
# گزارش تمرین عملی سوم درس هوش مصنوعی

آریا جلالی
۹۸۱۰۵۶۶۵

# فهرست مطالب

# ۱ کتابخانه‌های استفاده شده:

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import pandas as pd
import os
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.datasets import make_moons,make_circles
from keras.datasets import mnist
import pandas as pd
from sklearn.multiclass import OneVsRestClassifier
import seaborn as sns
```

از کتابخانه‌ی numpy برای کار کردن با داده‌های متفاوت و دستکاری عکس‌ها استفاده شده است. در ادامه از کتابخانه‌ی matplotlib و PIL برای نمایش نمودار داده‌ها و نمایش عکس‌ها استفاده شده است.از کتابخانه‌ی pandas برای تبدیل آرایه و ماتریس‌های numpy به فایل csv استفاده شده است که کار با داده‌ها را راحت‌تر میکند. از os برای بازکردن فایل‌ها برای بخش نهایی استفاده شده است، و در نهایت ابزارهای لازم برای SVM از کتابخانه‌ی import sklearn شده است.

## ۱.۱ نصب کتابخانه‌های موردنیاز:

```
pip install numpy
pip install tensorflow
pip install -U scikit-learn
pip install matplotlib
python3 -m pip install --upgrade pip
python3 -m pip install --upgrade Pillow
pip install pandas
pip install seaborn
```

# ۲ تابع‌های کمکی استفاده شده:

```
def generate_dataset(min_value, max_value, size,
    positive_condition):
    data = pd.DataFrame(
        np.concatenate((
            np.random.uniform(min_value, max_value, (size, 2)),
            -np.ones((size, 1))), axis=1),
```

```
        columns=['x', 'y', 'target'])
    data.target[positive_condition(data.x, data.y)] = 1
    return data
```

از این تابع برای ساخت داده‌های 2 بخشی استفاده میکنیم به این صورت یک تابع یک بازه در فضای 2 بعدی میگیرد و به اندازه‌ی size نقطه ایجاد میکند و اگر مختصات این نقاط در تابع positive_condition صدق کند مقدار آن 1 و در غیر این صورت برابر با 1- خواهد بود. در نهایت این نقاط با مقدار متناظر خودشان با کمک کتابخانه‌ی pandas در یک فایل csv نوشته میشوند.

```
def plot_dataset():
  plt.figure(figsize=(10, 10))
  #Basic Shape
  plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=y, s=25, cmap=plt.cm
    .Paired)
```

از تابع بالا برای کشیدن داده‌های بدست آمده استفاده میکنیم.

```
def plot_support_vectors():
  plt.figure(figsize=(10, 10))
  plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=y, s=25, cmap=plt.cm
    .Paired)
  plt.scatter(model.support_vectors_[:,0],model.support_vectors_
    [:,1], s = 25, c = 'yellow')
```

با استفاده از تابع بالا نیز نقاطی که support vector هستند را زرد میکنیم برای دید بهتر.

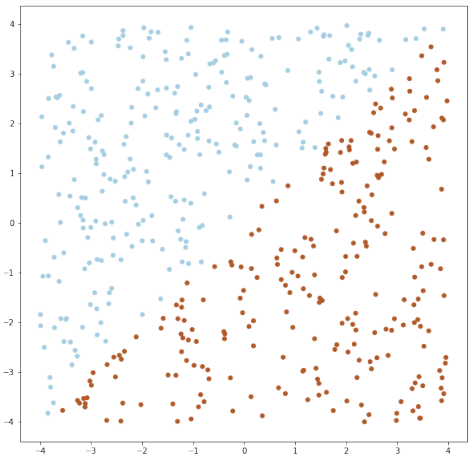
```
def plot_svm_result():
  plt.figure(figsize=(10, 10))
  ax = plt.gca()
  plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=y, s=25, cmap=plt.cm
    .Paired)
  xlim = ax.get_xlim()
  ylim = ax.get_ylim()
  xx = np.linspace(xlim[0], xlim[1], 30)
  yy = np.linspace(ylim[0], ylim[1], 30)
  YY, XX = np.meshgrid(yy, xx)
  xy = np.vstack([XX.ravel(), YY.ravel()]).T
  Z = model.decision_function(xy).reshape(XX.shape)
  ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha
    =0.5,
             linestyles=['--', '-', '--'])

