


امیررضا صدیقین


۹۹۳۶۱۴۰۲۴


### سوال ۱ :


کدهای مربوط به این بخش در فولدر problem1 در code ها است.


با استفاده از کد مربوط به فایل text\_to\_csv.py داده‌های لازم را منظم در CSV ها ریختم که در فولدر مربوط به data در problem1 موجود است.


 A\_test.csv


 A\_train.csv


 B\_test.csv


 B\_train.csv


 C\_test.csv


 C\_train.csv


 D\_test.csv


 D\_train.csv


 E\_test.csv


 E\_train.csv


 F\_test.csv


 F\_train.csv


 G\_test.csv

 G\_train.csv

 H\_test.csv

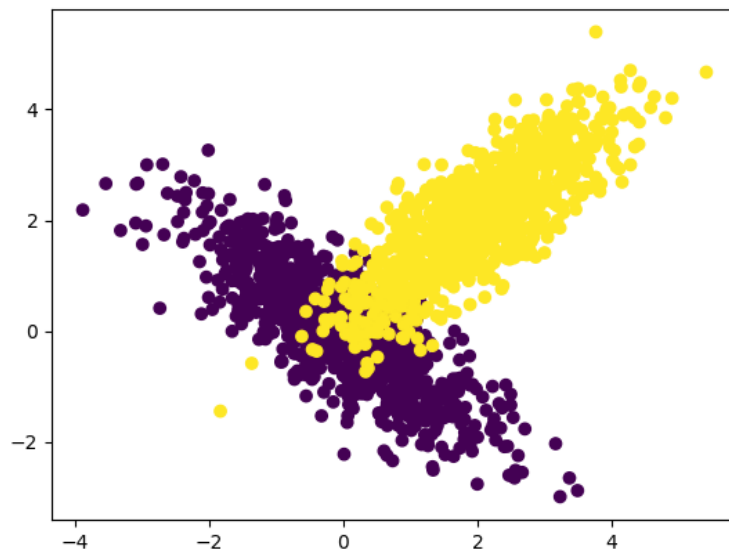
 H\_train.csv

 I\_test.csv

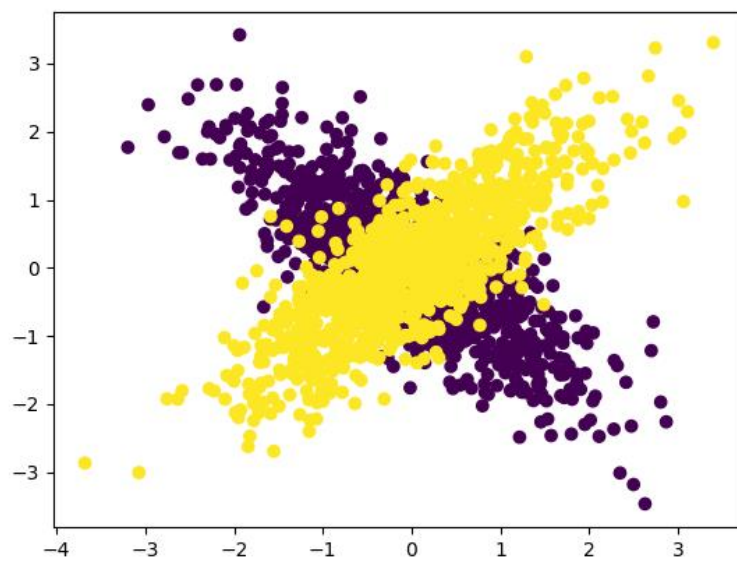
 I\_train.csv

با استفاده از کد مربوط در فایل plot.py داده ها را نمایش داده ام که در result/plots عکس ها موجود است و به شرح زیر است.

