پروژه SVM

بخش اول – الف

در این تمرین با استفاده از 4 مقدار مختلف برای C (0.1 , 1 , 10 , 100) مرز های تصمیم SVM را بررسی کردیم.

```
mat = loadmat('./Dataset/data1.mat')
X = mat['X']
y = mat['y'].astype(int).reshape(-1)
x_class_0 = X[y == 0]
x_class_1 = X[y == 1]
C_values = [0.1, 1, 10, 100]
```

با این تابع Y را به یک بعد تبدیل می کنیم

در این بخش دیتاست رو لود کردیم و مقادیر C را مشخص کردیم.

```
for i in range(len(C_values)):
    model = SVC(C=C_values[i], kernel='linear')
    model.fit(X, y)
```

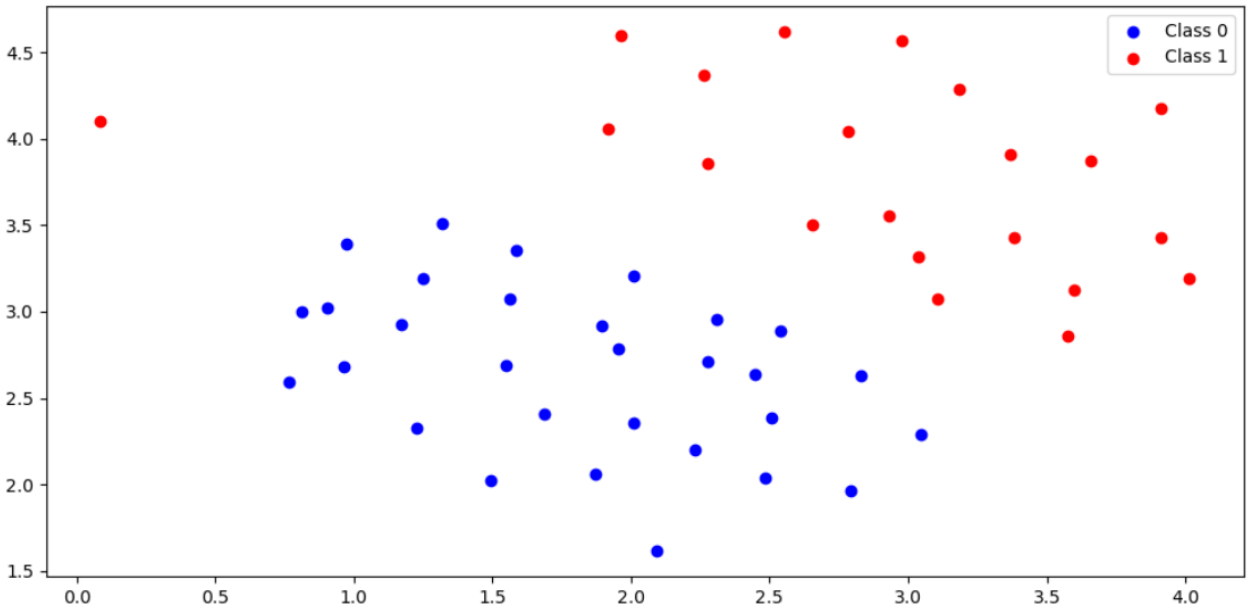
برای ساخت مدل SVM یک حلقه با تکرار تعداد مقادیر C ایجاد کرده ایم که هر بار یک سری عملیات انجام می‌دهد که در ادامه توضیح خواهم داد.

اینجا مدل ما svm با کرنل خطی است که مقدار C آن هر بار تغییر می‌کند.

```
plt.figure(figsize=(10, 5))
plt.scatter(x_class_0[:,0], x_class_0[:,1] , color='blue', label='Class 0')
plt.scatter(x_class_1[:,0], x_class_1[:,1] , color='red', label='Class 1')
```

در این بخش داده های درون دیتاست را اسکتر می‌کنیم.

نتیجه کد:




```
plt.scatter(model.support_vectors_[:, 0], model.support_vectors_[:, 1], facecolors='none', edgecolors='black' , label="Support Vectors")
```

با استفاده از این دستور ساپورت وکتور های مدل را پلات می‌کنیم.

```
x_min = X[:, 0].min() - 1
x_max = X[:, 0].max() + 1
y_min = X[:, 1].min() - 1
y_max = X[:, 1].max() + 1

XX, YY = np.meshgrid(np.linspace(x_min, x_max, 200), np.linspace(y_min, y_max, 200))
Z = model.decision_function(np.c_[XX.ravel(), YY.ravel()])
Z = Z.reshape(XX.shape)
plt.contour(XX, YY, Z, colors=['gray', 'black', 'gray'], levels=[-1, 0, 1], linestyles=['--', '-', '--'], linewidths=[1, 1.5, 1])
```

در اینجا مقدار مینیموم و ماکسیمم x ها و y ها را پیدا می‌کنیم.

با استفاده از linspace و تابع meshgrid یک شبکه مش ایجاد می‌کنیم تا بتوانیم مرز تصمیم و حاشیه‌ها را رسم کنیم.

با استفاده از contour مرز اصلی تصمیم و دو حاشیه آن را نشان می‌دهیم.

```
plt.title(f"C = {C_values[i]}")
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.tight_layout()
plt.legend()
plt.show()
```

در این بخش عنوان هر نمودار را مشخص می کنیم که مربوط به c چند است.

مقادیر روی محور x و y را مشخص می‌کنیم و نمودار را رسم می‌کنیم.

```
acc_svm = accuracy_score(y, model.predict(X))
print(f"SVM Accuracy : {acc_svm*100}")
print(f"SVM Support Vectors count = {len(model.support_)}")
```

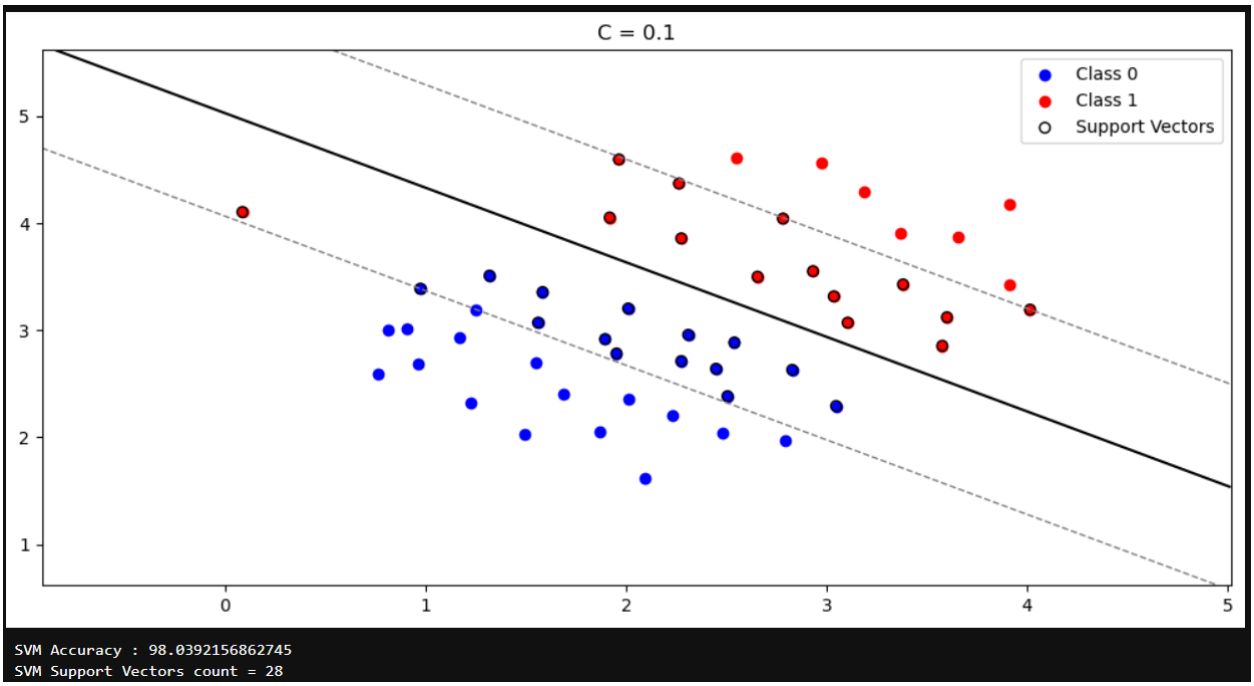
اینجا میزان دقت مدل را با استفاده از accuracy_score می‌سنجیم.

سپس تعداد ساپورت وکتور های مدل را نمایش می‌دهیم.