  ax.scatter(model.support_vectors_[:, 0], model.
    support_vectors_[:, 1], s=25,
             linewidth=1, facecolors='none', edgecolors='k')
  plt.show()
```

از تابع بالا نیز برای رسم کردن خط جدا کننده SVM و خط margin استفاده میکنیم.

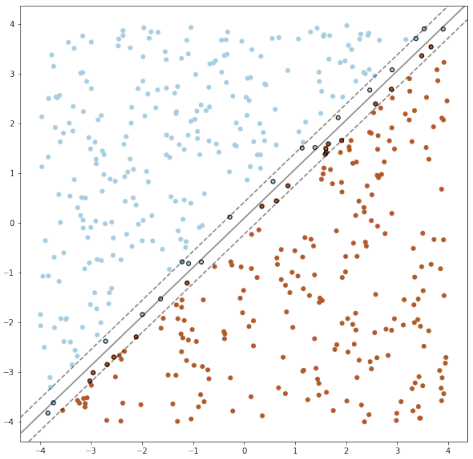
# ۳ نتایج بخش اول:

## ۱.۳ $x > y$

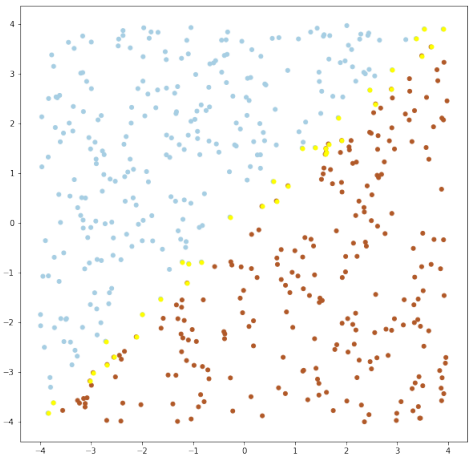
در این بخش با استفاده از linear kernel اقدام به دسته‌بندی نقاط میکنیم.



شکل ۱: نقاط مورد بررسی



شکل ۲: خط جدا کننده و خطوط margin

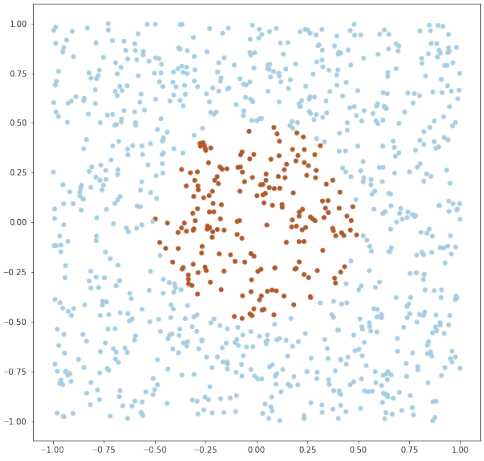


شکل ۳: بردارهای پشتیبان

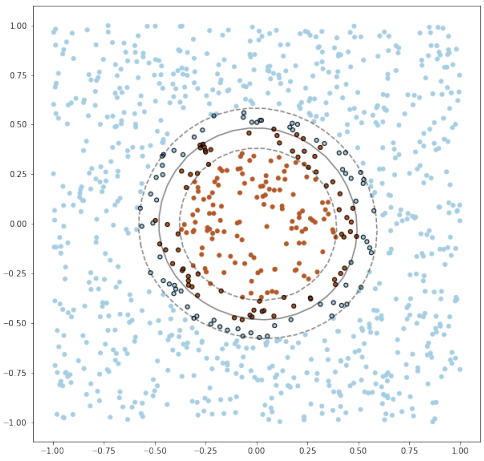
همانطور که مشاهده میکنید چون داده‌ها به صورت خطی جداپذیر هستند کرنل linear آن‌ها را به صورت کامل classify کرده است، و نیازی به استفاده از کرنل‌های پیچیده‌تر و تصویر کردن نقاط به ابعاد بالاتر نیست.

## ۲.۳ $x^2 + y^2 < \frac{1}{4}$

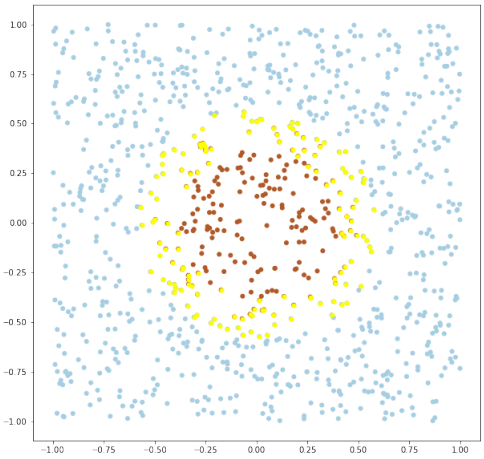