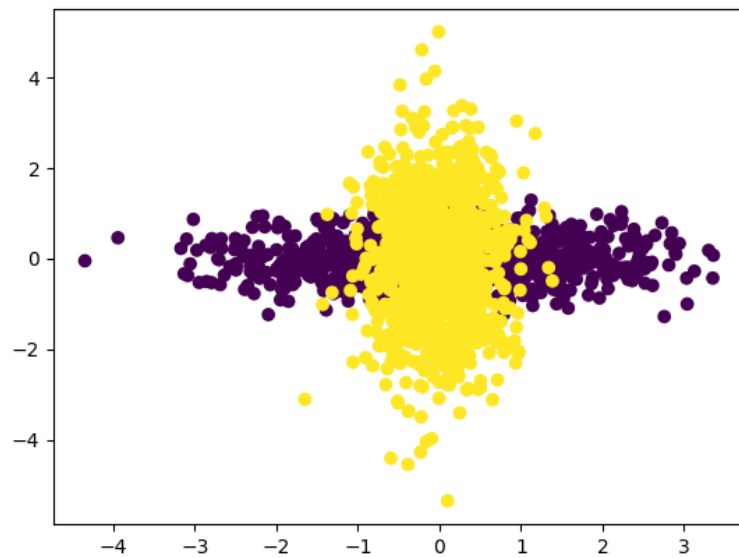
A •



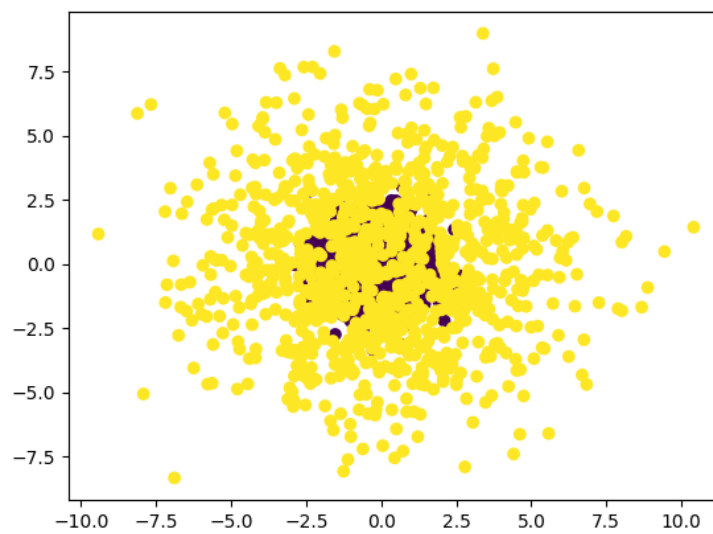
B •



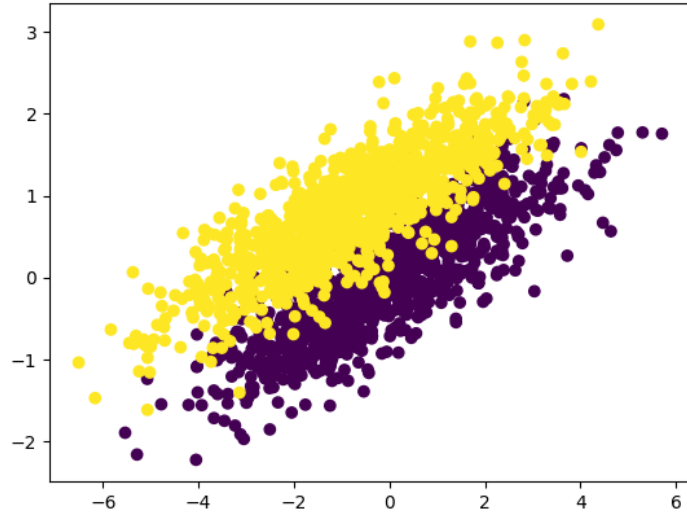
C •



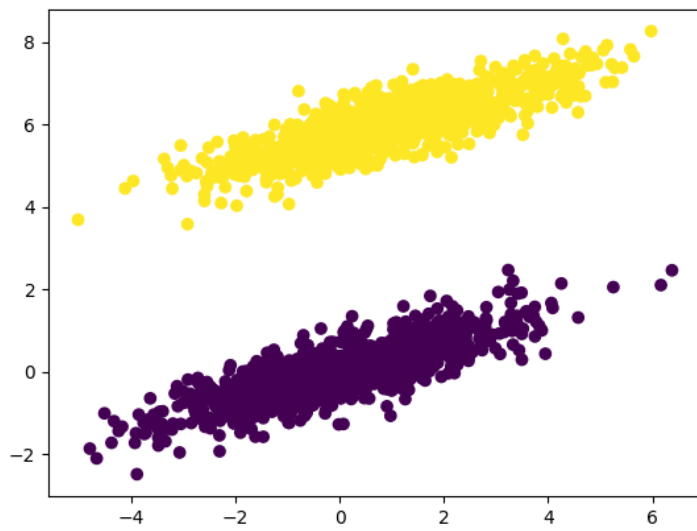
D •



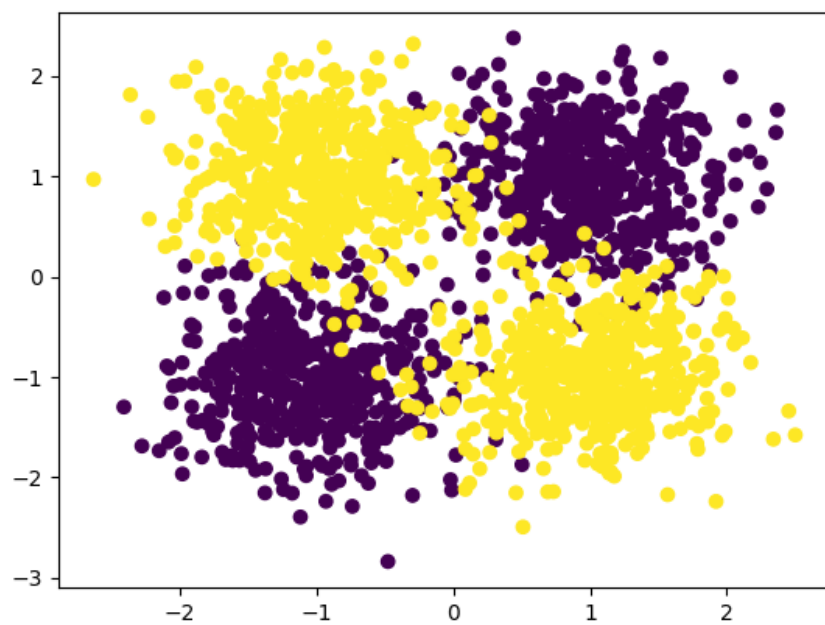
E •



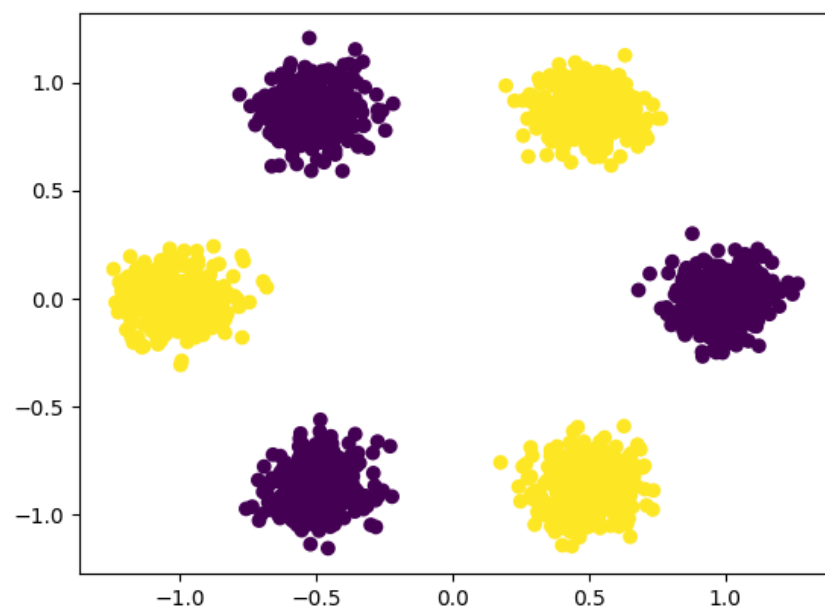
F •



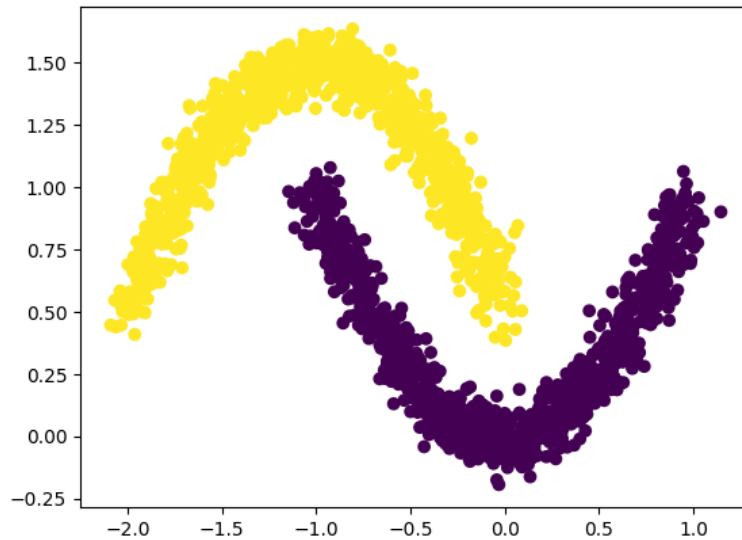
G •



H •



I •



در فایل مربوط به `classify_knn.py` تابعی به اسم `knn_classifier` است که وظیفه‌ی `classify` کردن داده‌های `train` با هر ابعاد و تعداد دسته را بر عهده دارد و نیز داده‌های تست را `classify` میکند (`predict` میکند) (کد مربوط به بخش اصلی سوال)

```
@profile
def knn_classifier(k, x_train, y_train, x_test) -> pd.DataFrame:
    classifier = KNeighborsClassifier(n_neighbors=k)
    classifier.fit(x_train, y_train)
    prediction = classifier.predict(x_test)
    return prediction
```

□

در فایل `classify_knn.py` با استفاده از تابع فوق و تابع `classify_toy_data`، این الگوریتم را بر روی هر یک از مسائل A تا I انجام دادم که دسته‌بندی داده‌های تست در فایل‌های `csv` مربوط به هر مسئله در فولدر `/result/predictions` آمده است که نشان میدهد لیبل درست و پیش بینی الگوریتم هر کدام چه بوده

است. همچنین دقت و ماتریس‌های تداخل در فایل `problem1_1.json` واقع در `result` ذخیره شده است که به

شکل زیر است: (پاسخ بخش الف)

```
{  
    'A': {  
        'confusion_matrix': [[980, 20],  
                               [73, 927]],  
        'accuracy': 0.9535  
    },  
    'B': {  
        'confusion_matrix': [[839, 161],  
                               [253, 747]],  
        'accuracy': 0.793  
    },  
    'C': {  
        'confusion_matrix': [[842, 158],  
                               [292, 708]],  
        'accuracy': 0.775  
    },  
    'D': {  
        'confusion_matrix': [[948, 52],  
                               [215, 785]],  
        'accuracy': 0.8665  
    },  
    'E': {  
        'confusion_matrix': [[909, 91],  
                               [102, 898]],  
        'accuracy': 0.9035  
    },  
    'F': {  
        'confusion_matrix': [[1000, 0],  
                               [0, 1000]],  
        'accuracy': 1.0  
    },  
    'G': {  
        'confusion_matrix': [[959, 41],  
                               [59, 941]],  
        'accuracy': 0.95  
    },  
    'H': {  
        'confusion_matrix': [[999, 0],  
                               [0, 999]],  
        'accuracy': 1.0  
    },  
    'I': {  
        'confusion_matrix': [[1000, 0],  
                               [0, 1000]],  
        'accuracy': 1.0  
    }  
}
```