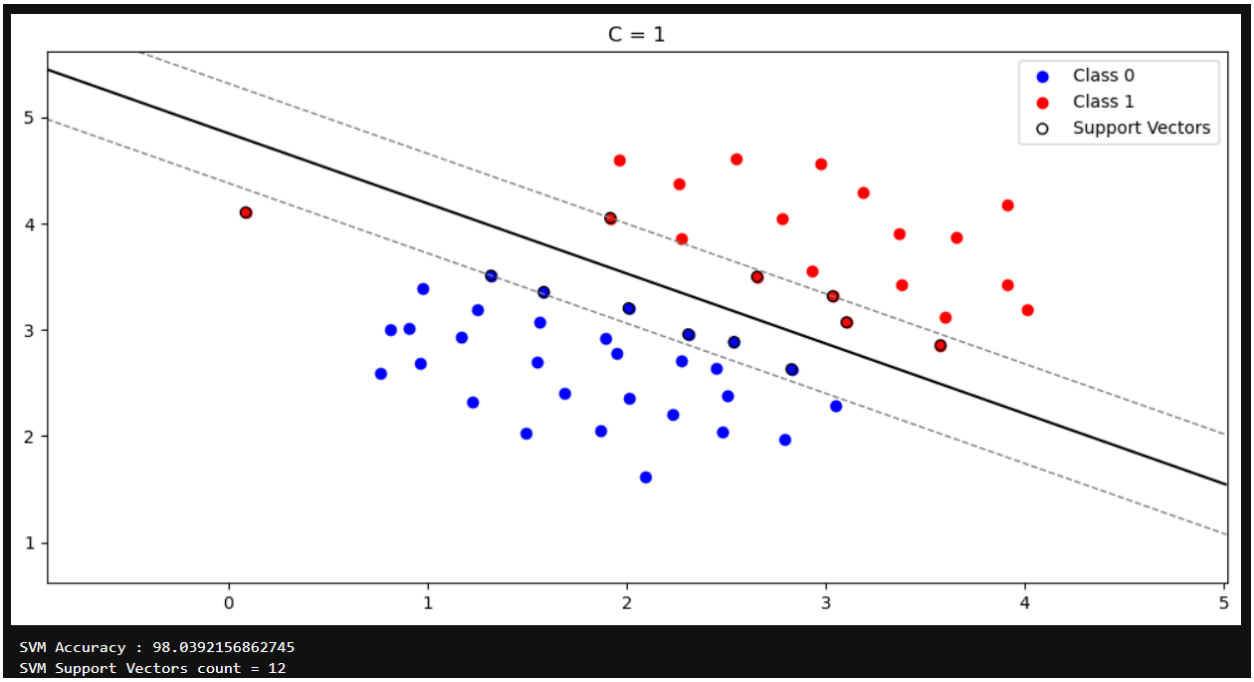
این روند 4 بار تکرار می‌شود و 4 مدل svm ایجاد می‌شود که

خروجی نهایی به شکل زیر است:

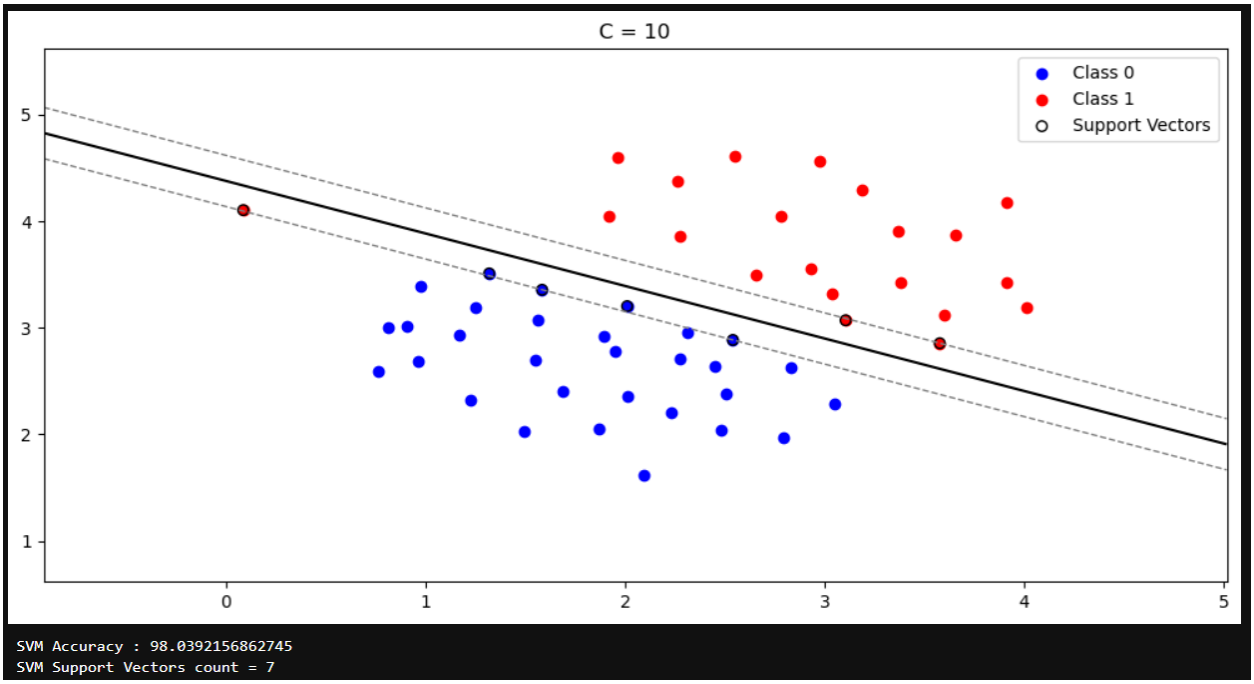
مدل با C = 0.1



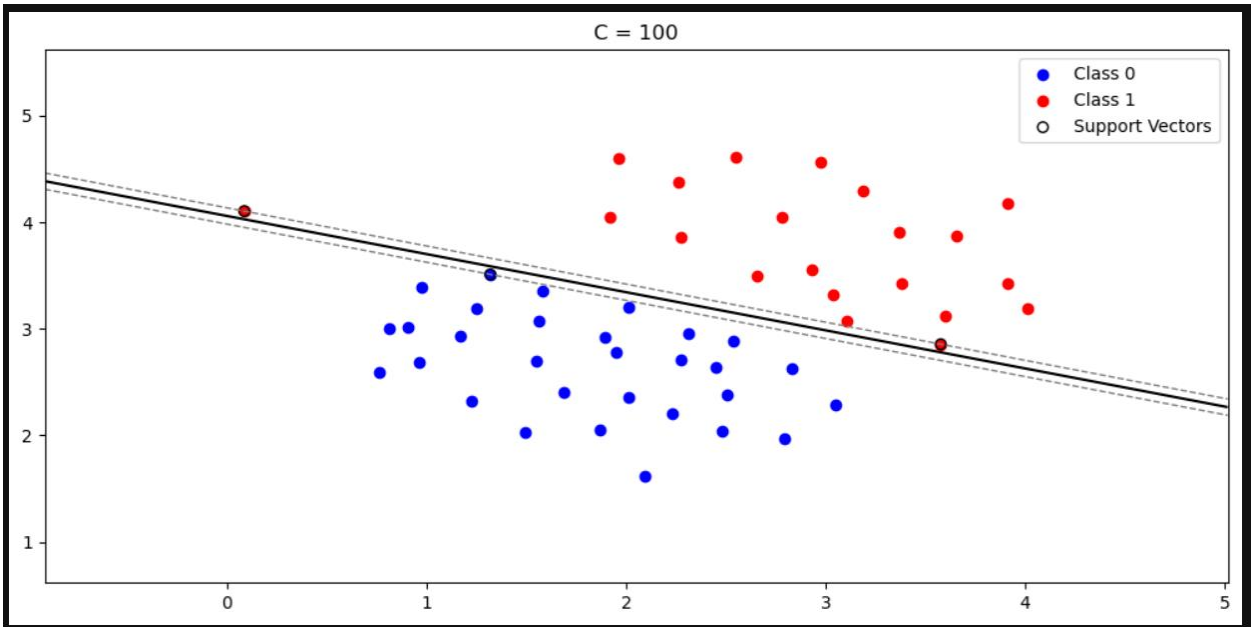
مدل با C = 1



مدل با C = 10



مدل با C = 100



با استفاده از Logistic regression

```
from sklearn.linear_model import LogisticRegression

data = loadmat('./Dataset/data1.mat')
X = data['X']
y = data['y'].astype(int).reshape(-1)
x_class_0 = X[y == 0]
x_class_1 = X[y == 1]

log_reg = LogisticRegression()
log_reg.fit(X, y)
```

از مدل های خطی لاجستیک رگرشن رو ایمپورت کردیم و دیتاست رو لود کردیم.

