figsize=(16, 3))

print('----------------------------------------')
print()

print('type(X_train[ 0 ]):', type(X_train[0]))
print('X_train[ 0 ].dtype:', X_train[0].dtype)
print('X_train[ 0 ].min():', X_train[0].min())
print('X_train[ 0 ].max():', X_train[0].max())
print('X_train[ 0 ].shape = (HEIGHT*WIDTH,): (reshape=True), ', X_train
[0].shape)
print()

print('type(Y_train[ 0 ]):', type(Y_train[0]))
print('Y_train[ 0 ].dtype:', Y_train[0].dtype)
print('Y_train[ 0 ]: (one_hot=False), ', Y_train[0])
print()

fig.add_subplot(1, 3, 1)
```

```
plt.title('Y_train[ 0 ] = ' + str(Y_train[0]))
plt.imshow(X_train[0].reshape([32, 32]), cmap='gray')

print('----------------------------------------')
print()

print('type(X_test[ 0 ]):', type(X_test[0]))
print('X_test[ 0 ].dtype:', X_test[0].dtype)
print('X_test[ 0 ].min():', X_test[0].min())
print('X_test[ 0 ].max():', X_test[0].max())
print('X_test[ 0 ].shape = (HEIGHT, WIDTH, CHANNEL): (reshape=False), '
, X_test[0].shape)
print()

print('type(Y_test[ 0 ]):', type(Y_test[0]))
print('Y_test[ 0 ].dtype:', Y_test[0].dtype)
print('Y_test[ 0 ].min():', Y_test[0].min())
print('Y_test[ 0 ].max():', Y_test[0].max())
print('Y_test[ 0 ]: (one_hot=True), ', Y_test[0])
print()

fig.add_subplot(1, 3, 2)
plt.title('Y_test[ 0 ] = ' + str(Y_test[0]))
plt.imshow(X_test[0].reshape([32, 32]), cmap='gray')

print('----------------------------------------')
print()

print('type(X_remaining[ 0 ]):', type(X_remaining[0]))
print('X_remaining[ 0 ].dtype:', X_remaining[0].dtype)
print('X_remaining[ 0 ].min():', X_remaining[0].min())
print('X_remaining[ 0 ].max():', X_remaining[0].max())
print('X_remaining[ 0 ].shape = (HEIGHT*WIDTH,): (reshape=True), ', X_r
emaining[0].shape)
print()

print('type(Y_remaining[ 0 ]):', type(Y_remaining[0]))
print('Y_remaining[ 0 ].dtype:', Y_remaining[0].dtype)
print('Y_remaining[ 0 ].min():', Y_remaining[0].min())
print('Y_remaining[ 0 ].max():', Y_remaining[0].max())
print('Y_remaining[ 0 ]: (one_hot=True), ', Y_remaining[0])
print()

fig.add_subplot(1, 3, 3)
plt.title('Y_remaining[ 0 ] = ' + str(Y_remaining[0]))
plt.imshow(X_remaining[0].reshape([32, 32]), cmap='gray')

plt.show()
```

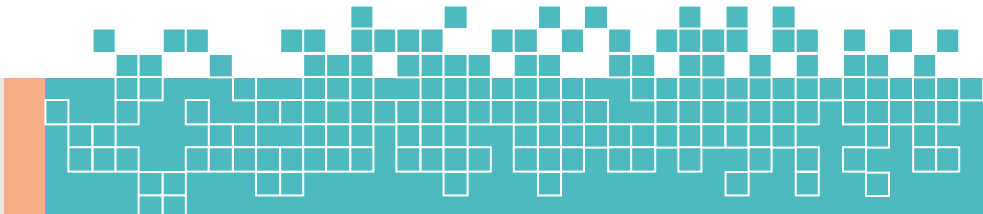



```
print('#####################################################################
################')
print()

#Now we want to save these data as csv in our pc:

import pandas as pd
import numpy as np
X_test = X_test.reshape(20000, 1024)
df1 = pd.DataFrame(X_train)
df2 = pd.DataFrame(Y_train)
df3 = pd.DataFrame(X_test)
df4 = pd.DataFrame(Y_test)
df1.to_csv("X_train.csv" , index = False)
df2.to_csv("Y_train.csv" , index = False)
df3.to_csv("X_test.csv" , index = False)
df4.to_csv("Y_test.csv" , index = False)

#####################################################################
#########

Reading Train 60000.cdb ...
Reading Test 20000.cdb ...
Reading RemainingSamples.cdb ...

type(train_images):  <class 'list'>
len(train_images):  60000

type(train_labels):  <class 'list'>
len(train_labels):  60000

---------------------------------------

type(train_images[ 0 ]): <class 'numpy.ndarray'>
train_images[ 0 ].dtype: uint8
train_images[ 0 ].min(): 0
train_images[ 0 ].max(): 255
train_images[ 0 ].shape = (HEIGHT, WIDTH): (27, 20)

type(train_labels[ 0 ]): <class 'int'>
train_labels[ 0 ]: 6

---------------------------------------

type(train_images[ 1 ]): <class 'numpy.ndarray'>
train_images[ 1 ].dtype: uint8
train_images[ 1 ].min(): 0
```

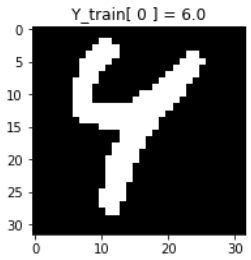
```
train_images[ 1 ].max(): 255
train_images[ 1 ].shape = (HEIGHT, WIDTH): (20, 21)

type(train_labels[ 1 ]): <class 'int'>
train_labels[ 1 ]: 5

----------------------------------------

type(train_images[ 2 ]): <class 'numpy.ndarray'>
train_images[ 2 ].dtype: uint8
train_images[ 2 ].min(): 0
train_images[ 2 ].max(): 255
train_images[ 2 ].shape = (HEIGHT, WIDTH): (10, 15)

type(train_labels[ 2 ]): <class 'int'>
train_labels[ 2 ]: 0

----------------------------------------

type(train_images[ 3 ]): <class 'numpy.ndarray'>
train_images[ 3 ].dtype: uint8
train_images[ 3 ].min(): 0
train_images[ 3 ].max(): 255
train_images[ 3 ].shape = (HEIGHT, WIDTH): (36, 17)

type(train_labels[ 3 ]): <class 'int'>
train_labels[ 3 ]: 2
```

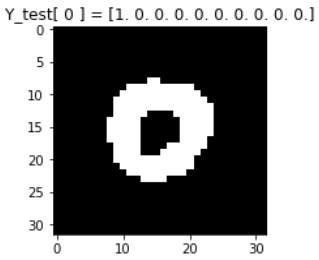


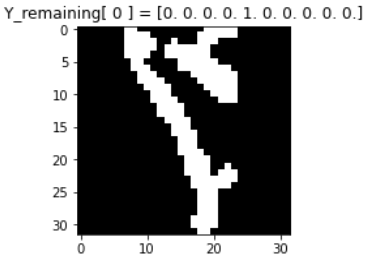
```
#########################################################################
#########

Reading train dataset (Train 60000.cdb)...
Reading test dataset (Test 20000.cdb)...
Reading remaining samples dataset (RemainingSamples.cdb)...

type(X_train):  <class 'numpy.ndarray'>
X_train.dtype: float32
X_train.shape: (reshape=True),  (60000, 1024)

type(Y_train):  <class 'numpy.ndarray'>
Y_train.dtype: float32
Y_train.shape: (one_hot=False),  (60000,)

type(X_test):  <class 'numpy.ndarray'>
X_test.dtype: float32
X_test.shape: (reshape=False),  (20000, 32, 32, 1)

type(Y_test):  <class 'numpy.ndarray'>
Y_test.dtype: float32
Y_test.shape: (one_hot=True),  (20000, 10)

type(X_remaining):  <class 'numpy.ndarray'>
X_remaining.dtype: float32
X_remaining.shape: (reshape=True),  (22352, 1024)

type(Y_remaining):  <class 'numpy.ndarray'>
Y_remaining.dtype: float32
Y_remaining.shape: (one_hot=True),  (22352, 10)