در این بخش چون نقاط به صورت خطی جداپذیر نیستند، داده‌ها را با استفاده از کرنل rbf که در ادامه به نحوه‌ی عملکرد آن خواهیم پرداخت استفاده خواهیم کرد.



شکل ۴: نقاط مورد بررسی




شکل ۵: خط جدا کننده و خطوط margin

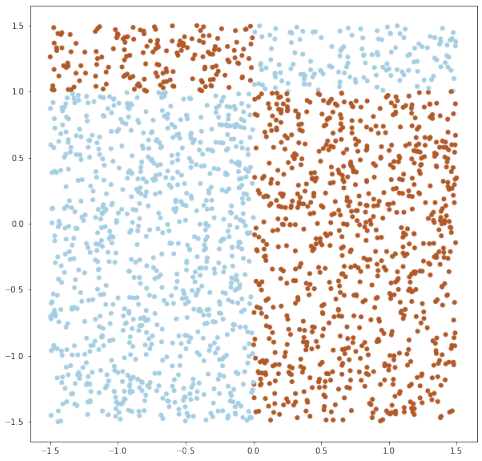



شکل ۶: بردارهای پشتیبان

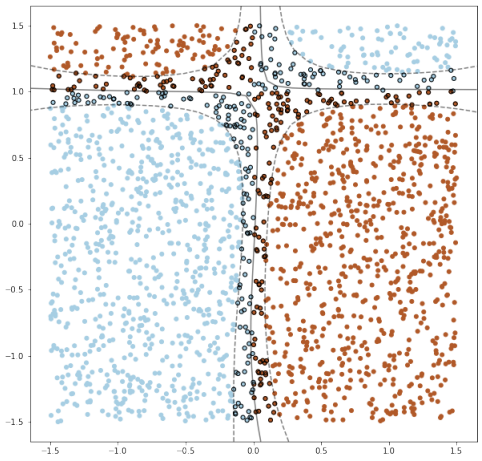
در اینجا کرنل rbf نیز کار نسبتا خوبی در جدا کردن نقاط انجام داده است و لازم به ذکر است که به صورت default پارامتر C ماشین بردار پشتیبان برابر با 1 است.

### ۳.۳ $x > yx$

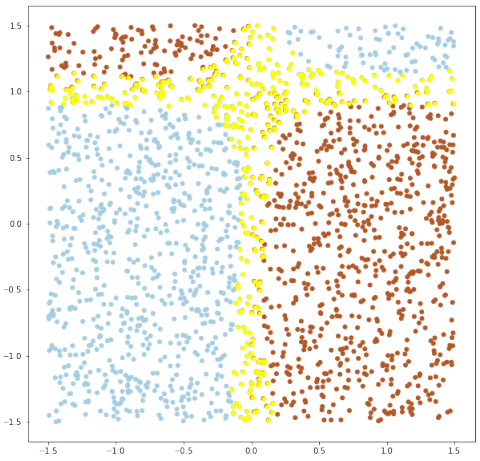
در این بخش نیز چون داده‌ها به صورت خطی جداپذیر نیستند از کرنل rbf استفاده کردیم.




شکل ۷: نقاط مورد بررسی




شکل ۸: خط جدا کننده و خطوط margin



شکل ۹: بردارهای پشتیبان

همانطور که قابل مشاهده است کرنل rbf هنوز چون نقاط زیاد شکل پیچیده‌ای ندارند میتواند به خوبی با C = 1 دسته‌بندی خوبی انجام دهد.

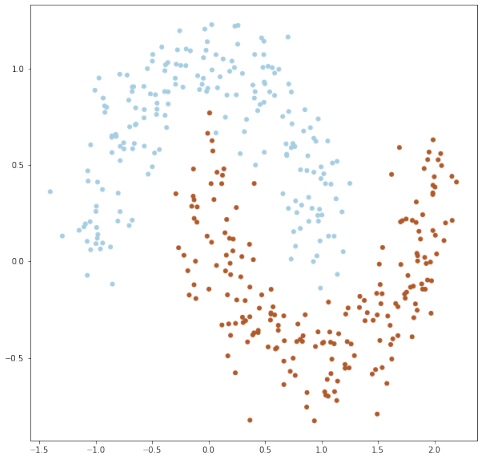