یک مدل از این نوع ساختیم و دیتا رو روش فیت کردیم.

```
plt.figure(figsize=(10, 5))
plt.title("Logistic regression")
plt.scatter(x_class_0[:,0], x_class_0[:,1] , color='blue', label='Class 0')
plt.scatter(x_class_1[:,0], x_class_1[:,1] , color='red', label='Class 1')
```

در این بخش داده های دیتاست رو که به دو کلاس 0 و 1 دسته بندی شده‌اند را بر روی نمودار پراکندگی رسم می‌کنیم.

```
x_min = X[:, 0].min() - 1
x_max = X[:, 0].max() + 1
y_min = X[:, 1].min() - 1
y_max = X[:, 1].max() + 1

XX, YY = np.meshgrid(np.linspace(x_min, x_max, 200), np.linspace(y_min, y_max, 200))
Z_log = log_reg.predict(np.c_[XX.ravel(), YY.ravel()]).reshape(XX.shape)
plt.contour(XX, YY, Z_log, levels=[0.5], colors='black', linewidths = 1)
```

در این بخش همانند بخش svm مقدار مینیموم و ماکسیمم x ها و y ها را پیدا می‌کنیم.

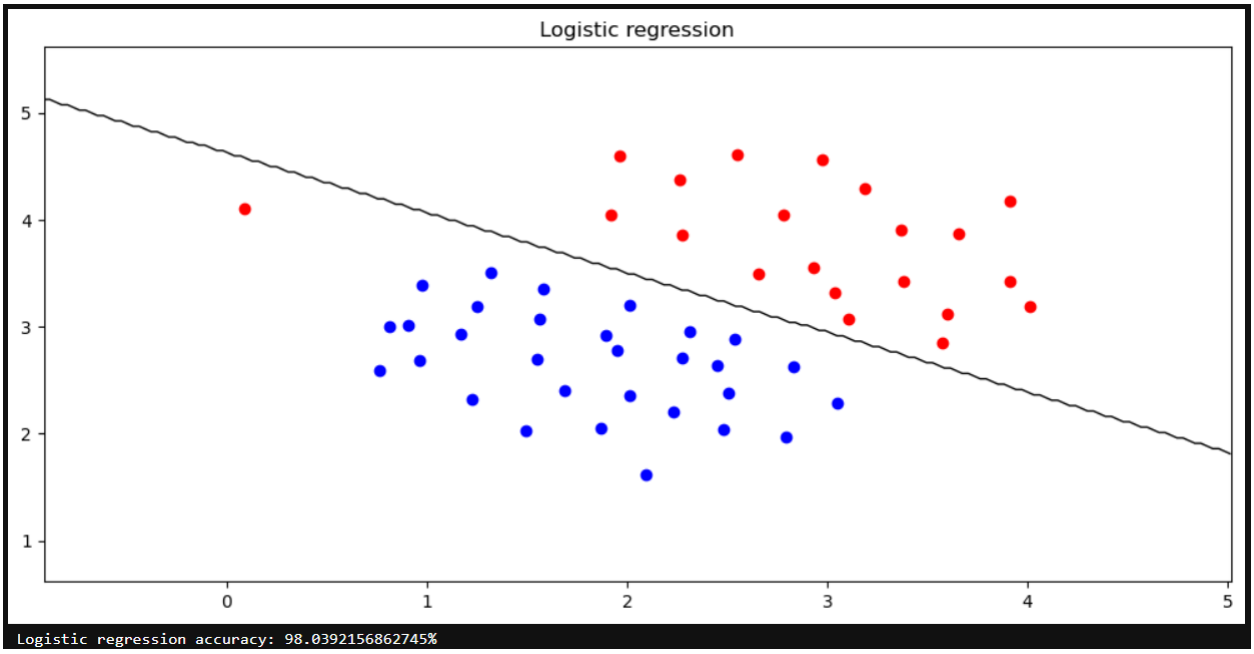
با استفاده از linspace و تابع meshgrid یک شبکه مش برای رسم پیش بینی های مدل ایجاد می‌کنیم.

با استفاده از مدل جواب نهایی را پیشبینی می‌کنیم و در نهایت خط ایجاد شده را رسم می‌کنیم.

```
plt.tight_layout()
plt.show()
y_pred_log = log_reg.predict(X)
accuracy_log = accuracy_score(y, y_pred_log)
print(f"Logistic regression accuracy: {accuracy_log * 100}")
```

نمودار را رسم می‌کنیم.

با استفاده از مدل برچسب های مقادیر x را پیشبینی می‌کنیم و سپس میزان دقت مدل را با استفاده از برچسب های واقعی می‌سنجیم و آن را نمایش می‌دهیم که به شکل زیر است:



نتیجه گیری :

در svm پارامتر c نقش بسزایی در دقت مدل دارد ؛ همانطور که مشاهده شد هر چه مقدار c بیشتر شد تعداد ساپورت وکتور ها نیز افزایش یافت و از طرفی مدل تلاش می‌کند همه نمونه‌ها را به درستی دسته‌بندی کند و حاشیه ها کمتر می‌شود و مرز تصمیم تنگ تر می‌شود و به داده های پرت بیشتر توجه کند.

از این رو با افزایش بی اندازه c ممکن است دچار overfitting شویم.

و حال اگر مقدار c کم باشد مدل به داده های پرت کمتر توجه می کند و باعث می شود حاشیه ها بیشتر شود اما ممکن است دقت مدل کاهش یابد.

مقایسه SVM و Logistic Regression

در SVM یافتن مرز بهینه بین کلاس‌ها قوی‌تر است چرا که مرز تصمیم را به گونه‌ای تنظیم می‌کند که حاشیه بیشتری بین دو کلاس ایجاد شود اما Logistic Regression یک مدل احتمالی است و احتمال تعلق به یک کلاس خاص را محاسبه می‌کند.

در داده های تفکیک پذیر خطی مثل این دیتاست مرز تصمیم مدل رگرسیون با مدل svm بسیار نزدیک است.

در مدل رگرسیون داده‌های پرت می‌توانند بر مرز تصمیم تأثیر بگذارند و باعث تغییر آن شوند ولی در مدل SVM این تأثیر بستگی به مقدار C دارد.

با تنظیم مناسب C می‌توان مدل را به‌گونه‌ای تنظیم کرد که تأثیر داده‌های پرت کاهش یابد.

بخش اول – ب

مدل خطی

برای حل این سوال ابتدا با استفاده از کرنل خطی svm داده هارو دسته بندی می‌کنیم.

Linear Kernel

```
]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

   linear_svm = SVC(kernel='linear')
   linear_svm.fit(X_train, y_train)
```

اینجا با استفاده از لایبرری train_test_split داده ها را به دو بخش Train و Test تقسیم کردیم و مدل svm را با کرنل خطی به روی داده ها فیت کرده ایم.

```python
def plot_decision_boundary(model, X, y, title):
    x_min, x_max = X[:, 0].min() - 0.1, X[:, 0].max() + 0.1
    y_min, y_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),np.arange(y_min, y_max, 0.01))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.figure(figsize=(10, 5))
    plt.contourf(xx, yy, Z, alpha=0.1)
    plt.scatter(x_class_0[:,0], x_class_0[:,1] , color='blue', label='Class 0')
    plt.scatter(x_class_1[:,0], x_class_1[:,1] , color='red', label='Class 1')
    plt.title(title)
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.legend()
    plt.show()
```

در کد بالا یک تابع برای رسم مرز تصمیم تعریف کرده ایم که ورودی های آن مدل ، X ، Y و عنوان است.