----------------------------------------

type(X_train[ 0 ]): <class 'numpy.ndarray'>
X_train[ 0 ].dtype: float32
X_train[ 0 ].min(): 0.0
X_train[ 0 ].max(): 1.0
X_train[ 0 ].shape = (HEIGHT*WIDTH,): (reshape=True),  (1024,)

type(Y_train[ 0 ]): <class 'numpy.float32'>
Y_train[ 0 ].dtype: float32
Y_train[ 0 ]: (one_hot=False),  6.0

----------------------------------------

type(X_test[ 0 ]): <class 'numpy.ndarray'>
X_test[ 0 ].dtype: float32
X_test[ 0 ].min(): 0.0
```

```
X_test[ 0 ].max(): 1.0
X_test[ 0 ].shape = (HEIGHT, WIDTH, CHANNEL): (reshape=False),  (32, 32
, 1)

type(Y_test[ 0 ]): <class 'numpy.ndarray'>
Y_test[ 0 ].dtype: float32
Y_test[ 0 ].min(): 0.0
Y_test[ 0 ].max(): 1.0
Y_test[ 0 ]: (one_hot=True),  [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

----------------------------------------

type(X_remaining[ 0 ]): <class 'numpy.ndarray'>
X_remaining[ 0 ].dtype: float32
X_remaining[ 0 ].min(): 0.0
X_remaining[ 0 ].max(): 1.0
X_remaining[ 0 ].shape = (HEIGHT*WIDTH,): (reshape=True),  (1024,)

type(Y_remaining[ 0 ]): <class 'numpy.ndarray'>
Y_remaining[ 0 ].dtype: float32
Y_remaining[ 0 ].min(): 0.0
Y_remaining[ 0 ].max(): 1.0
Y_remaining[ 0 ]: (one_hot=True),  [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```







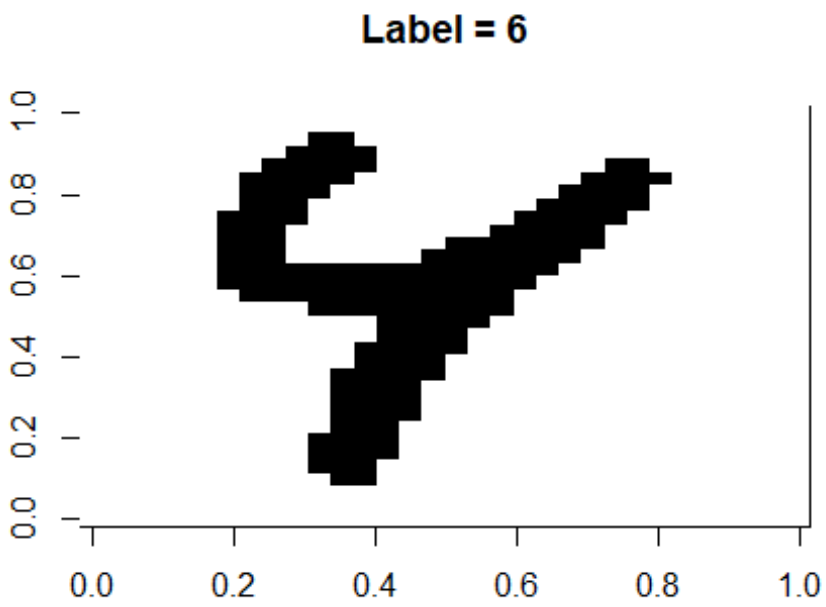
بررسی داده ها با مدل لجستیک چند جمله ای با نرم افزار R

بعد از پیش پردازش داده ها با پایتون آماده سازی فایل داده ها برای استفاده در نرم افزار R به پیاده سازی و تحلیل آنها مشابه مجموعه داده mnist میپردازیم.

```
Data=load("HodaDigits.RData")
dim(Pictures)

## [1] 60000  1024

dim(Pictures.test)

## [1] 20000  1024

length(Labels)

## [1] 60000
```

ابعاد داده ها ۶۰۰۰۰ در ۱۰۲۴ =۳۲*۳۲ است.

```
df <- data.frame(Labels = Labels, Pictures = Pictures)
row <- 1
image(matrix(Pictures[row, ], 32)[, 32:1],
      col = grey(seq(1,0,length=256)),
      main = paste0("Label = ", Labels[row]))
```



امتحان کردن نمایش تصویر ها در نرم افزار R به صورت بالاست. همان طور که مشاهده می شود عدد اولین سطر عدد ۶ می باشد.

```
Labels <- factor(Labels)

table(Labels)

## Labels
##    0    1    2    3    4    5    6    7    8    9
## 6000 6000 6000 6000 6000 6000 6000 6000 6000 6000
```

در این خروجی که از لیبل های داده های آموزشی دریافت کردیم تمامی اعداد به طور مساوی در مدل قرار دارند. یعنی مدل شامل ۶۰۰۰ عدد صفر ، ۶۰۰۰ عدد ۱ و... است.

برای اجرای مدل لجستیک چند جمله ای نیاز داریم که داده های Labels که نقش متغیر y را بازی میکنند به صورت factor در نظر بگیریم برای اجرای این مدل هم نیاز به پکیج nnet داریم.

این پکیج برای اجرای مدل های شبکه عصبی پیشرو و مدل لگاریتم خطی چند جمله ای کاربرد دارد.

با اجرای دستورات این پکیج سعی بر اجرای مدل چند جمله ای داریم در ادامه به معرفی آرگومان های اجرا شده در این دستورات خواهیم پرداخت.

```
library(nnet)
#m1<- mulitnom (Labels~ . -1 ,MaxNWts=11000,data=df)
#save(m1, file =  "train model .RData" )
```

آرگومان های کلی دستور mulitnom :

formula: response ~ predictors

به صورت مدل رگرسیونی نوشته میشود مدل بدون عرض از مبدا در نظر گرفته شده است.

MaxNWts:

ماکسیمم ضرایبی که محاسبه میکند حداکثر چقدرباشد چرا که به صورت پیش فرض مقادیر بیشتر از ۱۰۰۰ را مدل نمی پذیرد با این کار اجازه محاسبه ضرایب بیشتر از ۱۰۰۰ را باتوجه به حجم داده ها می دهیم.

data:

داده های آموزشی که مدل روی آن اجرا میشود.

نکته : با توجه به زمان بر بودن ساخت مدل ، مدل را ذخیره می کنیم و از این پس با مدل ذخیره شده کار خواهیم کرد.

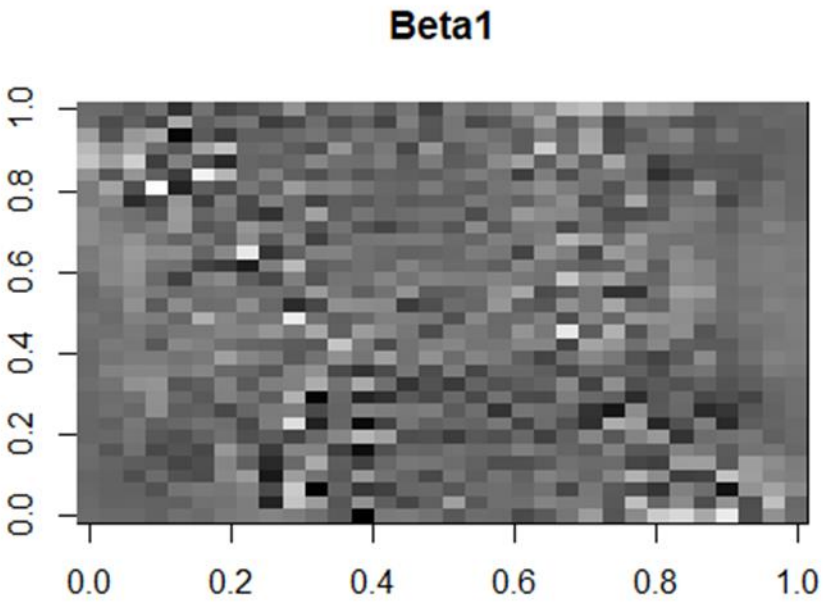
```
load("train model .RData")
beta.m1 <- coef(m1)
dim(beta.m1)

## [1]    9 1024
```

در هر سطر یک بردار ۱۰۲۴ از ضرایب بتا ها داریم که شامل β_9 ... β_1 می باشد.