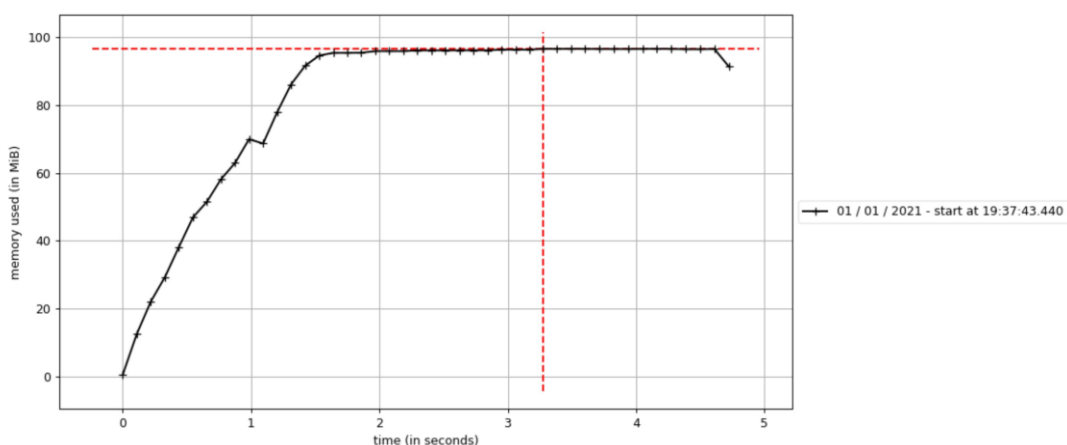
همچنین با استفاده از memory\_profile در پایتون مقدار memory و زمان نیز record شده است (با دستور

mprof run classify\_knn.py & mprof plot ) که البته به خاطر سنگین بودن خود این فرایند record زمان

بیشتر از حالت عادی است (حالت عادی این الگوریتم حدود ۱ ثانیه طول میکشد)

نمودار مصرف حافظه بر اساس زمان در فایل memory\_time واقع در result آمده است که به صورت

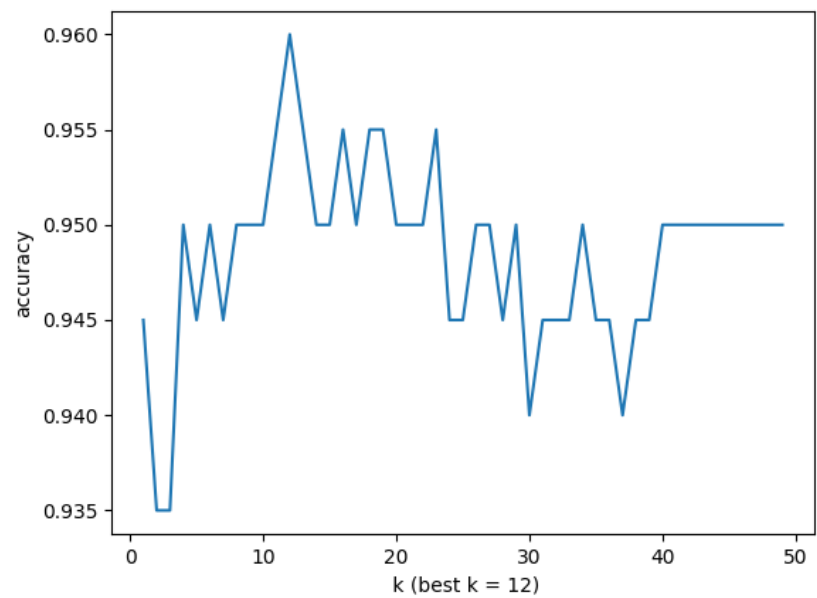
زیر است.



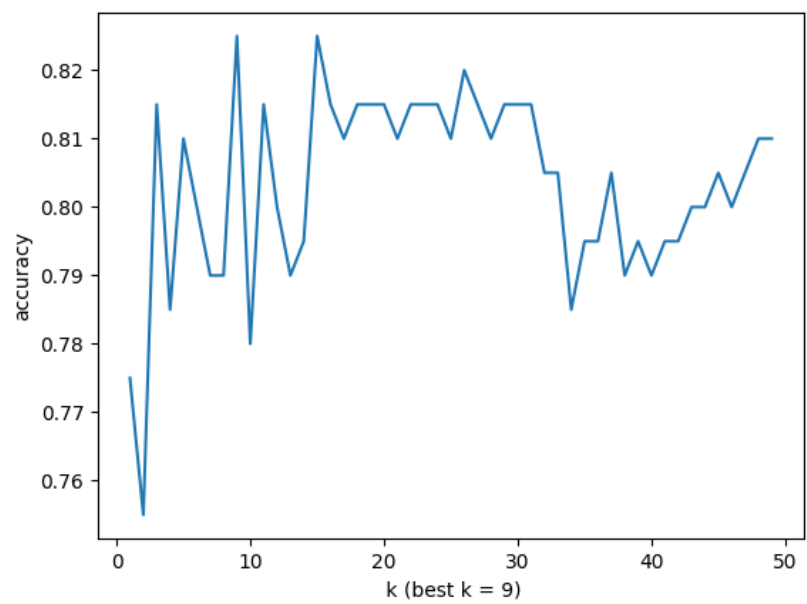
قسمت ب : در فایل best\_k.py عملیات مربوط به این بخش انجام شده است که در آن دقت روی داده‌های validation\_data بر حسب k را به تصویر میکشد که برای هر مسئله (نه فقط ۱) این موضوع انجام شده است و در تمام k های بین ۱ تا ۲۰۰ را به تصویر کشیده است. این تصاویر در فولدر /result/k\_accuracy\_plots آمده است که به شرح زیر نیز است:

A •

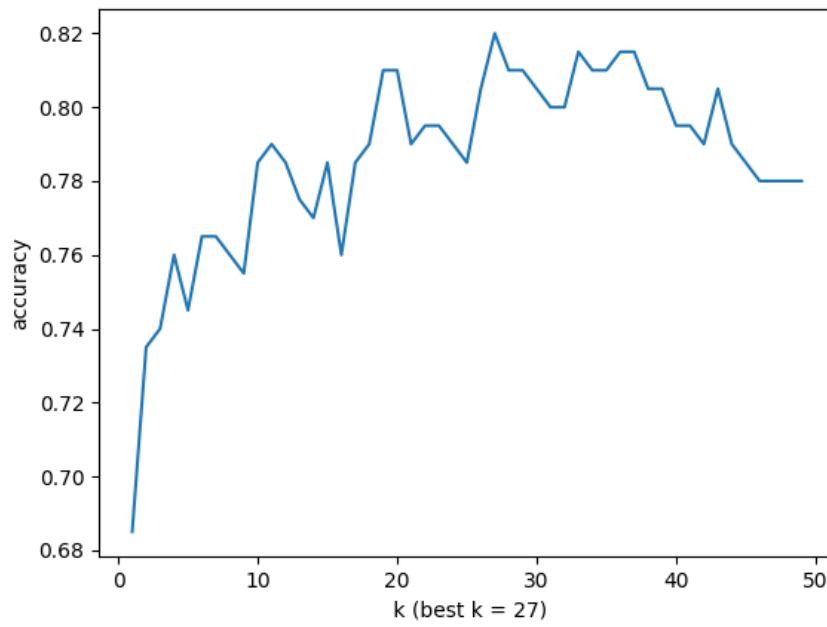




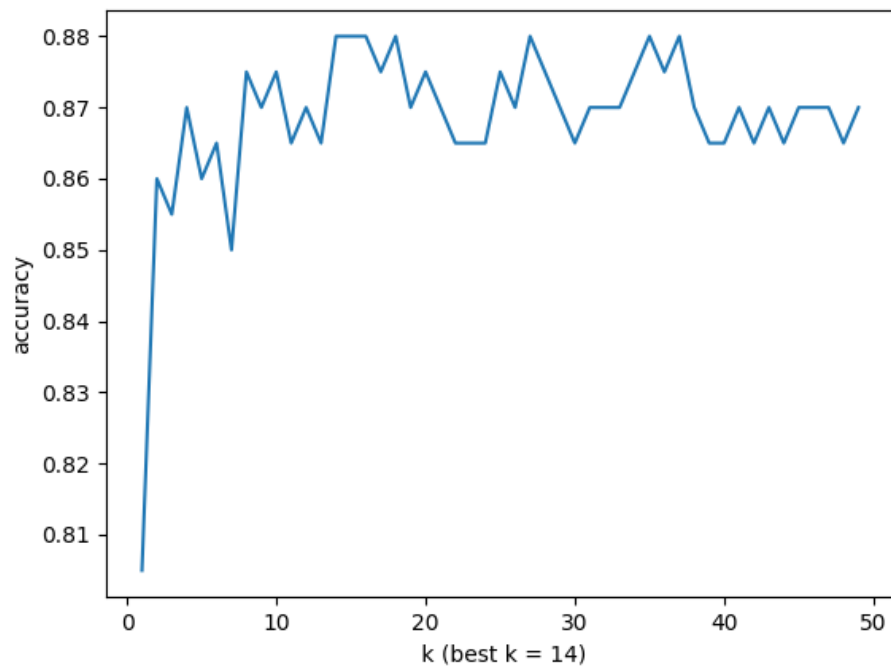
B •



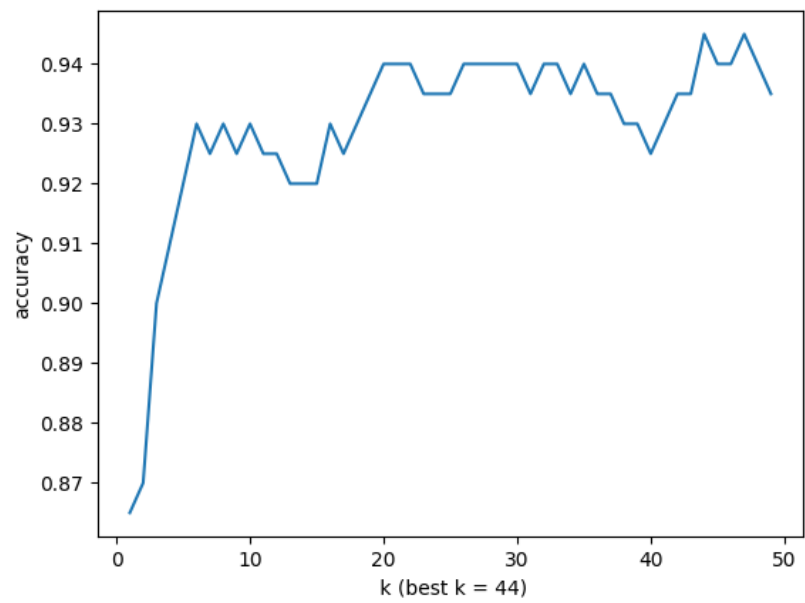
C •



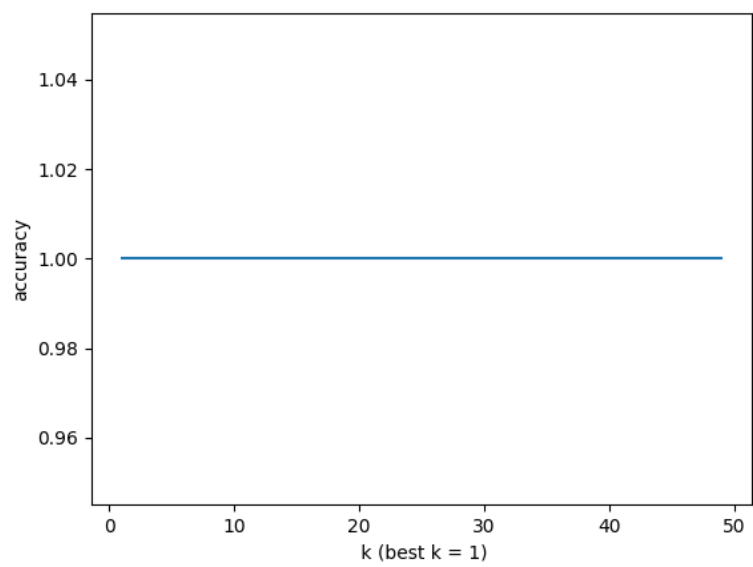
D •



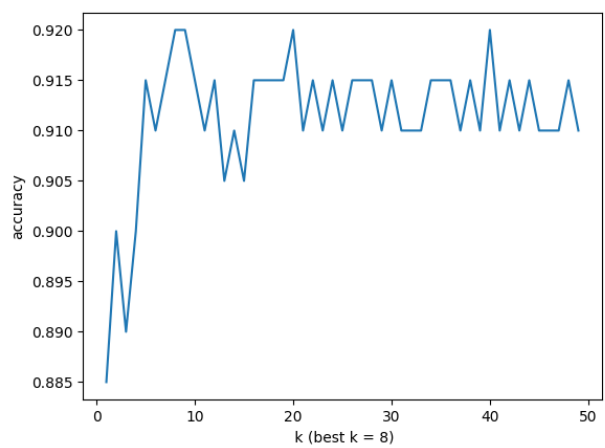
E •



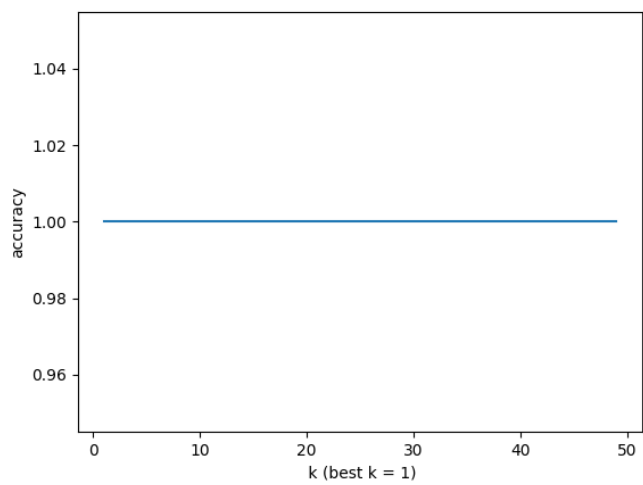
F •