در ادامه مقدار مین و مکس داده های 2 ویژگی را حساب می کنیم

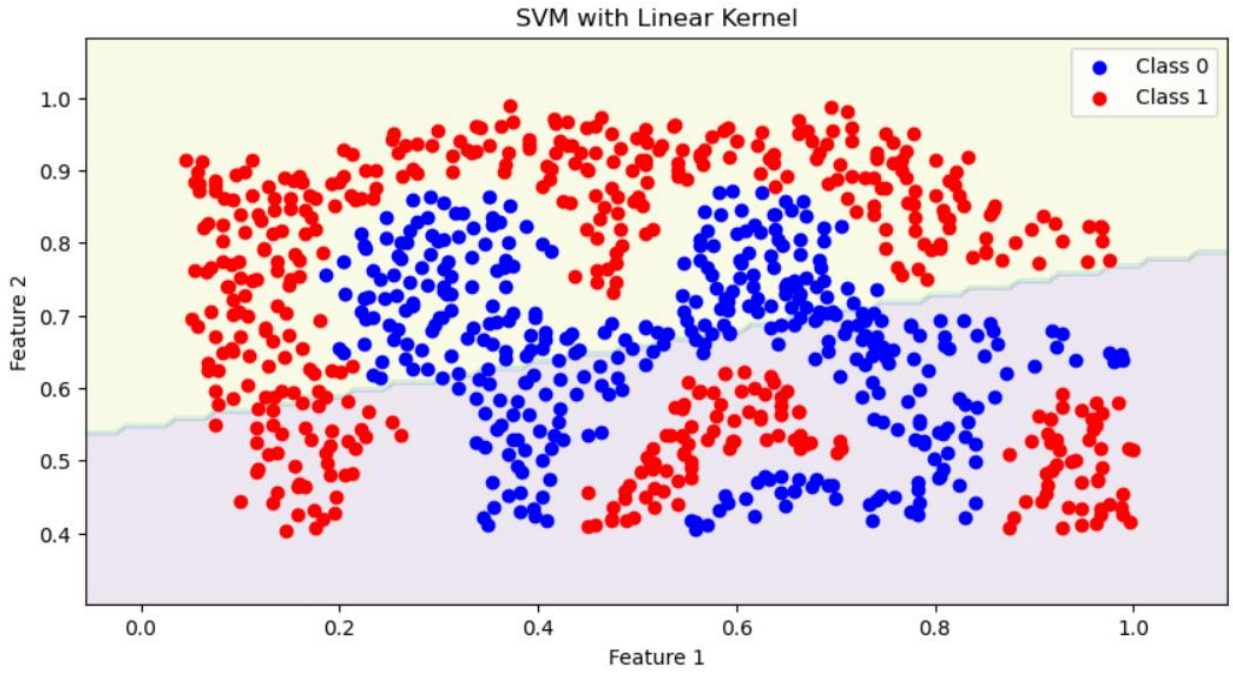
یک شبکه مش ایجاد می کنیم تا پیش بینی های مدل را بتوانیم رسم کنیم.

متغیر Z شامل پیش‌بینی های مدل برای نقاط ایجاد شده است و سپس شکل آن را به شکل آرایه xx تغییر می‌دهیم.

پس از ساخت فیگور از تابع contourf برای رسم سطوح پر شده استفاده می‌کنیم. این سطوح بر اساس مقادیر Z که پیش‌بینی‌های مدل هستند، رسم می‌شوند.
سپس نقاط داده شده را رسم می کنیم. (نقاط مربوط به کلاس ۰ (آبی) و نقاط مربوط به کلاس 1 (قرمز).
و در نهایت نمودار را نمایش می‌دهیم.

```python
plot_decision_boundary(linear_svm, X, y, title="SVM with Linear Kernel")
```

حالا با استفاده از تابعی که ایجاد کردیم نمودار مدل خطی آن را رسم می کنیم.




```
y_train_linear = linear_svm.predict(X_train)
y_test_linear = linear_svm.predict(X_test)

linear_train_accuracy = accuracy_score(y_train, y_train_linear)
linear_test_accuracy = accuracy_score(y_test, y_test_linear)

print(f"Train accuracy with Linear Kernel: {linear_train_accuracy:.2f}")
print(f"Test accuracy with Linear Kernel: {linear_test_accuracy:.2f}")
```

و در ادامه میزان دقت مدل در داده های آموزش و داده های تست را با استفاده از کتابخانه مربوطه محاسبه می کنیم.

```
Train accuracy with Linear Kernel: 0.57
Test accuracy with Linear Kernel: 0.59
```

مدل چند جمله‌ای

حال سعی می کنیم با استفاده از کرنل چند جمله ای میزان دقت را افزایش دهیم.

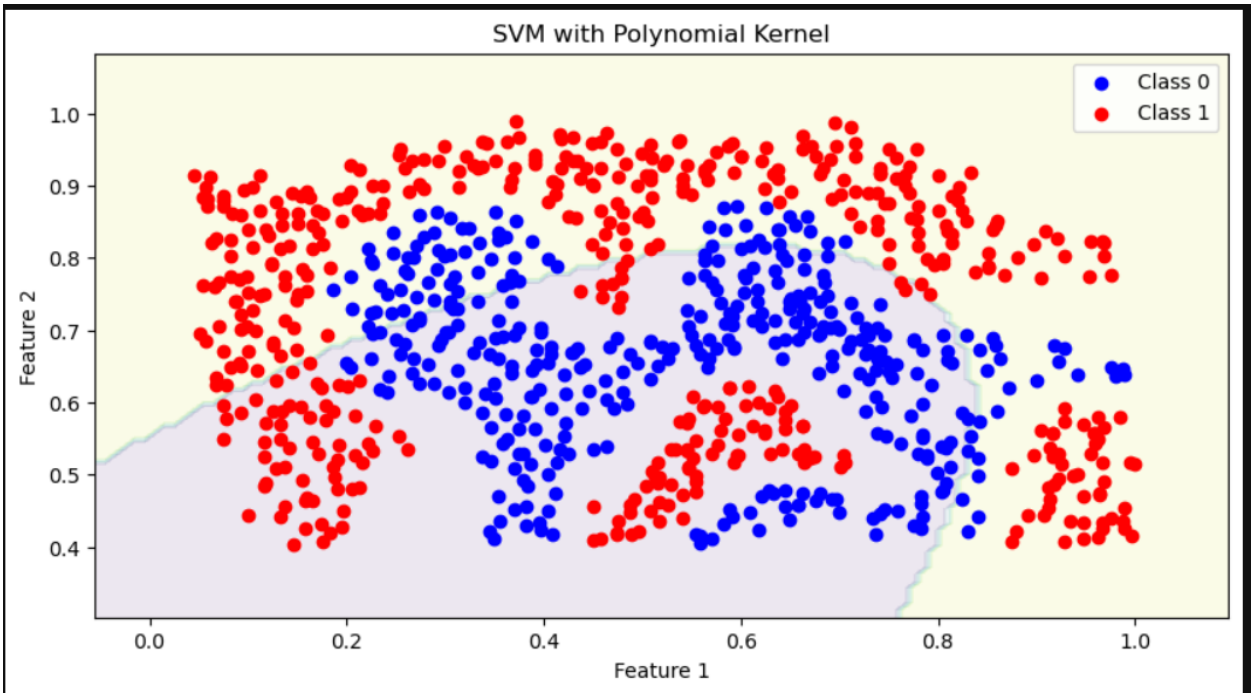
Polynomial Kernel

```python
poly_svm = SVC(kernel='poly', degree=2)
poly_svm.fit(X_train, y_train)

plot_decision_boundary(poly_svm, X, y, title="SVM with Polynomial Kernel")


y_train_poly = poly_svm.predict(X_train)
y_test_poly = poly_svm.predict(X_test)

poly_train_accuracy = accuracy_score(y_train, y_train_poly)
poly_test_accuracy = accuracy_score(y_test, y_test_poly)

print(f"Train accuracy with Polynomial Kernel: {poly_train_accuracy:.2f}")
print(f"Test accuracy with Polynomial Kernel: {poly_test_accuracy:.2f}")
```

اول مدلمان را با استفاده از کرنل چند جمله ای و درجه 2 روی داده ها فیت می کنیم.

سپس تابع رسم مرز تصمیم را صدا می کنیم تا نمودار رسم شود.

و در ادامه میزان دقت مدل روی داده های Train و Test را محاسبه می کنیم که بهتر از مدل خطی است.