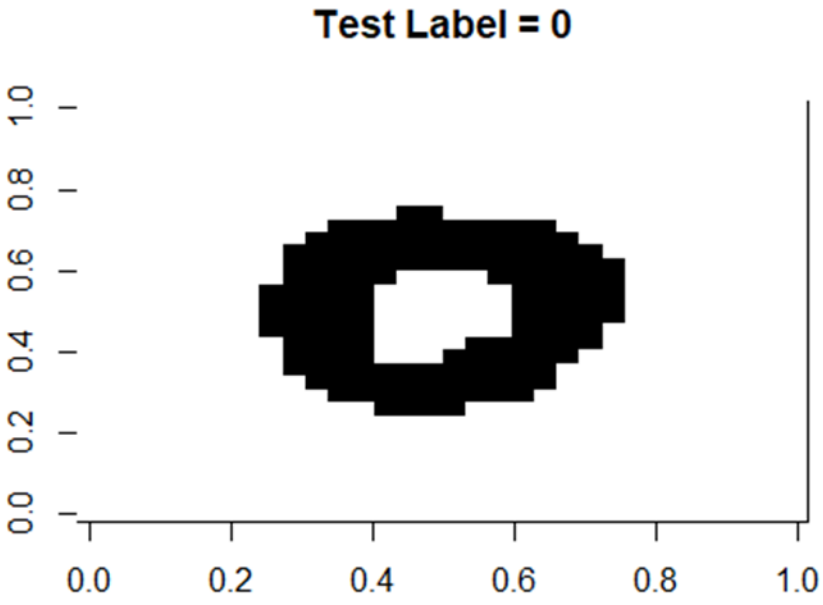
حال برای مصور سازی بتا ها آنها را در قالب ماتریس ۳۲*۳۲ رسم میکنیم خروجی کد زیر شامل یک پلات شامل تشخیص طیف رنگی تیره و روشن است طیف خاکستری نشان دهنده فضای خالی تصویراست (پیسکل های سفید). هرچه به سمت تیرگی پیش میرود نشان دهنده پیکسل های مشکی شامل نوشته ها است .

```
row = 1

image(matrix(beta.m1[row, ], 32)[, 32:1],
      col = grey(seq(1,0,length=256)),
      main = paste0("Beta", row))
```

به عنوان مثال برای ضرایب ستون اول بتا ها به این صورت رسم می شوند.




```
row <- 1
image(matrix(Pictures.test[row, ], 32)[,32:1],
      col = grey(seq(1,0,length=256)),
      main = paste0("Test Label = ", Labels.test[row]))
```



برای پاسخ به این سوال که آیا مدل ما می تواند برای مجموعه داده هایی که در مدل وجود ندارد هم پیشبینی درستی داشته باشد یا خیر؟ کد های زیر را با مجموعه داده های آزمایش به جای داده های آموزشی اجرا خواهیم کرد و دقت مدل، با مجموعه داده های آزمایش را اندازه گیری می کنیم.

```
data1 <- data.frame(t(Pictures.test[row,]))
names(data1) = names(df[,-1])
predict(m1, newdata = data1)

## [1] 0
## Levels: 0 1 2 3 4 5 6 7 8 9

round(predict(m1, newdata = data1, type = "p"), 2)

## 0 1 2 3 4 5 6 7 8 9
## 1 0 0 0 0 0 0 0 0 0
```

همانطور که مشاهده می شود برای یک داده آزمایشی که در مدل وجود ندارد تصویر به درستی تشخیص داده شده و با احتمال ۱ دقیقا صفر پیشبینی شده است.

حال با محاسبه ماتریس در هم ریختگی میتوان به این موضوع دست یافت که لیبل های پیشبینی شده با عکس اصلی هم خوانی دارد یا خیر؟

```
df.test <- data.frame(Pictures.test)
names(df.test) = names(df[,-1])

Predict.test <- predict(m1, newdata = df.test)
(CM <- table(Predict.test, Labels.test))

##             Labels.test
## Predict.test    0    1    2    3    4    5    6    7    8    9
##            0 1956   11   24   25   41   26   68   69   49   30
##            1    2 1923   73    7    6   19   20   16    4   44
##            2    1   32 1696  135   36   13   62   24   32   23
##            3    0    0   66 1668   89    5   14    4    5   10
##            4    5    2   55  110 1720   66   29    6   25   34
##            5   18    3    2    1   16 1773    8   32   29    4
##            6   12    4   27   27   25   27 1657   19   34   48
##            7    5   14    5   11   11   30   27 1821    7    2
##            8    1    2    6    1    7   37   23    9 1777   14
##            9    0    9   46   15   49    4   92    0   38 1791

(Accuracy <- sum(diag(CM)) / nrow(Pictures.test) * 100)

## [1] 88.91
```

```
table(Labels.test)

## Labels.test
##    0    1    2    3    4    5    6    7    8    9
## 2000 2000 2000 2000 2000 2000 2000 2000 2000 2000

CM.percent <- scale(CM, center = F, scale = colSums(CM)) * 100
round(CM.percent)

##             Labels.test
## Predict.test  0  1  2  3  4  5  6  7  8  9
##            0 98  1  1  1  2  1  3  3  2  2
##            1  0 96  4  0  0  1  1  1  0  2
##            2  0  2 85  7  2  1  3  1  2  1
##            3  0  0  3 83  4  0  1  0  0  0
##            4  0  0  3  6 86  3  1  0  1  2
##            5  1  0  0  0  1 89  0  2  1  0
##            6  1  0  1  1  1  1 83  1  2  2
##            7  0  1  0  1  1  2  1 91  0  0
##            8  0  0  0  0  0  2  1  0 89  1
##            9  0  0  2  1  2  0  5  0  2 90
```