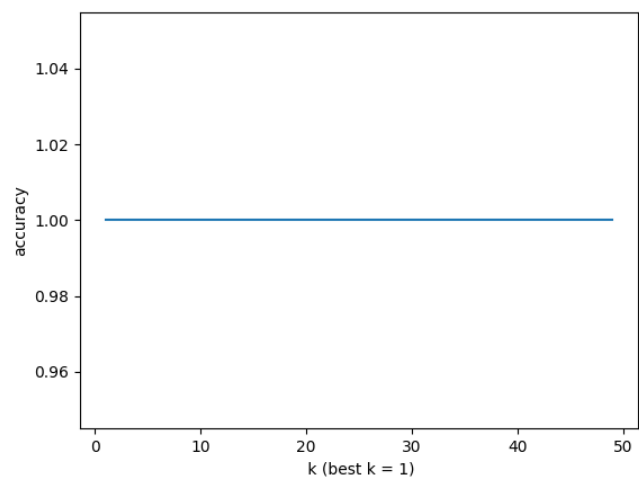
G •



H •



I •



که چون در دسته بندی ۱ داده‌های دو دسته به خوبی از هم تفکیک هستند و فاصله‌ی به نسبت خوبی با هم دارند همواره دقت ۱۰۰ درصد شده است و  $k$  دخیل نبوده است. در predict هایمان روی داده های تست نیز اثری نداشته و همان نتایج را به ارمغان آورده است.