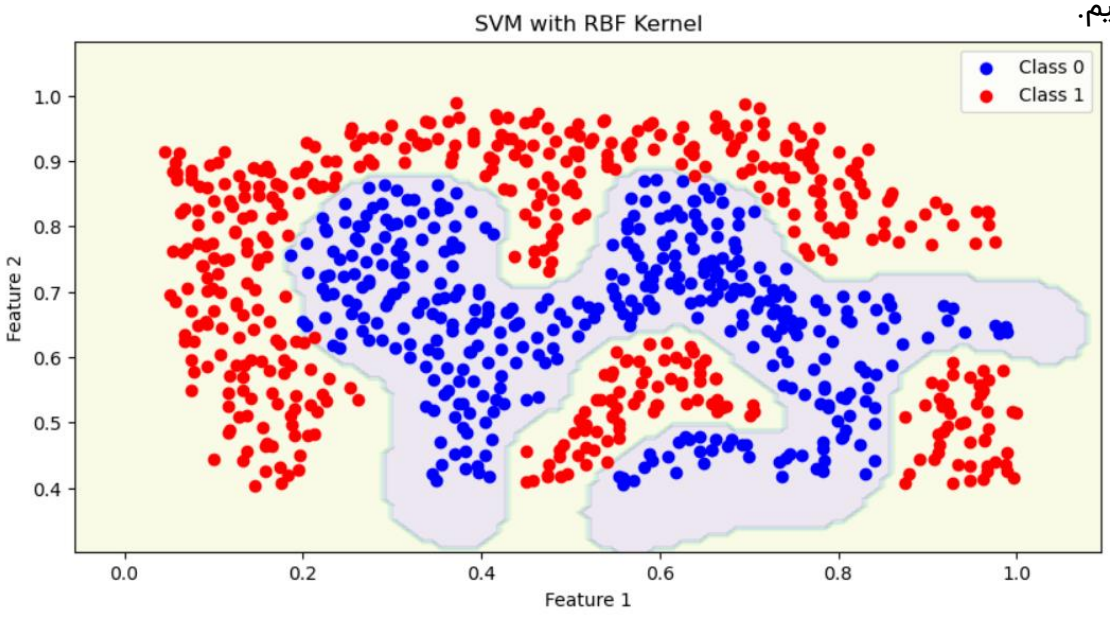
مدل گاوسی

RBF Kernel

```
C_values = np.arange(1,200,5).tolist()
gamma_values = np.arange(0,200,5).tolist()

best_score = 0
best_model = None
for C in C_values:
    for gamma in gamma_values:
        model = SVC(kernel='rbf' , C=C, gamma=gamma)
        model.fit(X_train, y_train)
        score = model.score(X_train, y_train)
        if score > best_score:
            best_score = score
            best_model = model

rbf_svm = best_model
rbf_svm.fit(X_train, y_train)

plot_decision_boundary(rbf_svm, X, y, title="SVM with RBF Kernel")

y_train_rbf = rbf_svm.predict(X_train)
y_test_rbf = rbf_svm.predict(X_test)

rbf_train_accuracy = accuracy_score(y_train, y_train_rbf)
rbf_test_accuracy = accuracy_score(y_test, y_test_rbf)

print(f"Train accuracy with Rbf Kernel: {rbf_train_accuracy:.2f}")
print(f"Best parameters: C={rbf_svm.C} , gamma={rbf_svm.gamma}")
print(f"Test accuracy with Rbf Kernel: {rbf_test_accuracy:.2f}")
```

در اینجا از کرنل RBF استفاده کردیم با استفاده از 2 حلقه فور تو در تو سعی کردیم بهترین پارامتر ها را برای مدل انتخاب کنیم.

سپس داده ها را فیت کردیم و در ادامه با استفاده از تابعی که بالاتر توضیح دادیم نمودار مدل را رسم می‌کنیم.




```
Train accuracy with Rbf Kernel: 1.00
Best parameters: C=11 , gamma=180
Test accuracy with Rbf Kernel: 0.99
```

در اینجا مشاهده شد که برای این دیتاست مدل RBF مناسب تر است و بهترین پارامتر برای این مدل برابر مقادیر بالا است و دقت تست برابر است با 99 درصد.

بخش اول – ج

برای پیدا کردن بهترین مدل این دیتاست باید هایپرپارامتر های مختلف را بررسی کنیم.

می توانیم برای انجام این کار از حلقه های تو در تو و یا از کتابخانه GridSearchCV استفاده کنیم.

که ما از روش دوم استفاده می کنیم.

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
Xval_scaled = scaler.transform(Xval)
```

برای بهبود نتیجه باید داده هایمان را نرمال‌سازی کنیم(Standardize)؛ که برای اینکار از کتابخانه StandardScaler استفاده می کنیم.

این عمل باعث می‌شود که داده‌ها به توزیع نرمال نزدیک‌تر شوند و مدل ما بهتر عمل کند.

```
param_grid = {
    'C': np.arange(1, 20, 1),
    'gamma': ('scale', 'auto'),
    'coef0' : np.arange(0.1,10,0.5),
    'degree' : np.arange(2,10,1)
}
```

برای پیدا کردن بهترین هایپر پارامتر ها باید میزان تغییرات هر هایپرپارامتر را مشخص کنیم.

مقادیر هر هایپر پارامتر به صورت زیر است:

مقادیر C:

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```

مقادیر گاما:

```
('scale', 'auto')
```

مقادیر coef0:

```
[0.1 0.6 1.1 1.6 2.1 2.6 3.1 3.6 4.1 4.6 5.1 5.6 6.1 6.6 7.1 7.6 8.1 8.6
 9.1 9.6]
```

مقادیر degree:

```
[2 3 4 5 6 7 8 9]
```

در ادامه یک مدل svm با کرنل RBF میسازیم

```
svm = SVC(kernel = 'rbf')
grid_search = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_scaled, y.ravel())
```

در این بخش GridSearchCV با استفاده از پارامتر های که ایجاد کردیم بهترین مدل را پیدا میکند.
مقدار cv به تعداد تقسیم‌ها (folds) در کراس ولیدیشن اشاره دارد. یعنی داده‌ها به ۵ بخش تقسیم می‌شوند و مدل روی هر بخش آزمایش می‌شود.

```
best_svm = grid_search.best_estimator_

yval_pred = best_svm.predict(Xval_scaled)
validation_accuracy = accuracy_score(yval, yval_pred)
```

اینجا بهترین مدلی که آموزش دیده به متغیر best_svm اختصاص داده میشود.
میزان دقت مدل با داده های های ولیدیشن را محاسبه میکنیم.

```
print(f"Best Parameters : {grid_search.best_params_}")
print(f"The validation accuracy {validation_accuracy}")
```

در انتها بهترین مقادیر برای مدل و میزان دقت مدل (95 درصد) را چاپ می کنیم.

```
Best Parameters : {'C': 14, 'coef0': 0.1, 'degree': 2, 'gamma': 'scale'}
The validation accuracy 0.955
```

سوال دوم - الف

کد این سوال در فایل Problem2 - MNIST.ipynb قرار گرفته است.

```
MNIST = np.load('./data/mnist.npz')
X_train = MNIST['x_train']
X_test = MNIST['x_test']
y_train = MNIST['y_train']
y_test = MNIST['y_test']
X_train = X_train.reshape(-1, 28*28)
X_test = X_test.reshape(-1, 28*28)
```

دیتاست MNIST را دانلود کرده و داده های Train و Test را جدا می کنیم.
سپس ابعاد این داده ها به (6000,784) تبدیل می کنیم.

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)
```

برای آموزش بهتر مدل باید داده هایمان را نرمال سازی (Standardize) کنیم تا روند آموزش بهتر پیش برود که این کار را با استفاده از کتابخانه StandardScaler انجام دادیم.

```
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

با استفاده از کتابخانه PCA که برای کاهش بعد است، ویژگی های داده ها را از 784 به 2 تغییر می دهیم.

```
samples_for_plot = np.concatenate((X_train_pca, y_train.reshape(-1,1)),axis=1)
indexes_for_plt = np.array([])
for i in range(10):
    counter = 1
    while counter!=101:
        index = np.random.randint(0,60000, size=1)
        index = index[0]
        if samples_for_plot[index][2]==i:
            indexes_for_plt = np.append(indexes_for_plt,index)
            counter+=1
```

برای رسم نمودار نیاز داریم که قسمتی از داده ها را برای بهتر دیده شدن پراکندگی داده ها نمایش دهیم و از آنجایی که در سوال ذکر شده که توزیع هر کلاس باید یکسان باشد باید مراحل بالا را انجام دهیم.

دید کلی الگوریتم بالا:

Y های مربوط به X را به انتهای ویژگی های X کانکت می کنیم تا بتوانیم برای توزیع یکسان هر کلاس از آن استفاده کنیم. ابعاد به دست آمده (6000,3) که بعد سوم همان لیبل عکس ها است.

سپس یک حلقه فور به تکرار تعداد کلاس ها داریم که 10 است ایجاد کردیم و یک حلقه while برای پیدا کردن 100 عدد برای ایندکس هر عکس با استفاده از نامپای ایجاد می کنیم که در مجموع 1000 عکس داریم (100 تا برای هر عدد)

بخشی از خروجی این الگوریتم:

```
[45589 30965 50774  8113 43050 44811 53575  2051 52294 24537  3175 37856
 49601 11105 10530  8527 58489 35485 19009  8187 35634 45402 37744  7685
 41927 26685 40002 16326 49621 33647 15208  6406 57252 46740 58012 20877
  5978  4744 54391 29560 25919 13582 41137 50523 46017 26007 58457 54284
  9744 25137  8355 19908 40103 18836 13764 37970 58665 51422 28221  8096
 54703   689 41191 48168  8723 46917 56444 47704  9158 19796 36391 36447
  7012 47139 21703 12785 46703 36146 39628 57642 41989  5502 46436 11780
 46905 21874  5470  8917 37192 39306 37708 57912 19104  1090 41117  5019
```

این اعداد ایندکس عکس هایی است توسط کد بالا به صورت تصادفی انتخاب شده اند که توزیع کلاس ها به صورت یکسان است و برای هر کلاس 100 عکس داریم