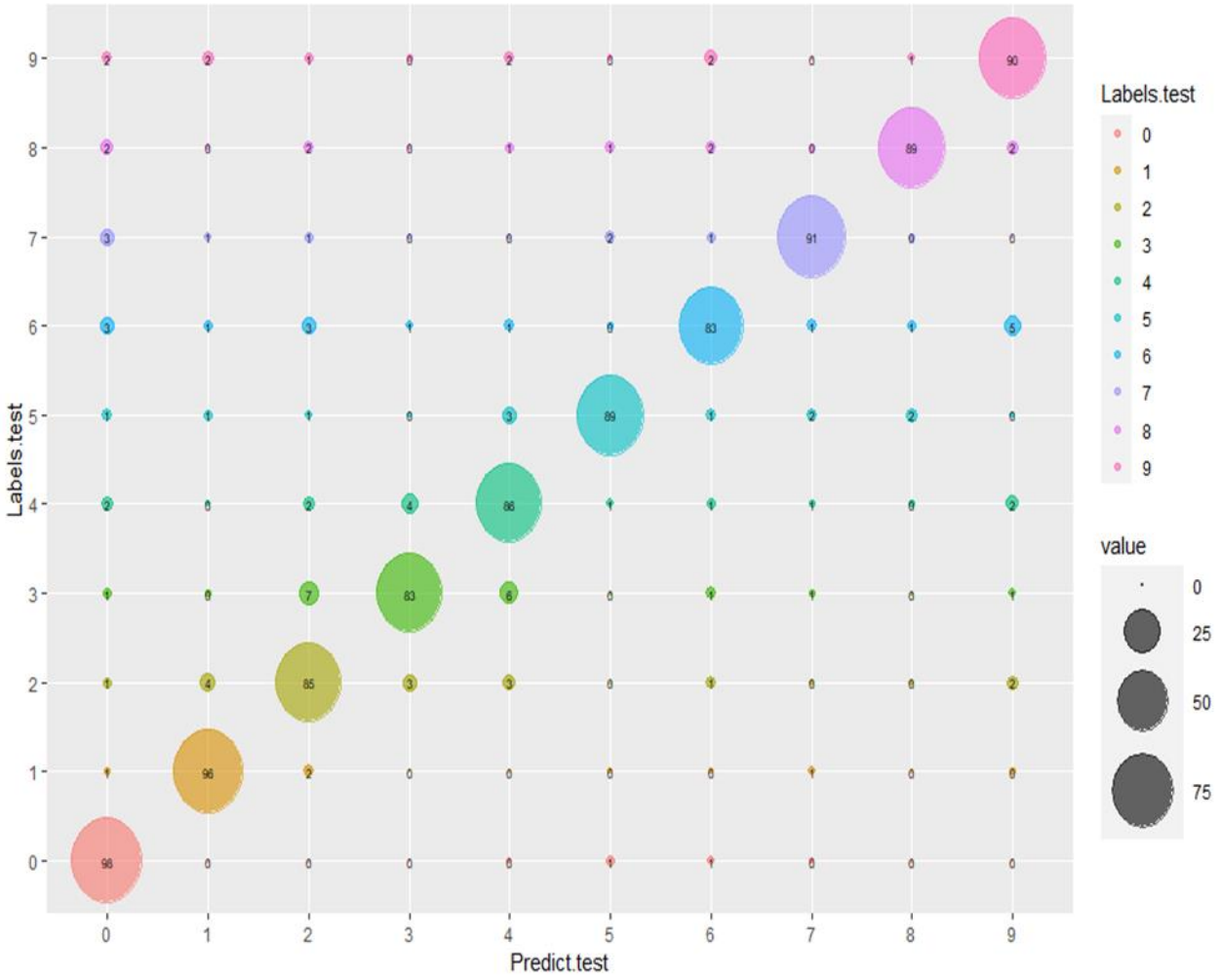
اعداد روی قطر اصلی ماتریس نمایانگر مقادیر درست پیشبینی شده از داده ها می باشند.

دراین رشته کد ها درصد هر مشاهده به تفکیک ستونی بدون مرکزی سازی نیز محاسبه شده و دقت مدل را نیز برآورد نموده ایم که دقت مدل برای پیشبینی مقادیر جدید برابر ۸۸,۹۱ می باشد.

حال برای درک بهتر مفاهیم گفته شده نمودار تشخیص تصاویر و قدرت پیشبینی را رسم می کنیم.

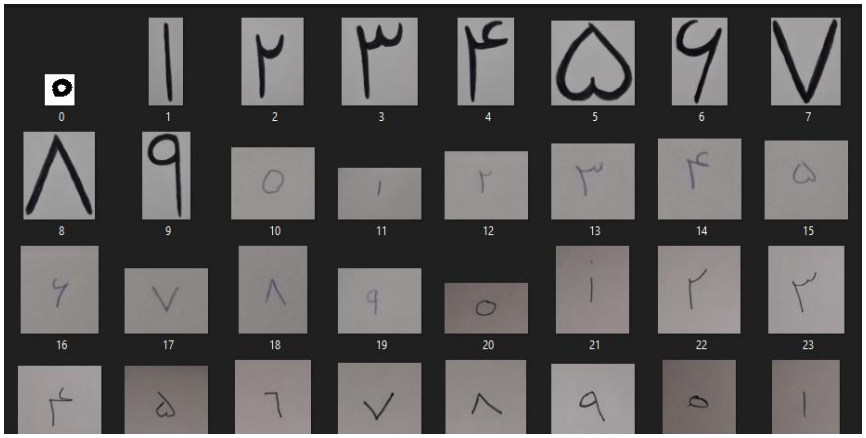
برای این منظور از پکیج مصور سازی ggplot2 استفاده می کنیم:

```
CM.long <- reshape2::melt(CM.percent)
CM.long$Labels.test <- factor(CM.long$Labels.test)
CM.long$Predict.test <- factor(CM.long$Predict.test)

library(ggplot2)

ggplot(CM.long, aes(x = Predict.test, y = Labels.test, label = round(va
lue))) +
  geom_point(aes(color = Labels.test, size = value), alpha = 0.6) +
  geom_text(size = 2) + scale_size(range = c(0.2, 18.5))
```



حال تصمیم بر این داریم که از مجموعه دست نوشته هایی که خود به تعداد ۷۰ تصویر جمع آوری کرده ایم، به عنوان داده های آزمایش استفاده کنیم:

نمونه هایی از دستخط های مختلف جمع آوری شده



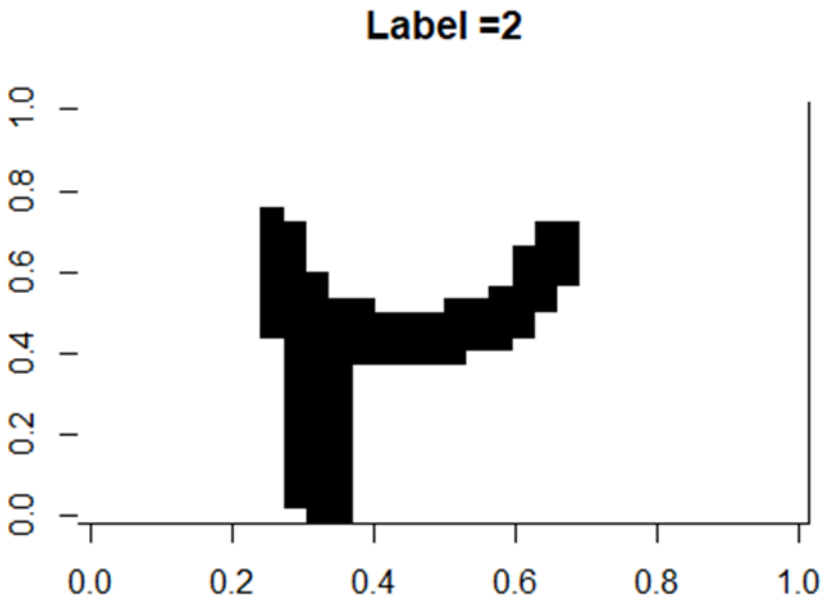
```
library(imagefx)

## Warning: package 'imagefx' was built under R version 4.1.2

library(imager)

## Warning: package 'imager' was built under R version 4.1.2

## Loading required package: magrittr

##
## Attaching package: 'imager'

## The following object is masked from 'package:magrittr':
##
##     add

## The following objects are masked from 'package:stats':
##
##     convolve, spectrum

## The following object is masked from 'package:graphics':
##
##     frame

## The following object is masked from 'package:base':
##
##     save.image

library(OpenImageR)

## Warning: package 'OpenImageR' was built under R version 4.1.2
```

```
library("EBImage")

##
## Attaching package: 'EBImage'

## The following objects are masked from 'package:OpenImageR':
##
##     readImage, writeImage

## The following objects are masked from 'package:imager':
##
##     channel, dilate, display, erode, resize, watershed

library(magick)

## Warning: package 'magick' was built under R version 4.1.2

## Linking to ImageMagick 6.9.12.3
## Enabled features: cairo, freetype, fftw, ghostscript, heic, lcms, pa
ngo, raw, rsvg, webp
## Disabled features: fontconfig, x11

n=70
data = matrix(rep(0,n*1024),nrow= n )
for(i in 0:69){
im <- readImage(paste0("real-test-pic/",i,".jpg"))
m = sort(dim(im))[2]
im <- image_read(paste0("real-test-pic/",i,".jpg"))
im = image_convert(im , type = 'Bilevel')
im = image_crop(im, geometry_area(m, m), repage = FALSE)
im = image_resize(im, geometry_size_pixels(32, 32, preserve_aspect = FA
LSE))
image_write(im, path = paste0("real-test-pic/",100*i,".jpg"),
            format = "jpg" , quality = 100)
im <- readImage(paste0("real-test-pic/",100*i,".jpg"))
data[i+1,]= abs(1-round(im))
}
```