## ۴.۳ مقایسه‌ی کرنل **rbf** با **poly**

در این بخش به مقایسه‌ی 2 کرنل rbf و polynomial میپردازیم. فرمول کرنل poly برابر با ضابطه‌ی مقابل است:

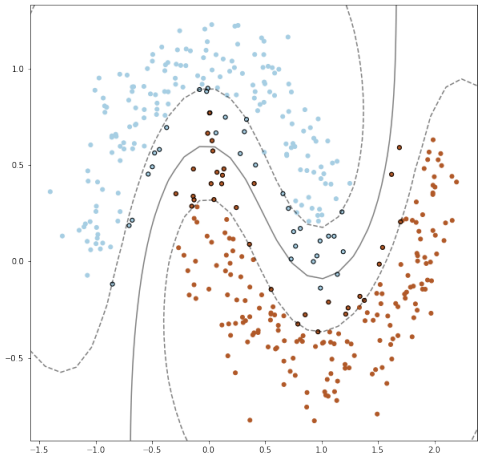
$$K(x, y) = (x^T y + c)^d \quad (۱)$$

d درجه‌ی کرنل را مشخص می‌کند و به طور معمول هرچه بالاتر باشد احتمال overfit شدن بیشتر است. c نیز متغیر نامنفی مستقل کرنل است که تاثیر نقاط با مقادیر بزرگ و مقادیر کوچک را افزایش میدهد تا بعضی از داده‌ها تاثیر خودشان را از دست ندهند. این متغیر برای d های کوچک میتواند برابر با 0 باقی بماند زیرا اعداد بزرگ با d کوچک زیاد بزرگ و اعداد کوچک (کوچک‌تر از 1 ) زیاد کوچک نمیشوند.
شکل‌های این بخش توسط تابع make_moons کتابخانه‌ی sklearn تولید شده‌اند.

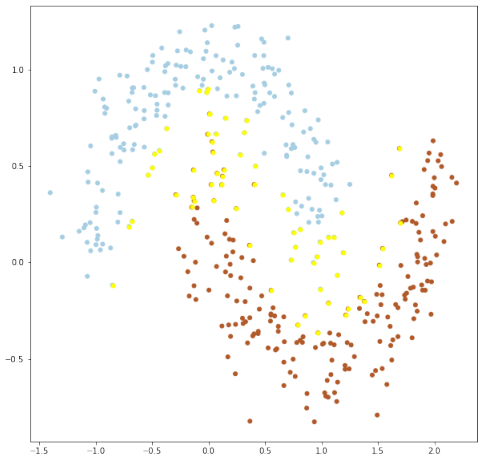
### ۱.۴.۳ کرنل **rbf**:



شکل ۱۰: نقاط مورد بررسی

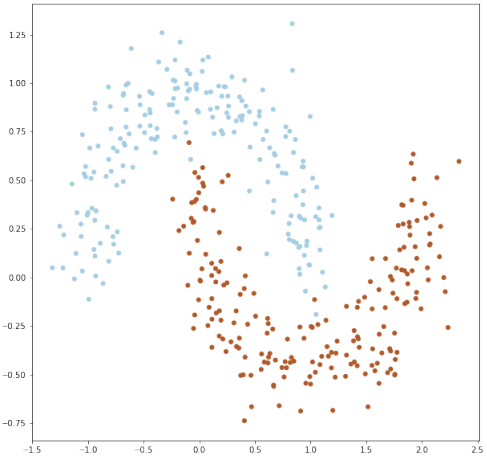


شکل ۱۱: خط جدا کننده و خطوط margin

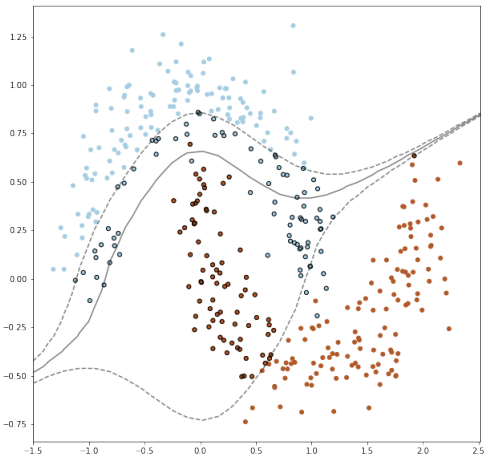


شکل ۱۲: بردارهای پشتیبان