```
plt.figure(figsize=(10, 8))
scatter = plt.scatter(x_samples_for_plot[:, 0], x_samples_for_plot[:, 1], c=y_samples_for_plot)
plt.colorbar(scatter)
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.title("2D PCA Projection")
plt.show()
```

ایندکس داده هایی که توسط کد بالا ایجاد شده است درون متغیر indexes_for_plt است.

حال مقدار ویژگی های این 1000 عکس را درون متغیر های بالا می ریزیم تا بتوانیم آن ها رسم کنیم.

سپس یک فیگور جدید می سازیم و داده های این 1000 عکس را که مقدار y آن ها نیز مشخص شده را اسکتر میکنیم. پس از آن یک colorbar برای مشاهده پراکندگی داده ها رسم می کنیم.

سوال دوم - ب

مدل SVM به طور پیش فرض برای برای مسئله های دو کلاسه طراحی شده است.

حال برای مسائل چند کلاسه از دو روش one-vs-one و one-vs-all استفاده می‌شود که در ادامه توضیح می دهیم:

1- one-vs-one

در این روش برای هر جفت کلاس یک مدل svm آموزش می دهیم که تعداد مدل های svm برای این مسئله برابر است با ترکیب 2 از 10 که می شود 45.

روش کار به این صورت است که مدل برای همه جفت کلاس ها آموزش داده می شود و برای پیش بینی هر نمونه ورودی همه مدل ها تصمیم گیری می کنند و یکی از دو کلاس را انتخاب می کنند در نهایت کلاسی که بیشترین تعداد رای را در بین مدل‌ها داشته باشد، به عنوان کلاس نهایی انتخاب می‌شود.

2- One-vs-All

در این روش برای هر کلاس یک مدل svm جدا آموزش داده میشود که آن کلاس را از بقیه کلاس ها جدا می کند. برای این مسئله 10 مدل نیاز داریم.

روش کار این مدل به این صورت است که داده های یک کلاس را از مابقی کلاس ها جدا میکند و این کار را 10 بار برای هر کلاس انجام می دهد.هر مدل برای خودش یک مقدار تصمیم (confidence score) یا احتمال برای تعلق نمونه به کلاس مربوطه‌اش تولید می‌کند. سپس کلاسی که بالاترین مقدار تصمیم را دارد، به عنوان کلاس نهایی انتخاب می‌شود.

روش ovo

برای حل مسئله با این روش یک کلاس با این اسم ایجاد کردیم و توابع fit و predict را پیاده سازی کردیم که در ادامه توضیح خواهیم داد:

برای ساخت یک نمونه از این کلاس باید مقادیر کرنل، مقادیر C و مقادیر گاما را ست کنیم.

```
class OvOClassifier:
    def __init__(self, C_values, kernel_values, gamma_values):
        self.C_values = C_values
        self.kernel_values = kernel_values
        self.gamma_values = gamma_values
        self.model_classifiers = {}
```

```python
def fit(self, X, y):
    for (class_1, class_2) in combinations(np.unique(y), 2):
        inedx = (y == class_1) | (y == class_2)
        X_pair = X[inedx]
        y_pair = np.where(y[inedx] == class_1, 1, -1)

        best_score = 0
        best_model = None

        for kernel in self.kernel_values:
            for C in self.C_values:
                for gamma in self.gamma_values:
                    clf = SVC(C=C, kernel=kernel, gamma=gamma, max_iter=3000,random_state=42)
                    clf.fit(X_pair, y_pair)
                    score = clf.score(X_pair, y_pair)

                    if score > best_score:
                        best_score = score
                        best_model = clf

        print("Score of",(class_1, class_2),"is: ",score)
        print('Best Parameters is: kernel =',best_model.kernel,'C =',best_model.C,'gamma =',best_model.gamma)
        print()
        self.model_classifiers[(class_1, class_2)] = best_model
    print(self.model_classifiers)
```

در تابع fit ما X و y را به عنوان ورودی می گیریم سپس یک حلقه فور برای ساخت svm های دو به دو برای کلاس های درون y میزنیم که تعداد تکرار آن برابر ترکیب 2 از تعداد کلاس ها است.

حال برای جفت کلاس انتخاب شده توسط حلقه فور X هایی را که در آن 2 کلاس هستند را جدا می کنیم و برای مقدار y آنها اگر جزو کلاس اول باشند برابر 1 و اگر جزو کلاس دوم باشند برابر 1- قرار می دهیم.

سپس 3 حلقه فور تو در تو برای هایپرپارامتر های svm ها میزنیم تا بتوانیم بهترین مدل را پیدا کنیم. (برای کاهش زمان آموزش مقدار max_iter را برابر 3000 گذاشتیم)

در نهایت پس از پیدا کردن بهترین مدل svm برای 2 جفت کلاس مقدار score و بهترین هایپرپارامتر را در خروجی نمایش می دهیم که برای مثال برای کلاس 0 و 1 به این صورت است :

```
Score of (0, 1) is:  1.0
Best Parameters is: kernel = poly C = 10 gamma = scale
```

در ادامه مدل را در متغییر model_classifiers ذخیره می کنیم.

و در انتها کل این مدل ها را نمایش می دهیم.

```
def predict(self, X):
    votes = np.zeros((X.shape[0], len(self.model_classifiers)))

    for i, ((class_1, class_2), clf) in enumerate(self.model_classifiers.items()):
        pred = clf.predict(X)
        votes[:, i] = np.where(pred == 1, class_1, class_2)

    y_pred = np.apply_along_axis(lambda x: np.bincount(x.astype(int)).argmax(), axis=1, arr=votes)
    return y_pred
```

تابع predict برای پیش بینی مقادیر تست استفاده می شود تا بتوانیم دقت نهایی مدل اصلی را بسنجیم. برای این کار ابتدا یک ماتریس صفر به ابعاد تعداد نمونه های تست در تعداد مدل های دو به دو ایجاد می کنیم که برابر است با 10000 در 45 که نشان دهنده این است که هر مدل چه جوابی برای آن نمونه پیش بینی کرده است.

یک حلقه فور داریم که به تعداد مدل های 2 به 2 تکرار می شود(45 بار) که درون هر حلقه ابتدا آن مدل برای X های ورودی یک y پیش بینی می کند و مقدار آن را در ستون مربوط به آن مدل می ریزد. این روند تا پر شدن ماتریس ادامه پیدا می کند.

حال برای اینکه یک جواب برای هر x ورودی بدهیم باید برای آن x از مدل ها رای گیری کنیم تا بفهمیم این x متعلق به کدام کلاس است. برای مثال اگر x را 5 دهیم. از همه مدل ها می پرسیم که مقدار y این x برابر چیست ؟؟برای 5 در هر 45 مدل یک جواب ارائه می شود که بیشترین تکرار این جواب ها برابر است با جواب نهایی.

```
C_values = [0.1, 1, 10]
kernel_values = ['poly','linear','rbf']
gamma_values = ['scale', 'auto']

ovo_model = OvOClassifier(C_values, kernel_values, gamma_values)
ovo_model.fit(X_train_scaled, y_train)
y_pred = ovo_model.predict(X_test_scaled)
```

پس از ساخت کلاس نوبت به ساخت نمونه و فراخوانی تابع هایی که در بالا توضیح دادیم می رسد. اول مقادیر هایپر پارامتر ها را مشخص می کنیم که برای اجرای سریع تر کد از 3 c و 3 کرنل استفاده کردیم.

حال یک مدل از کلاس مربوطه با مقادیر هایپرپارمتر ها ایجاد می کنیم. سپس تابع fit را برای ساخت مدل های 2 به 2 با X هایی که نرمال سازی کردیم و y صدا می زنیم.

پس از آموزش مدل های کوچک برای x های تست جواب را پیش بینی میکنیم.