(بخش ۱ را با  $k=1$  تست کردم)

```
'I': {
  'confusion_matrix': [[1000, 0],
                       [0, 1000]],
  'accuracy': 1.0
}
```

## سوال ۲

در اسن بخش dataهای استخراج شده از سوال 1 نیز آورده شده است.

کد های مربوط به این بخش در فولدر problem2 و فایل bayes\_classifier نوشته شده است.

تابع `class_multivariate_normal` یک توزیع نرمال برای هر کلاس متناسب با بردار میانگین و ماتریس کواریانس برمیگرداند

```
def class_multivariate_normal(mu, cov_matrix):
    N = multivariate_normal(mu, cov_matrix)
    return N
```

تابع bayes\_classifier بر اساس pdf هر سطر از داده  $p(w_i|x)$  را حساب میکند و  $\text{argmax}_i$  آن را حساب میکند و آن داده را به آن کلاس تخصیص میدهد. (تابع بخش اصلی سوال)

( $p(w_i|x)$  با استفاده از priorها و pdf مربوط به  $x$  محاسبه میشود)

```
def bayes_classifier(priors, test_data, classes_info):
    classes_N = []
    for class_info in classes_info:
        classes_N.append(class_multivariate_normal(mu=class_info["mu"], cov_matrix=class_info["cov_matrix"]))

    predictions = []
    for x in test_data:
        posteriors = np.array([classes_N[i].pdf(x) * priors[i] for i in range(len(priors))]) # p(x|w_i)
        predictions.append(posteriors.argmax() + 1)
    return predictions
```

بخش الف :

تابع مربوط به problem2\_1 مربوط به این بخش است که متناسب با اطلاعات داده شده در جدول داده‌های تست predict شده‌اند.

دسته بندی‌های داده‌های تست در /result/predictions/ مثل سوال قبل آورده شده است.

همچنین دقت و ماتریس تداخل نیز در فایل problem2\_1.json واقع در result نیز ذخیره شده است که به صورت زیر است:

```
{
  'A': {
    'confusion_matrix': [[980, 20],
                        [71, 929]],
    'accuracy': 0.9545
  },
  'B': {
    'confusion_matrix': [[801, 199],
                        [201, 799]],
    'accuracy': 0.8
  },
  'C': {
    'confusion_matrix': [[802, 198],
                        [193, 807]],
    'accuracy': 0.8045
  },
  'D': {
    'confusion_matrix': [[931, 69],
                        [217, 783]],
    'accuracy': 0.857
  },
  'E': {
    'confusion_matrix': [[913, 87],
                        [78, 922]],
    'accuracy': 0.9175
  },
  'F': {
    'confusion_matrix': [[1000, 0],
                        [0, 1000]],
    'accuracy': 1.0
  }
}
```

بخش ب :

کد مربوط به این بخش در تابع problem2\_2 آمده است که مثل همان problem2\_1 است با این تفاوت که ماتریس کواریانس قطری است.

نتایج در problem2\_2.json آمده است.

```

{
  'A': {
    'confusion_matrix': [[1000, 0],
                          [143, 857]],
    'accuracy': 0.9285
  },
  'B': {
    'confusion_matrix': [[1000, 0],
                          [1000, 0]],
    'accuracy': 0.5
  },
  'C': {
    'confusion_matrix': [[802, 198],
                          [193, 807]],
    'accuracy': 0.8045
  },
  'D': {
    'confusion_matrix': [[931, 69],
                          [217, 783]],
    'accuracy': 0.857
  },
  'E': {
    'confusion_matrix': [[825, 175],
                          [168, 832]],
    'accuracy': 0.8285
  },
  'F': {
    'confusion_matrix': [[1000, 0],
                          [0, 1000]],
    'accuracy': 1.0
  }
}

```

بخش ب و پ :

دقت‌ها به صورت زیر است:

	A	B	C	D	E	F
KNN	۰/۹۵	۰/۷۹	۰/۷۷	۰/۸۶	۰/۹۰	۱/۰۰
BAYES_NONISOTROPIC	۱/۰۰	۰/۹۱	۰/۸۵	۰/۸۰	۰/۸۰	۰/۹۵
BAYES_ISOTROPIC	۱/۰۰	۰/۸۲	۰/۸۵	۰/۸۰	۰/۵۰	۰/۹۲

در صورتی که ماتریس های کوواریانس هر یک از مجموعه داده ها را به صورت قطری فرض کنیم، دقت دسته بندی برای داده هایی که شکل دایره ای ندارند کاهش می یابد. دلیل آن پدیده ای به اسم برازش مدلی ایزوتروپیک بر روی داده های غیر ایزوتروپیک است.

به دلیل داشتن توزیع نرمال گوسین این مسائل و به دلیل بهینه بودن دسته بند بیز برای این مسائل دقت مربوط به دسته بند بیز بیشتر از knn است.

### سوال ۳ :

کدهای مربوط به این سوال در پوشه ی problem3 موجود است .

در این پوشه در فولدر images عکس های مربوطه موجود است.