بعد از پردازش تصویر به اجرای مدل می پردازیم:

```
Data=load("HodaDigits.RData")

df <- data.frame(Labels = Labels, Pictures = Pictures)


library(nnet)

#m1 <- multinom(Labels ~ . - 1, MaxNWts = 70000, data = df)
#save(m1, file = "F:/lessons/Gosaste/data/Hodamodel.RData")
load("train model .RData")
```

```r
beta.m1 <- coef(m1)
dim(beta.m1)

## [1]    9 1024



row=3
image(matrix(data[row,], 32)[, 32:1], row=3
image(matrix(data[row,], 32)[, 32:1],
      col = grey(c(1,0)),
      main = paste0("Label =", row-1 ))
```




```r
data_test <- data.frame(data)
names(data_test) = names(df[,-1])
round(predict(m1, newdata = data_test[row,], type = "p"), 2)

##    0    1    2    3    4    5    6    7    8    9
## 0.09 0.00 0.00 0.00 0.00 0.00 0.00 0.90 0.00 0.00

pp = predict(m1, newdata = data_test)
l = rep(c(0:9),n/10)
(CM <- table(pp, l))
```

```
##    l
## pp  0 1 2 3 4 5 6 7 8 9
##   0 3 0 2 2 2 2 0 1 1 1
##   1 0 1 1 0 0 0 0 1 1 2
##   2 0 1 1 0 0 0 0 1 0 1
##   3 1 0 0 1 0 2 0 0 0 0
##   4 0 0 0 1 2 0 0 0 2 1
##   5 0 0 1 0 0 2 0 2 0 1
##   6 1 1 0 0 1 0 0 0 0 0
##   7 2 4 1 2 1 0 7 2 1 0
##   8 0 0 1 0 1 1 0 0 2 0
##   9 0 0 0 1 0 0 0 0 0 1
```

```
(Accuracy <- sum(diag(CM)) / n * 100)
```

```
## [1] 21.42857
```

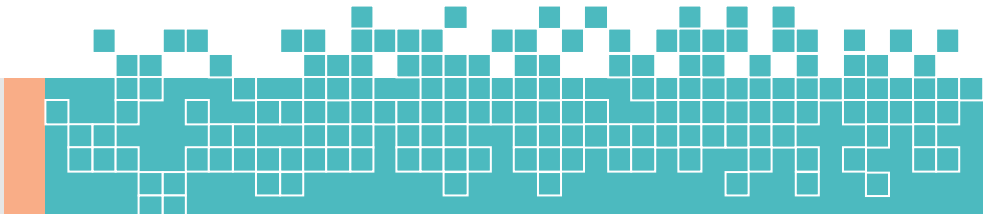
```
table(l)
```

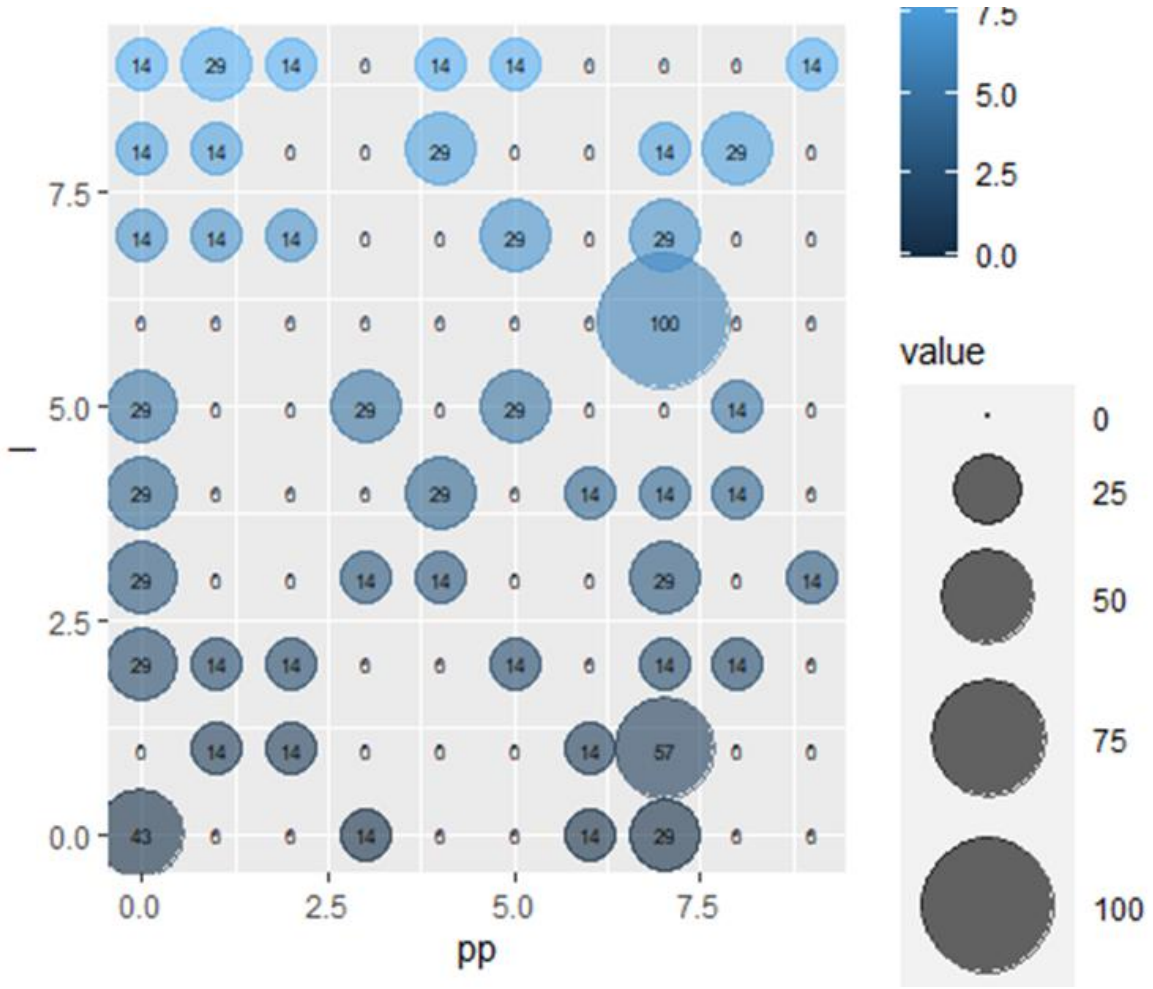
```
## l
## 0 1 2 3 4 5 6 7 8 9
## 7 7 7 7 7 7 7 7 7 7
```

```
CM.percent <- scale(CM, center = F, scale = colSums(CM)) * 100
round(CM.percent)
```

```
##    l
## pp    0   1   2   3   4   5   6   7   8   9
##   0  43   0  29  29  29  29   0  14  14  14
##   1   0  14  14   0   0   0   0  14  14  29
##   2   0  14  14   0   0   0   0  14   0  14
##   3  14   0   0  14   0  29   0   0   0   0
##   4   0   0   0  14  29   0   0   0  29  14
##   5   0   0  14   0   0  29   0  29   0  14
##   6  14  14   0   0  14   0   0   0   0   0
##   7  29  57  14  29  14   0 100  29  14   0
##   8   0   0  14   0  14  14   0   0  29   0
##   9   0   0   0  14   0   0   0   0   0  14
```

```
CM.long <- reshape2::melt(CM.percent)
CM.long$Labels.test <- factor(CM.long$l)
CM.long$Predict.test <- factor(CM.long$pp)

library(ggplot2)


ggplot(CM.long, aes(x = pp, y = l, label = round(value))) +
  geom_point(aes(color = l, size = value), alpha = 0.6) +
  geom_text(size = 2) + scale_size(range = c(0.2, 18.5))
```



دقت مدل برابر ۲۱ درصد با دست نوشته های واقعی است.