```
accuracy = accuracy_score(y_test, y_pred)
classification_report_output = classification_report(y_test, y_pred)
confusion_matrix = confusion_matrix(y_test, y_pred)
end_time = time.time()

print('\nTraining time:',(end_time-start_time)/60/60,"hours/n")
print("\nTest accuracy:", accuracy)
print("\nClassification Report:\n", classification_report_output)
print("\nConfusionReport:\n",confusion_matrix)
```

حال نوبت به نمایش معیار های ارزیابی می رسد که با 60000 داده آموزشی و max_iter 3000 با این هایپرپارامتر ها برابر است با:

```
Training time: 4.370783057543966 hours/n

Test accuracy: 0.9729

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.95      0.98      0.97      1032
           3       0.97      0.98      0.98      1010
           4       0.97      0.96      0.97       982
           5       0.97      0.97      0.97       892
           6       0.98      0.97      0.98       958
           7       0.96      0.97      0.96      1028
           8       0.97      0.97      0.97       974
           9       0.98      0.95      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000


ConfusionReport:
 [[ 966    0    1    3    1    2    4    1    2    0]
 [   0 1128    2    0    0    1    2    1    1    0]
 [   3    0 1011    1    1    0    2    6    7    1]
 [   0    0    5  988    2    5    1    4    5    0]
 [   0    0   14    0  946    0    4    6    2   10]
 [   2    1    3    7    2  865    5    0    5    2]
 [   5    2    2    0    5    8  932    1    3    0]
 [   0    6   11    3    2    3    0  995    0    8]
 [   2    0    8    6    3    6    0    6  940    3]
 [   3    3    4    8   11    2    0   15    5  958]]
```

دقت مدل برای داده های تست برابر است با 97.2 درصد

در عکس پایین مدل های 2 به 2 و هایپرپارامتر های آن ها قابل مشاهده است.

{(0, 1): SVC(C=10, kernel='poly', max_iter=3000, random_state=42), (0, 2): SVC(C=10, gamma='auto', kernel='poly', max_iter=3000, random_state=42), (0, 3): SVC(C=10, kernel='poly', max_iter=3000, random_state=42), (0, 4): SVC(C=10, kernel='poly', max_iter=3000, random_state=42), (0, 5): SVC(C=10, max_iter=3000, random_state=42), (0, 6): SVC(C=10, max_iter=3000, random_state=42), (0, 7): SVC(C=10, gamma='auto', kernel='poly', max_iter=3000, random_state=42), (0, 8): SVC(C=10, kernel='poly', max_iter=3000, random_state=42), (0, 9): SVC(C=1, kernel='linear', max_iter=3000, random_state=42), (1, 2): SVC(C=10, kernel='poly', max_iter=3000, random_state=42), (1, 3): SVC(C=10, kernel='poly', max_iter=3000, random_state=42), (1, 4): SVC(C=10, kernel='poly', max_iter=3000, random_state=42), (1, 5): SVC(C=10, kernel='poly', max_iter=3000, random_state=42), (1, 6): SVC(C=0.1, kernel='linear', max_iter=3000, random_state=42), (1, 7): SVC(C=1, kernel='linear', max_iter=3000, random_state=42), (1, 8): SVC(C=10, kernel='poly', max_iter=3000, random_state=42), (1, 9): SVC(C=10, kernel='poly', max_iter=3000, random_state=42), (2, 3): SVC(C=10, gamma='auto', max_iter=3000, random_state=42), (2, 4): SVC(C=10, max_iter=3000, random_state=42), (2, 5): SVC(C=10, max_iter=3000, random_state=42), (2, 6): SVC(C=10, gamma='auto', kernel='poly', max_iter=3000, random_state=42), (2, 7): SVC(C=10, gamma='auto', max_iter=3000, random_state=42), (2, 8): SVC(C=10, max_iter=3000, random_state=42), (2, 9): SVC(C=10, gamma='auto', max_iter=3000, random_state=42), (3, 4): SVC(C=10, gamma='auto', kernel='poly', max_iter=3000, random_state=42), (3, 5): SVC(C=10, max_iter=3000, random_state=42), (3, 6): SVC(C=10, gamma='auto', kernel='poly', max_iter=3000, random_state=42), (3, 7): SVC(C=10, gamma='auto', max_iter=3000, random_state=42), (3, 8): SVC(C=10, max_iter=3000, random_state=42), (3, 9): SVC(C=10, max_iter=3000, random_state=42), (4, 5): SVC(C=10, kernel='poly', max_iter=3000, random_state=42), (4, 6): SVC(C=1, kernel='linear', max_iter=3000, random_state=42), (4, 7): SVC(C=10, max_iter=3000, random_state=42), (4, 8): SVC(C=10, kernel='poly', max_iter=3000, random_state=42), (4, 9): SVC(C=10, max_iter=3000, random_state=42), (5, 6): SVC(C=10, max_iter=3000, random_state=42), (5, 7): SVC(C=10, kernel='poly', max_iter=3000, random_state=42), (5, 8): SVC(C=10, kernel='poly', max_iter=3000, random_state=42), (5, 9): SVC(C=10, kernel='poly', max_iter=3000, random_state=42), (6, 7): SVC(C=10, gamma='auto', kernel='poly', max_iter=3000, random_state=42), (6, 8): SVC(C=10, max_iter=3000, random_state=42), (6, 9): SVC(C=0.1, kernel='linear', max_iter=3000, random_state=42), (7, 8): SVC(C=10, max_iter=3000, random_state=42), (7, 9): SVC(C=10, gamma='auto', max_iter=3000, random_state=42), (8, 9): SVC(C=10, max_iter=3000, random_state=42)}

دقت و هایپر پارامتر های چند تا از این کلاس ها

```
Score of (0, 2) is:  1.0
Best Parameters is: kernel = poly C = 10 gamma = auto
```

```
Score of (1, 7) is:  0.9996155916045206
Best Parameters is: kernel = linear C = 1 gamma = scale
```

```
Score of (2, 7) is:  0.9999181870244621
Best Parameters is: kernel = rbf C = 10 gamma = auto
```

```
Score of (3, 5) is:  0.9995671745152355
Best Parameters is: kernel = rbf C = 10 gamma = scale
```

```
Score of (4, 6) is:  0.9997448979591836
Best Parameters is: kernel = linear C = 1 gamma = scale
```

روش ova

برای حل مسئله با این روش ابتدا داده ها را شافل می کنیم.
سپس برای بهبود سرعت به جای کل داده ها از 6000 تا استفاده می کنیم.

```
X_train_scaled, y_train = shuffle(X_train_scaled, y_train, random_state=42)
```

```
x_samples_for_train = []
y_samples_for_train = []
for i in np.unique(y_train):
    indexes = np.where(y_train == i)[0]
    selected_indexes = np.random.choice(indexes, 600, replace=False)
    x_samples_for_train.append(X_train_scaled[selected_indexes])
    y_samples_for_train.append(y_train[selected_indexes])

x_samples_for_train = np.vstack(x_samples_for_train)
y_samples_for_train = np.hstack(y_samples_for_train)
```

در این یک حلقه فور زدیم که به تکرار تعداد کلاس ها است. ابتدا ایندکس هایی که y آن برابر همان کلاس است را انتخاب می کنیم. برای مثال در شروع حلقه، i برابر 0 صفر است و ایندکس داده هایی که y آن ها صفر است برگردانده میشود. سپس با استفاده از نامپای 600 تا از این ایندکس ها را به صورت رندوم انتخاب می شود و پس از آن x ها و y هایی با آن ایندکس به لیست مربوطه اضافه می شود. این کار تا کلاس 9 تکرار می شود که در نهایت 6000 داده با توزیع یکسان داریم.

```
class OvAClassifier:
    def __init__(self, C_values, kernel_values, gamma_values):
        self.C_values = C_values
        self.kernel_values = kernel_values
        self.gamma_values = gamma_values
        self.classifiers = {}
```

همانند مدل قبل یک کلاس برای این مدل هم ایجاد می کنیم که با ورودی 3 هایپر پارامتر ساخته می شود.

```
def fit(self, X, y):
    classes = np.unique(y)
    for cls in classes:
        y_class = np.where(y == cls, 1, -1)
        ova_best_score = 0
        ova_best_model = None

        for kernel in self.kernel_values:
            for C in self.C_values:
                for gamma in self.gamma_values:
                    ova_model = SVC(C=C, kernel=kernel, gamma=gamma, probability=True)
                    ova_model.fit(X, y_class)
                    score = ova_model.score(X, y_class)
                    if score > ova_best_score:
                        ova_best_score = score
                        ova_best_model = ova_model
        print("Score of",cls,"is: ",ova_best_score)
        print('Best Parameters is: kernel =',ova_best_model.kernel,'C =',ova_best_model.C,'gamma =',ova_best_model.gamma)
        print()
        self.classifiers[cls] = ova_best_model
```

تابع fit همانند مدل ovo است با این تفاوت که حلقه فور به تعداد کلاس ها اجرا می شود و y داده های هر کلاس از بقیه کلاس ها جدا می شود. برای مثال ابتدا cls برابر 0 است و سمپل هایی که y آن ها برابر 0 است به 1 و بقیه کلاس ها به 1- تبدیل می شوند.
سپس در حلقه های تو در تو بهترین هایپر پارامتر انتخاب می شود و بهترین مدل در متغیر مربوطه ذخیره می شود.

```
def predict(self, X):
    scores = np.zeros((X.shape[0], len(self.classifiers)))

    for class_label, clf in self.classifiers.items():
        scores[:, class_label] = clf.predict_proba(X)[:, 1]

    y_pred = np.argmax(scores, axis=1)
    return y_pred
```

اینجا هم یک ماتریس به ابعاد تعداد x های ورودی و تعداد مدل ها داریم که برابر است با (10000,10). در ادامه یک فور به تعداد کلاس ها داریم. در روش ova ما احتمال تعلق به کلاس ها را محاسبه می کنیم و در ماتریس می ریزیم.

در نهایت در متغیر y_pred پیشبینی هایی که بیشترین احتمال را دارند پیدا می کنیم.