کدهای مربوطه در فایل problem3.py موجود است.

تابع load\_image وظیفه ی خواندن image ها و تبدیل به داده ها چندبعدی را دارد.

```
def load_images(path):
    img = imread(path)
    images = []
    for i in range(34):
        for j in range(33):
            m = img[i * 16:(i + 1) * 16, j * 16:(j + 1) * 16]
            if not (j == 32 and i >= 12):
                images.append(np.squeeze(m.T.reshape(-1)))
    return images
```

تابع get\_data وظیفه ی خواندن عکس ها را دارد و به آن ها برچسب هر کدام از اعداد را میدهد. همچنین داده ها train و test نیز جدا میکند (از هر عدد نصف برای train و نصف برای test). همچنین داده ها کمی به دست آمده در فولدر results/data برای هر عدد ذخیره شده است.

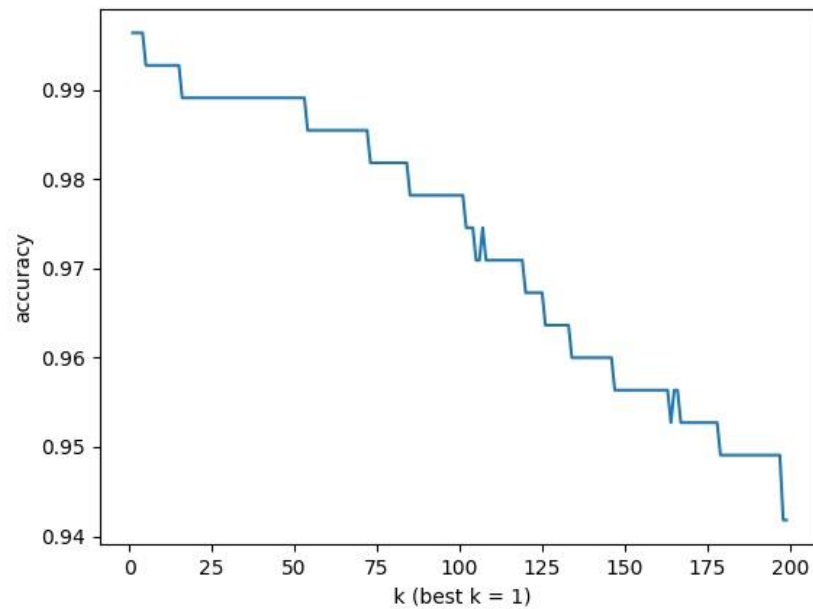


تابع problem3\_1 بخش اول این سوال را انجام میدهد که الگوریتم knn را برای آن اجرا میکند و پیشبینی‌ها را در فایل predictions.csv در فولدر results ذخیره شده است. همچنین ماتریس تداخل و دقت را نیز در فایل accuracy.json در results ذخیره شده است. (با استفاده از توابع سوال ۱)

```
'confusion_matrix': [[548, 0, 0, 1, 1],  
                     [17, 526, 6, 0, 1],  
                     [4, 3, 535, 0, 8],  
                     [50, 1, 0, 499, 0],  
                     [3, 1, 17, 1, 528]],  
'accuracy': 0.9585454545454546
```

با استفاده از  $k=10$  دقت ۹۵ درصدی حاصل شده است و بیشترین خطایی که وجود داشته به دلیل شباهت ۱ و ۴ بوده است و بیتر به دلیل نامناسب بودن اشکال ۱ بوده است.

کد مربوط به بخش دوم نیز در تابع problem3\_2 آمده است که دیتاست مربوط به validation\_data جدا میشود و به ازای  $k$  های مختلف این بررسی انجام شد که حاصل آن عکس k\_plot.jpg واقع در فولدر results است.



که بهترین  $k$  برای ۱ (به دلیل خوب تفکیک اعداد) .  
همچنین با  $k$  برابر ۱ نیز بخش قبل تکرار شد که دقت بالاتری داده است. و نتیجه در `accuracy_with_best_k.json` ذخیره میشود.

```
{
  'confusion_matrix': [[275, 0, 0, 0, 0],
                        [10, 536, 0, 0, 4],
                        [1, 4, 531, 0, 14],
                        [17, 2, 0, 531, 0],
                        [1, 0, 17, 1, 531]],
  'accuracy': 0.9713131313131314
}
```

که دقت به 97 درصد رسیده است.

#### سوال ۴ :

این بخش در فولدر problem4 است که در فولدر images عکس ها موجود است و در فایل problem4.py کد های مربوطه موجود است و با استفاده از تابع GaussianNB عملیات دسته بندی با بیز را به طور کامل انجام داده ام و پیشبینی ها در فایل prediction.csv واقع در result ذخیره شده است و همچنین دقت و ماتریس تداخل نیز در accuracy.json نیز ذخیره شده است.

```
{
  'confusion_matrix': [[483, 37, 2, 21, 7],
                        [12, 521, 5, 6, 6],
                        [17, 50, 463, 3, 17],
                        [27, 18, 0, 500, 5],
                        [17, 10, 46, 7, 470]],
  'accuracy': 0.8861818181818182
}
```

که نسبت به knn دقت پایین آمده است (به دلیل نداشتن ساختار گوسین نرمال داده ها) و بخش ب نیز مثل همان بخش ب ی سوال ۲ میباشد.