بررسی داده ها با الگوریتم بردارماشین های پشتیبان (svm) با نرم افزار R

ماشین های بردار پشتیبان یک ابزار عالی برای طبقه بندی و کشفیات تازه و پیش بینی بر اساس رگرسیون هستند برای استفاده از این ابزار به دو پکیج زیر نیاز داریم :

`Kernlab`

برای اجرای svm.

`Caret`

برای سامان دهی داده ها و خروجی ماتریس در هم ریختگی.

از پکیج kernlab از دستور ksvm استفاده می کنیم :

```
formula: response ~ predictors
```

```
data:
```

داده های آموزشی.

```
scaled = FALSE
```

بردار منطقی که داده ها را دریک مقیا س قرار میدهد یعنی استاندارد سازی و مرکزی سازی داده ها را انجام می دهد.

```
kernel = "vanilladot"& "rbfdot"
```

تابع هسته مورد استفاده در آموزش و پیش بینی است. این پارامتر را می توان روی هر تابعی از هسته کلاس تنظیم کرد که ضرب داخلی را در فضای ویژگی بین دو آرگومان برداری محاسبه می کند (به help هسته ها مراجعه کنید).

kernlab محبوب ترین توابع هسته را ارائه می دهد که می توان با تنظیم پارامتر هسته روی رشته های مختلف به نتیجه رسید ما در اینجا هسته خطی را که دیفالت بسته هم می باشد به علاوه هسته پایه شعاعی گوسین را مورد استفاده قرار دادیم.

```
C = 1
```

پارامتر تنظیم کننده را برای مدل اول برابر ۱ در نظر گرفتیم و در ادامه سعی بر انتخاب کردن پارامتر تنظیم کننده مناسب با حداکثر دقت را ، داریم.

```
########################## Loading libraries##########################

#library(kernlab)
#library(caret)
#library(caTools)

Data=load("HodaDigits.RData")
df <- data.frame(Labels = Labels, Pictures = Pictures)

Labels <- factor(Labels)
Labels.test <- factor(Labels.test)
#----------------------- Linear Kernel -------------------------#

## Linear kernel using default parameters##

#model1_linear <- ksvm(Labels ~ ., data = Pictures, scaled = FALSE, ker
nel = "vanilladot", C = 1)

load("model1_linear.RData")

#eval1_linear <- predict(model1_linear, newdata = Pictures.test, type =
"response")
#confusionMatrix(eval1_linear, Labels.test)
```

```
Confusion Matrix and Statistics


          Reference
Prediction    0    1    2    3    4    5    6    7    8    9
         0 1976    2    1   10   11   21    6    7    3    4
         1    1 1979   35    0   14    6   21   10    5   39
         2    0    6 1850  152   56    1   40   18    2   26
         3    0    0   66 1742  110    3   12    1    4    3
         4    6    1   24   88 1778   18   16    7   14   13
         5   14    0    0    0    7 1936    4    5    9    2
         6    1    1   14    8    8    3 1818   11    9   68
         7    2    5    2    0    1    3   12 1939    7    0
         8    0    0    0    0    2    8    3    2 1918   19
```

```
         9    0    6    8    0   13    1   68    0   29 1826

Overall Statistics

               Accuracy : 0.9381
                 95% CI : (0.9347, 0.9414)
    No Information Rate : 0.1
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9312

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2 Class: 3
Sensitivity            0.9880  0.98950   0.9250  0.87100
Specificity            0.9964  0.99272   0.9833  0.98894
Pos Pred Value         0.9682  0.93791   0.8601  0.89748
Neg Pred Value         0.9987  0.99883   0.9916  0.98571
Prevalence             0.1000  0.10000   0.1000  0.10000
Detection Rate         0.0988  0.09895   0.0925  0.08710
Detection Prevalence   0.1021  0.10550   0.1076  0.09705
Balanced Accuracy      0.9922  0.99111   0.9541  0.92997
                     Class: 4 Class: 5 Class: 6 Class: 7
```

```
Sensitivity            0.88900  0.96800  0.90900  0.96950
Specificity            0.98961  0.99772  0.99317  0.99822
Pos Pred Value         0.90483  0.97926  0.93663  0.98376
Neg Pred Value         0.98769  0.99645  0.98992  0.99662
Prevalence             0.10000  0.10000  0.10000  0.10000
Detection Rate         0.08890  0.09680  0.09090  0.09695
Detection Prevalence   0.09825  0.09885  0.09705  0.09855
Balanced Accuracy      0.93931  0.98286  0.95108  0.98386
                     Class: 8 Class: 9
Sensitivity            0.9590  0.91300
Specificity            0.9981  0.99306
Pos Pred Value         0.9826  0.93593
Neg Pred Value         0.9955  0.99036
Prevalence             0.1000  0.10000
Detection Rate         0.0959  0.09130
Detection Prevalence   0.0976  0.09755
Balanced Accuracy      0.9786  0.95303
```

```
## Linear kernel using stricter C ##
#model2_linear <- ksvm(Labels ~ ., data = Pictures, scaled = FALSE, ker
nel = "vanilladot", C = 10)
#print(model2_linear)
load("model2_linear.RData")
#eval2_linear <- predict(model2_linear, newdata = Pictures.test, type =
"response")
#confusionMatrix(eval2_linear, Labels.test)
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2    3    4    5    6    7    8    9
         0 1976    2    1   10   12   21    6    7    3    4
         1    1 1978   43    1   13    6   20   10    5   41
         2    0    6 1817  164   55    1   52   18    2   24
         3    0    0   87 1712  137    3   11    0    3    2
         4    6    1   27  105 1752   18   17    8   14   12
         5   14    0    0    0    7 1936    4    5    9    2
         6    1    1   13    7    9    3 1812   11    8   72
         7    2    5    3    0    1    3   13 1939    7    0
         8    0    0    0    0    2    8    3    2 1920   19
         9    0    7    9    1   12    1   62    0   29 1824

Overall Statistics

               Accuracy : 0.9333
                 95% CI : (0.9298, 0.9367)
    No Information Rate : 0.1
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9259

 Mcnemar's Test P-Value : NA
```

Statistics by Class:

	Class: 0	Class: 1	Class: 2	Class: 3
Sensitivity	0.9880	0.9890	0.90850	0.85600
Specificity	0.9963	0.9922	0.98211	0.98650
Pos Pred Value	0.9677	0.9339	0.84946	0.87570
Neg Pred Value	0.9987	0.9988	0.98975	0.98404
Prevalence	0.1000	0.1000	0.10000	0.10000
Detection Rate	0.0988	0.0989	0.09085	0.08560
Detection Prevalence	0.1021	0.1059	0.10695	0.09775
Balanced Accuracy	0.9922	0.9906	0.94531	0.92125

	Class: 4	Class: 5	Class: 6	Class: 7
Sensitivity	0.8760	0.96800	0.90600	0.96950
Specificity	0.9884	0.99772	0.99306	0.99811
Pos Pred Value	0.8939	0.97926	0.93547	0.98277
Neg Pred Value	0.9863	0.99645	0.98959	0.99662
Prevalence	0.1000	0.10000	0.10000	0.10000
Detection Rate	0.0876	0.09680	0.09060	0.09695
Detection Prevalence	0.0980	0.09885	0.09685	0.09865
Balanced Accuracy	0.9322	0.98286	0.94953	0.98381