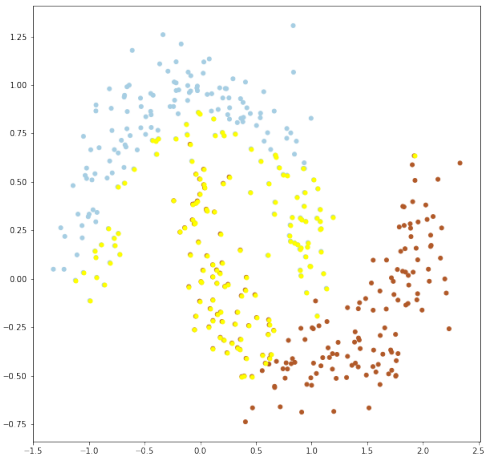
### ۲.۴.۳ کرنل **poly**:



شکل ۱۳: نقاط مورد بررسی



شکل ۱۴: خط جدا کننده و خطوط margin



شکل ۱۵: بردارهای پشتیبان

همانطور که از تصاویر بالا مشخص است خط جداکننده‌ی تولید شده توسط کرنل rbf بسیار بهتر از خط poly است و دلیل این امر این است که کرنل rbf به مراتب پیچیده‌تر است و به صورت کلی کرنل poly برای Natural Language processing استفاده میشود و برای binary classification بهتر است از کرنل‌های دیگر استفاده کنیم به خصوص که در حوزه‌ی زمانی نیز کرنل poly با کرنل rbf یک زمان میگیرند.

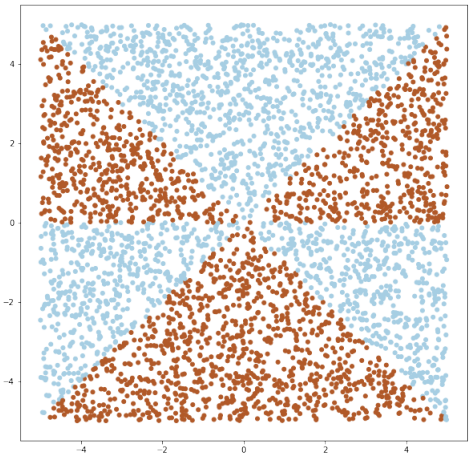
### ۵.۳ cross validation:

در این بخش نقاط موجود در نامعادله‌ی $y * (|x| - |y|) > 0$ را دسته‌بندی میکنیم و چون این نامعادله به مراتب پیچیده‌تر از نامعادلات قبلی است hyperparameter ها را با استفاده از GridSearchCV و استفاده از تکنیک cross validation بدست می‌آوریم.

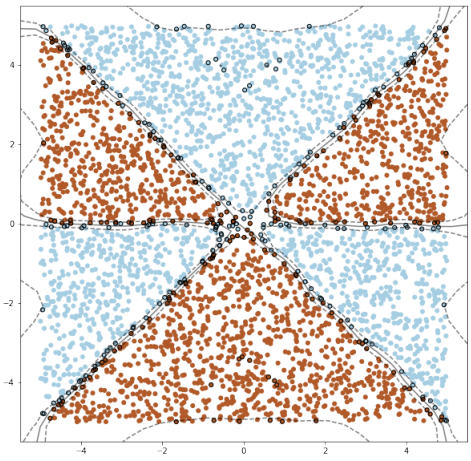
### ۱.۵.۳ کد استفاده شده:

```
data = generate_dataset(-5,5,4000,lambda x,y :y * (np.abs(x) -
    np.abs(y)) > 0)

X=data.iloc[:,0:2]
y=data['target']

folds = KFold(n_splits = 5, shuffle = True)

hyper_params = [{'gamma': [1, 1e-1, 1e-2, 1e-3, 1e-4], 'C': [1,
    5, 10]}]

# specify model
model = SVC(kernel="rbf")

model = GridSearchCV(estimator = model,
                         param_grid = hyper_params,
                         scoring= 'accuracy',
                         cv = folds,
                         verbose = 2,
                         n_jobs = 8,
                         return_train_score=True)
model.fit(data[['x', 'y']], data.target)
plot_dataset()
```

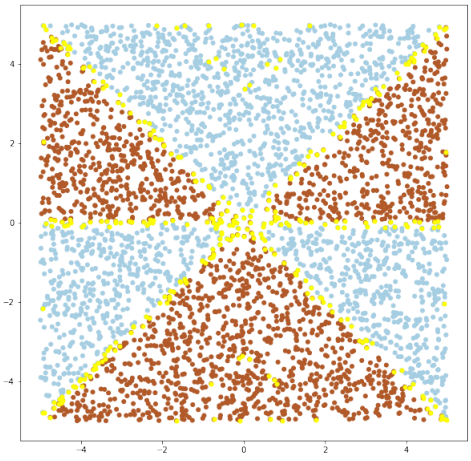
با استفاده از کد بالا پارامترهای 10 , 5 , 1 برای C و 1, 0.1, 0.01, 0.001, 0.0001 برای پارامتر گاما تست میشوند که برای هر پارامتر تعداد داده‌های train به 5 بخش تقسیم میشوند و تکنیک cross validation روی آن‌ها انجام میشود.
در نهایت با گفته‌های بالا نیاز به 75 بار فیت کردن داریم.



شکل ۱۶: نقاط مورد بررسی



شکل ۱۷: خط جدا کننده و خطوط margin



شکل ۱۸: بردارهای پشتیبان

با استفاده از تکنیک cross validation به دقت 98.6 درصدی با پارامترهای C = 10 و $\gamma$ = 1 رسیدیم.

## ۶.۳ کرنل **rbf**:

ضابطه‌ی کرنل rbf یا Radial basis function به صورت مقابل است:

$$K(x,y) = exp(-\frac{\|(x-y)\|^2}{2\sigma^2}) \quad \gamma = \frac{1}{2\sigma^2} \qquad (۲)$$

پارامتر گاما در واقع ناحیه اثرگذاری بردارهای پشتیبان را مشخص میکند و هرچه زیادتر باشد تاثیر بردارهای پشتیبان بیشتر میشود و احتمال overfit بیشتر شده و هرچه کمتر باشد ماشین بردار پشتیبان شکل کلی نقاط را درک نمی‌کند و عملکرد خوبی نخواهد داشت.

### ۷.۳ نتیجه‌گیری:

همانطور که از نتایج بالا مشخص است بهتر است برای داده‌های پیچیده که به صورت خطی جداپذیر هستند از کرنل rbf و برای داده‌هایی که به صورت خطی جداپذیر هستند از کرنل linear استفاده کنیم و به صورت کلی کرنل polynomial در classification زیاد کاربردی نیست.

## ۴ digit classification:

در این بخش به mutli class classification توسط SVM روی داده‌های mnist می‌پردازیم.

### ۱.۴ کد استفاده شده:

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()

X = np.concatenate((X_train, X_test)).reshape(70000, 28 * 28).
    astype('float32') / 255

y = np.concatenate((y_train, y_test)).reshape(70000)


x_train, x_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2)

seq = np.random.randint(0, len(x_train), len(x_train) // 10)
x_train,y_train = x_train[seq],y_train[seq]
seq = np.random.randint(0, len(x_test), len(x_test) // 10)
x_test,y_test = x_test[seq],y_test[seq]


plt.figure(figsize=(15, 7))
df = pd.DataFrame(np.sort(y_train), columns = ['digit'])



df['digit'].value_counts().sort_index().plot.bar()

folds = KFold(n_splits = 5, shuffle = True)

hyper_params = [{'gamma': [1, 1e-1, 1e-2, 1e-3, 1e-4], 'C': [1,
    5, 10]}]
```

```
26
27 # specify model
28 model = SVC(kernel="linear")
29
30 model = GridSearchCV(estimator = model,
31                         param_grid = hyper_params,
32                         scoring= 'accuracy',
33                         cv = folds,
34                         verbose = 2,
35                         n_jobs = 8,
36                         return_train_score=True)
37 model.fit(x_train, y_train)
```

همانطور که در داک تمرین گفته شده است ابتدا عکس‌های داده شده را که به صورت یک ماتریس است به صورت یک بردار در می‌آوریم و مقادیر آن را با تقسیم کردن بر 255 نرمال میکنیم و در ادامه همانند بخش قبل با استفاده از GridSearchCv و تکنیک cross validation پارامترهای مناسب برای کرنل‌های متفاوت را بدست می‌آوریم.