```
C_values = [0.1,1,10]
kernel_values = ['rbf' , 'poly' , 'linear']
gamma_values = ['scale', 'auto']

ova_model = OvAClassifier(C_values, kernel_values, gamma_values)
ova_model.fit(x_samples_for_train, y_samples_for_train)
y_pred = ova_model.predict(X_test_scaled)
```

حال یک مدل از کلاس ova با مقادیر هایپرپارمتر ها ایجاد می کنیم. سپس تابع fit را برای ساخت مدل های 0 تا 9 صدا می زنیم

پس از آموزش مدل های کوچک برای x های تست جواب را پیش بینی میکنیم.

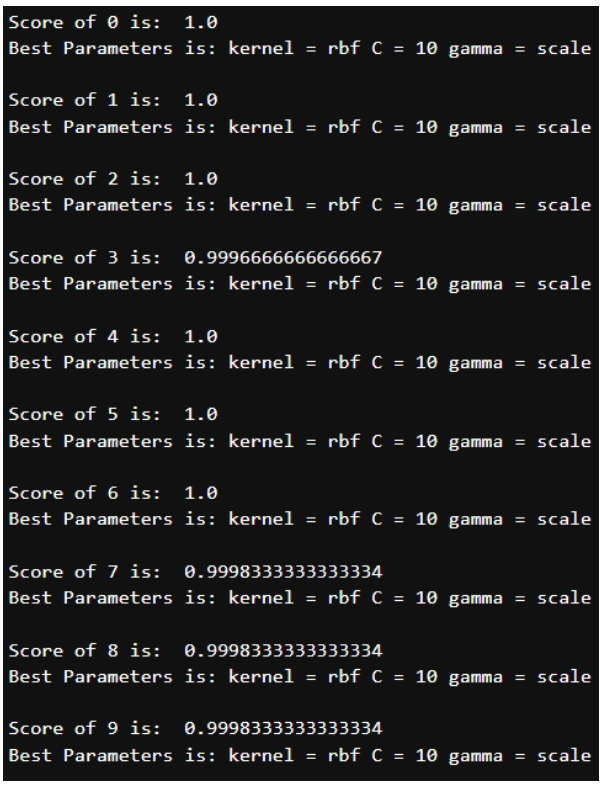
```
ova_model_accuracy = accuracy_score(y_test, y_pred)
ova_model_classification_report = classification_report(y_test, y_pred)
ova_model_confusion_matrix = confusion_matrix(y_test, y_pred)
ova_model_end_time = time.time()

print('\nTraining time:',(ova_model_end_time - ova_model_start_time)/60/60,"hours/n")
print("\nTest accuracy:", ova_model_accuracy)
print("\nClassification Report:\n", ova_model_classification_report)
print("\nConfusionReport:\n",ova_model_confusion_matrix)
```

در نهایت باید معیار های ارزیابی را چاپ کنیم:

میزان دقت مدل 94.7 درصد

```
Training time: 0.9195588476128048 hours/n

Test accuracy: 0.9471

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       980
           1       0.98      0.99      0.98      1135
           2       0.94      0.94      0.94      1032
           3       0.95      0.94      0.95      1010
           4       0.94      0.95      0.94       982
           5       0.95      0.94      0.94       892
           6       0.96      0.96      0.96       958
           7       0.88      0.94      0.91      1028
           8       0.95      0.93      0.94       974
           9       0.96      0.91      0.93      1009

    accuracy                           0.95     10000
   macro avg       0.95      0.95      0.95     10000
weighted avg       0.95      0.95      0.95     10000
```

در اینجا هم میزان دقت مدل های کوچکتر و هایپر پارامتر های آن قابل مشاهده است.



تاثیر عدم توازن داده ها:

مدل svm برای حاشیه بهینه ممکن است به جای اینکه به کلاس هدف توجه بیشتری کند داده ها را بر اساس کلاسی که داده بیشتری دارد جدا کند.

در مسائل با عدم توازن، مدل ممکن است به خاطر حجم زیاد داده‌های کلاس‌های غیرهدف نتواند به خوبی ویژگی‌های کلاس هدف را یاد بگیرد و دقت مدل کاهش یابد.

روش‌های رفع عدم توازن

1. وزن دهی به کلاس هدف
2. افزایش نمونه‌های کلاس هدف با Oversampling
3. استفاده از تکنیک‌های زیرنمونه‌گیری (Undersampling)
4. و ...

اما در این روش از آنجایی که همه کلاس های ما unbalance هستند، می توانیم از تکنیک های بالا استفاده نکنیم.

روش Crammer singer

```
X_train_scaled, y_train = shuffle(X_train_scaled, y_train, random_state=42)

C_values = [0.01, 0.1, 1, 10]
best_accuracy = 0
best_C = None
best_model = None
```

ابتدا داده ها را شافل می کنیم و چند مقدار برای c ست می کنیم.

```
for C in C_values:
    print(f"\nTraining with C={C}...")
    model = LinearSVC(multi_class='crammer_singer', C=C, max_iter=1000,dual='auto')
    model.fit(X_train_scaled, y_train)

    y_pred = model.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy for C={C}: {accuracy}")

    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_C = C
        best_model = model
```

درون حلقه فور یک مدل svm با روش crammer singer می سازیم. (با max_iter = 1000)

سپس مدلمان را فیت می کنیم و برای مقدار x که قبلا نرمالسازی کردیم، پیش بینی میکنیم.

سپس میزان دقت مدل با c مربوطه را چاپ می کنیم و در ادامه بهترین مدل را پیدا می کنیم.

```
print(f"\nBest C value: {best_C} with test accuracy: {best_accuracy}")

y_pred = best_model.predict(X_test_scaled)
crammer_model_accuracy = accuracy_score(y_test, y_pred)
crammer_model_classification_report = classification_report(y_test, y_pred)
crammer_model_confusion_matrix = confusion_matrix(y_test, y_pred)
crammer_model_end_time = time.time()

print('\nTraining time:',(crammer_model_end_time - crammer_model_start_time)/60/60,"hours\n")
print("\nTest accuracy:", crammer_model_accuracy)
print("\nClassification Report:\n", crammer_model_classification_report)
print("\nConfusionReport:\n",crammer_model_confusion_matrix)
```

سپس بهترین مدل و c آن را چاپ می کنیم. در انتها میزان دقت و معیار های ارزیابی را چاپ می کنیم که به شکل زیر است:

```
Test accuracy: 0.9276

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       980
           1       0.96      0.98      0.97      1135
           2       0.93      0.90      0.92      1032
           3       0.92      0.91      0.91      1010
           4       0.93      0.93      0.93       982
           5       0.90      0.87      0.89       892
           6       0.95      0.95      0.95       958
           7       0.93      0.93      0.93      1028
           8       0.89      0.89      0.89       974
           9       0.92      0.92      0.92      1009

    accuracy                           0.93     10000
   macro avg       0.93      0.93      0.93     10000
weighted avg       0.93      0.93      0.93     10000
```

```
ConfusionReport:
 [[ 963    0    0    1    2    4    3    4    2    1]
 [   0 1117    4    3    0    1    3    2    5    0]
 [   8   11  929   13   12    4    9    9   34    3]
 [   5    0   19  920    4   25    1    8   20    8]
 [   2    3    6    1  915    0    9    7    7   32]
 [   7    3    1   33    9  780   15   10   29    5]
 [  12    4    8    1    6   16  907    2    2    0]
 [   0   11   23    4    7    2    0  952    1   28]
 [   8   10    7   18   10   26    9   11  866    9]
 [   8    7    1   10   22    9    0   17    8  927]]
```

میزان دقت : 92.7 درصد

میزان دقت با C های مختلف:

```
Training with C=0.01...
Accuracy for C=0.01: 0.9276

Training with C=0.1...
Accuracy for C=0.1: 0.9266

Training with C=1...
Accuracy for C=1: 0.9178

Training with C=10...
Accuracy for C=10: 0.6385

Best C value: 0.01 with test accuracy: 0.9276

Training time: 2.060320395761066 hours
```

برای نمایش تعدادی از عکس ها از یک حلقه فور استفاده کردیم که عکس ها را با استفاده از کتابخانه matplotlib نمایش میدهد.

درون حلقه فور 10 بار ایندکس رندوم تولید می کند و سپس مقدار ویژگی های آن عکس را نمایش میدهد.

```
fig, pics = plt.subplots(2,5,figsize=(11,11))
for i, pic in enumerate(pics.flat):
    index = np.random.choice(len(X_train),1)
    index = index[0]
    pic.imshow(X_train[index], cmap='gray')
    pic.axis('off')

plt.tight_layout()
plt.show()
```