	Class: 8	Class: 9
Sensitivity	0.9600	0.91200
Specificity	0.9981	0.99328
Pos Pred Value	0.9826	0.93779

```
Neg Pred Value         0.9956  0.99025
Prevalence             0.1000  0.10000
Detection Rate         0.0960  0.09120
Detection Prevalence   0.0977  0.09725
Balanced Accuracy      0.9791  0.95264
```

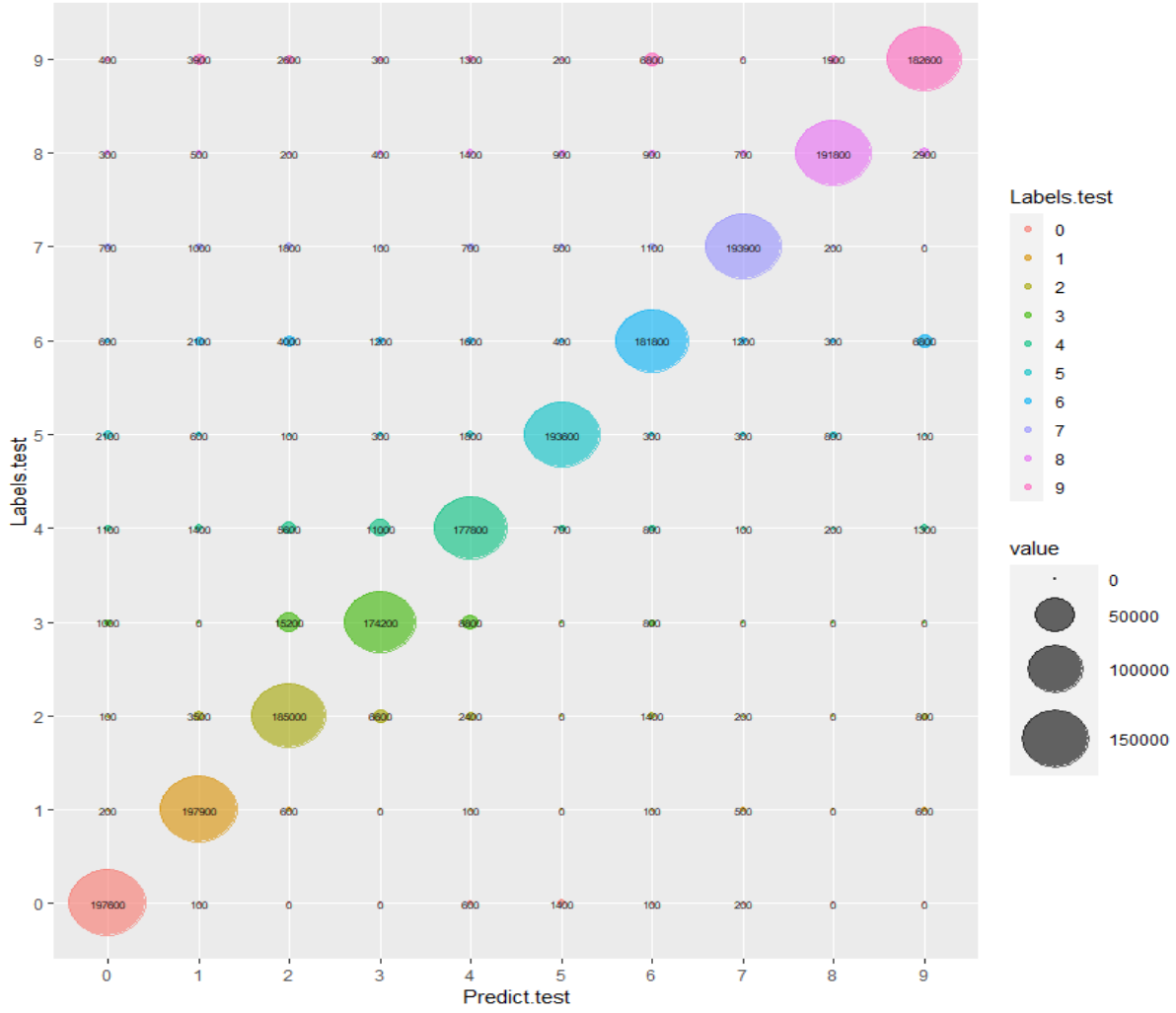
برای یافتن بهترین پارامتر C با روش اعتبار سنجی متقابل و مدل اصلی با دقت نهایی از این رشته کد استفاده می کنیم .

```
## Using cross validation to optimise C ##

#grid_linear <- expand.grid(C= c(0.001, 0.1 ,1 ,10 ,100)) # defining ra
nge of C

#fit.linear <- train(label ~ ., data = Pictures, metric = "Accuracy", m
ethod = "svmLinear",
 #                   tuneGrid = grid_linear, preProcess = NULL,
 #                   trControl = trainControl(method = "cv", number = 5
))
```

نمودار تشخیص تصاویر و قدرت پیشبینی مدل خطی شماره ۱:

```
################# Ploting modele1_linear ##############

#a=confusionMatrix(eval1_linear, Labels.test)
#(Accuracy <- sum(diag(a$table)) / nrow(Pictures.test) * 100)

93.81
#table(Labels.test)
Labels.test

   0    1    2    3    4    5    6    7    8    9

2000 2000 2000 2000 2000 2000 2000 2000 2000 2000


#CM.percent <- a$table * 100
#CM.long <- reshape2::melt(CM.percent)
#CM.long$Labels.test <- factor(CM.long$Reference)
#CM.long$Predict.test <- factor(CM.long$Prediction )

#library(ggplot2)
```

```
#ggplot(CM.long, aes(x = Predict.test, y = Labels.test, label = round(v
alue))) +
  #geom_point(aes(color = Labels.test, size = value), alpha = 0.6) +
  #geom_text(size = 2) + scale_size(range = c(0.2, 18.5))
```



مدل پیش فرض با هسته rbfdot:

```
#---------------------- Radial Kernel -------------------------#

## Radial kernel using default parameters##
#model1_rbf <- ksvm(label ~ ., data = Pictures, scaled = FALSE, kernel
= "rbfdot", C = 1, kpar = "automatic")
#print(model1_rbf)
load("model1_rbf.RData")
#eval1_rbf <- predict(model1_rbf, newdata = Pictures.test, type = "resp
onse")
#confusionMatrix(eval1_rbf, Labels.test)
```

```
Confusion Matrix and Statistics


          Reference
Prediction    0    1    2    3    4    5    6    7    8    9
         0 1985    4    2    5    2    5    1    2    0    2
         1    1 1988   21    0    3    3    7    8    3   24
         2    0    0 1922   83   18    0    5   15    2    2
         3    0    0   39 1860   58    0    0    0    0    1
         4    1    1    6   44 1909    6    2    5    7    4
         5   10    0    0    2    1 1974    5    1    2    3
         6    1    4    6    2    3    3 1945    5    4   11
         7    2    2    1    2    0    3    3 1964    0    1
         8    0    0    0    1    1    6    1    0 1972    5
         9    0    1    3    1    5    0   31    0   10 1947


Overall Statistics


               Accuracy : 0.9733
```