### ۲.۴ کرنل **linear**:

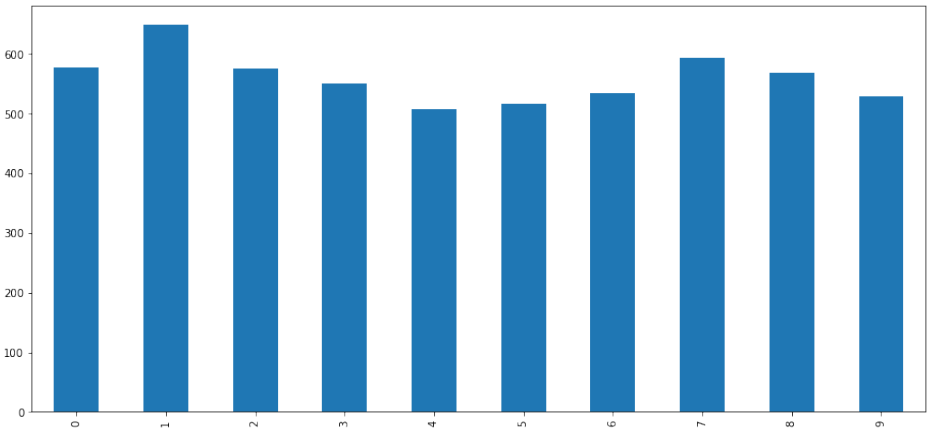
با استفاده از تکنیک cross validation به پارامترهای $C = 1$ و $\gamma = 1$ رسیدیم و به دقت 91.9 درصدی در داده‌های تست و به دقت 90.8 درصدی در داده‌های ترین رسیدیم.

### ۳.۴ کرنل **poly with degree 5**:

با استفاده از تکنیک cross validation به پارامترهای $C = 1$ و $\gamma = 1$ رسیدیم و به دقت 90.2 درصدی در داده‌های تست و به دقت 90 درصدی در داده‌های ترین رسیدیم

### ۴.۴ کرنل **rbf**:

با استفاده از تکنیک cross validation به پارامترهای $C = 10$ و $\gamma = 0.01$ رسیدیم و به دقت 96.1 درصدی در داده‌های تست و به دقت 96.6 درصدی در داده‌های ترین رسیدیم
همانطور که انتظار داشتیم کرنل rbf بهترین عملکرد و کرنل polynomial بدترین عملکرد را داشت.
** لازم به ذکر است به دلیل فراوان بودن تعداد عکس‌ها و طولانی بودن فرایند یادگیری SVM مجبور شدم تعداد داده‌ها را تقسیم بر 10 کنم و در صورتی که از تمام داده‌ها استفاده کنیم مطمئنا به نتایج بهتری میرسیدیم.




شکل ۱۹: فراوانی داده‌های متفاوت

همانطور که از نمودار بالا مشخص است تعداد داده‌های برداشته شده از هر رقم تقریبا برابر است و bias سمت یک رقم خاص نداریم.

## ۵.۴ نتیجه‌گیری:

با بررسی نتایج بالا مشخص میشود در صورت داشتن زمان و قدرت کامپیوتری فراوان دقت حاصل شده از SVM و شبکه‌ی عصبی مناسب تقریبا باهم برابر بوده و برای فعالیت‌های متفاوت بسیار بیشتر از چیزی است که نیاز داریم. اما همانطور که در بالا نیز گفته شد به دلیل اینکه فرایند یادگیری SVM به صورت Hill climbing نیست بسیار زمان‌بر است و این موضوع حتی برای کرنل linear که ساده‌ترین کرنل است نیز برقرار است و در این زمینه، شبکه مصنوعی بسیار بهتر عمل میکند. ولی در بحث پارامتر به وضوح SVM برنده است و هر کرنل حداکثر یک یا دو متغیر مستقل جدید اضافه میکند که با احتساب پارامتر C برای slack بسیار کمتر از پارامترهای موجود در یادگیری شبکه‌ی عصبی است.

## ۶.۴ دسته‌بندی پلاک‌های ماشین:

```
x,y = [],[]
for path in os.listdir(path = '.'):
    if os.path.isdir(path) and len(path) == 1:
        for filename in os.listdir(path):
            f = os.path.join(path, filename)
            # checking if it is a file
            if os.path.isfile(f):
                x.append(np.array(Image.open(f)).reshape(1, 16 * 16))
                y.append(path)

x = np.array(x).astype('float32') / 255
```

```
y = np.array(y)
x = x.reshape(1500, 16 * 16)
x_train, x_test, y_train, y_test = train_test_split(x, y,
    test_size=0.2)

folds = KFold(n_splits = 5, shuffle = True)

hyper_params = [{'gamma': [1, 1e-1, 1e-2, 1e-3, 1e-4], 'C': [1,
    5, 10]}]

# specify model
model = SVC(kernel="linear")

model = GridSearchCV(estimator = model,
                        param_grid = hyper_params,
                        scoring= 'accuracy',
                        cv = folds,
                        verbose = 2,
                        n_jobs = 8,
                        return_train_score=True)


model.fit(x_train, y_train)
```

در این بخش نیز همانند بخش قبل ابتدا داده‌ها را با تقسیم کردن مقدار هر پیکسل پس از تبدیل ماتریس عکس به بردار به 255 نرمالایز میکنیم و سپس با استفاده از SVM این داده‌ها را fit میکنیم.

### ۷.۴ کرنل **linear**:

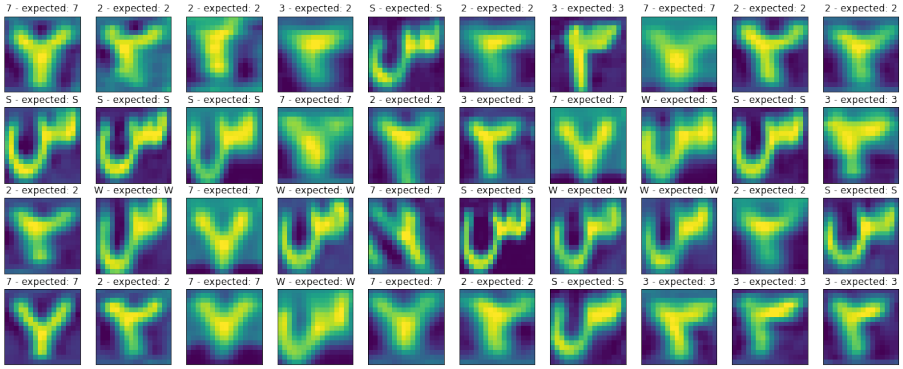
با استفاده از تکنیک cross validation به پارامترهای $C = 1$ و $\gamma = 1$ رسیدیم و به دقت 95.9 درصدی در داده‌های تست و به دقت 97 درصدی در داده‌های ترین رسیدیم.

### ۸.۴ کرنل **poly with degree 5**:

با استفاده از تکنیک cross validation به پارامترهای $C = 10$ و $\gamma = 0.01$ رسیدیم و به دقت 95.4 درصدی در داده‌های تست و به دقت 96.6 درصدی در داده‌های ترین رسیدیم.

### ۹.۴ کرنل **rbf**:

با استفاده از تکنیک cross validation به پارامترهای $C = 10$ و $\gamma = 0.01$ رسیدیم و به دقت 96.5 درصدی در داده‌های تست و به دقت 96.6 درصدی در داده‌های ترین رسیدیم.




شکل ۲۰: تعداد از عکس‌ها و حدس‌های زده شده.

## ۵ Multi classification:

روش‌های متفاوتی برای دسته‌بندی چندگانه با SVM وجود دارد که روش استفاده شده در این کد One-Vs-Rest است که به این صورت پیاده‌سازی میشود که ابتدا یکی از 10 کلاس حاصل را جدا میکند و تشخیص میدهد که آیا داده‌ی موردنظر مربوط به کلاس جدا شده است یا خیر. به عنوان مثال در مرحله‌ی اول میتوانیم تشخیص بدهیم آیا عکس داده شده برابر با 7 است یا خیر. ولی مشکل این روش این است که باید به ازای هر m کلاس یک مسئله‌ی binary classification تولید کنیم و با توجه به نتایج تشخیص دهیم کدام کلاس در این مرحله جدا شود و همین کار را برای m - 1 کلاس باقی انجام دهیم.

## ۶ چالش‌ها:

در این پروژه چالش‌های متفاوتی داشتم که نمونه‌هایی از آن‌ها کارکردن با توابع متفاوت SVM بود که با استفاده از مطالعه documentation حل شد. یکی دیگر از مشکلات مشکل زمان مصرف شده برای یادگیری SVM بود که با محدود کردن تعداد داده‌های ترین و تست مشکل حل شد. البته راه دیگر این بود با استفاده از PCA ابعاد ورودی را کاهش بدیم و سپس fit دهیم. و مشکل نهایی خواندن عکس‌های پلاک‌ها بود که با استفاده از توابع موجود در کتابخانه‌ی os حل شد.