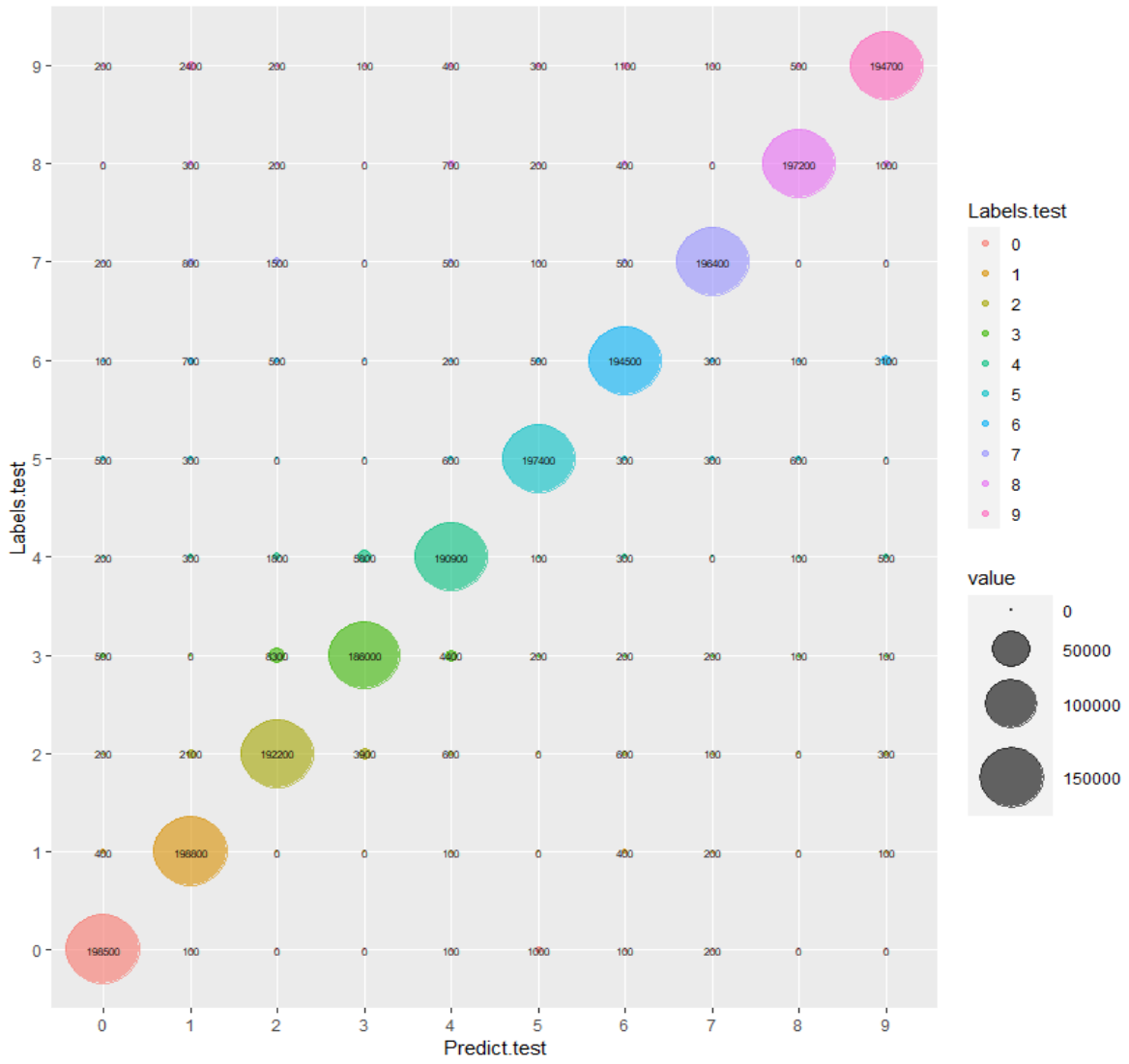
```
                 95% CI : (0.971, 0.9755)
    No Information Rate : 0.1
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9703

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2 Class: 3
Sensitivity           0.99250   0.9940   0.9610   0.9300
Specificity           0.99872   0.9961   0.9931   0.9946
Pos Pred Value        0.98855   0.9660   0.9389   0.9499
Neg Pred Value        0.99917   0.9993   0.9957   0.9922
Prevalence            0.10000   0.1000   0.1000   0.1000
Detection Rate        0.09925   0.0994   0.0961   0.0930
Detection Prevalence  0.10040   0.1029   0.1023   0.0979
Balanced Accuracy     0.99561   0.9951   0.9770   0.9623
                     Class: 4 Class: 5 Class: 6 Class: 7
Sensitivity           0.95450   0.9870  0.97250   0.9820
Specificity           0.99578   0.9987  0.99783   0.9992
Pos Pred Value        0.96171   0.9880  0.98034   0.9929
Neg Pred Value        0.99495   0.9986  0.99695   0.9980
Prevalence            0.10000   0.1000  0.10000   0.1000
```

```
Detection Rate         0.09545   0.0987  0.09725   0.0982
Detection Prevalence   0.09925   0.0999  0.09920   0.0989
Balanced Accuracy      0.97514   0.9928  0.98517   0.9906
                     Class: 8 Class: 9
Sensitivity            0.9860  0.97350
Specificity            0.9992  0.99717
Pos Pred Value         0.9930  0.97447
Neg Pred Value         0.9984  0.99706
Prevalence             0.1000  0.10000
Detection Rate         0.0986  0.09735
Detection Prevalence   0.0993  0.09990
Balanced Accuracy      0.9926  0.98533
```

نمودار تشخیص تصاویر و قدرت پیشبینی مدل شعاعی شماره ۱:

```
############### Ploting modele1_rbf ####################
#c=confusionMatrix(eval1_rbf, Labels.test)
#(Accuracy <- sum(diag(c$table)) / nrow(Pictures.test) * 100)

97.33
#table(Labels.test)
Labels.test

   0    1    2    3    4    5    6    7    8    9

2000 2000 2000 2000 2000 2000 2000 2000 2000 2000
#CM.percent <- c$table * 100


#CM.long <- reshape2::melt(CM.percent)
#CM.long$Labels.test <- factor(CM.long$Reference)
#CM.long$Predict.test <- factor(CM.long$Prediction )

#library(ggplot2)

#ggplot(CM.long, aes(x = Predict.test, y = Labels.test, label = round(v
alue))) +
```

```
#geom_point(aes(color = Labels.test, size = value), alpha = 0.6) +
#geom_text(size = 2) + scale_size(range = c(0.2, 18.5))
```



نتیجه گیری

در این پژوهش بررسی مجموعه ارقام دستنویس هدی که اولین مجموعه‌ی بزرگ ارقام دستنویس فارسی است پرداختیم که نتایج حاصل با مدل رگرسیون چند جمله ای به دقت پیشبینی نهایی ۸۸,۹۱ منجر شد. همچنین با الگوریتم ماشین های بردار پیشتیبان با دوهسته خطی و شعاعی وآرگومان های پیش فرض به ترتیب به دقت های پیشبینی ۹۳,۸۱ و ۹۷,۳۳ رسیدیم و در کل مدل دارای دقت بالا برای پیش بینی نویسه های ارقام فارسی است و بهترین مدل با دقت پیشبینی بیشتر را می توان مدل با هسته شعاعی از مجموعه بردار ماشین های پشتیبان دانست.

پیشنهادات

برای تحقیقات آتی با مدل های مشابه پیشنهاد می شود که :

۱. به بررسی حروف فارسی و اندازه گیری دقت پیشبینی برای حروف پرداخته شود و در ادامه از این نتایج برای ساخت اپلیکیشن های متن خوان و تبدیلگر فارسی به قالب های مختلف نوشتاری با تصویر اشاره کرد.

۲. از این مجموعه داده و دقت پردازش تصویر می توان برای ساخت اپیکیشن گویا برای افراد نابینا وکم بینا برای انجام امورمالی و اداری روز مره با تشخیص اعداد و سپس بیان خودکار اعداد استفاده نمود.

۳. برای پایش فرم های سنجش سلامت که در قالب اعداد ثبت می شوند نیز قابل استفاده خواهد بود و میتوان میزان بهبود و یا وخامت وضع بیمار را از راه دور تشخیص داد. همچنین می توان از تصاویراسکن های مختلف هم به این موضوع پی برد.

پایان