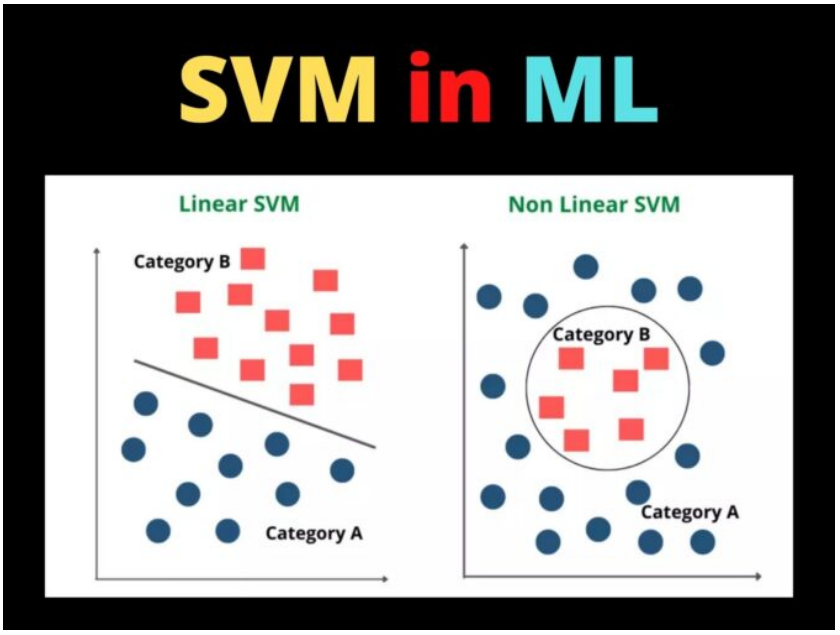
PRÁCTICA Nº0304

SVM



- Módulo: PIA
- Nombre y apellidos: Alvaro Lucio-Villegas de Cea

Índice

Teoría

Definición

Dado un conjunto de puntos, subconjunto de un conjunto mayor (espacio), en el que cada uno de ellos pertenece a una de dos posibles categorías, un algoritmo basado en SVM construye un modelo capaz de predecir si un punto nuevo (cuya categoría desconocemos) pertenece a una categoría o a la otra.

Dentro del SVM existen tres hiperparametros que nos ayudan a agudizar más nuestro análisis estos son:

- Kernel
- Margin
- C
- Gamma

Tipos de Kernel

Existen distintos tipos de kernel que se pueden aplicar pero nos vamos a centrar en:

- Linear

  El kernel lineal se usa cuando los datos son linealmente separables. Significa que los datos se pueden separar usando una sola línea. Es uno de los núcleos más comunes que se utilizan. Se usa principalmente cuando hay una gran cantidad de características en un conjunto de datos.
  El kernel lineal se usa a menudo para fines de clasificación de texto.

  Linear kernel : K(xi , xj) = xiT xj

- Polynomial

  El núcleo polinomial representa la similitud de los vectores (muestras de entrenamiento) en un espacio de características sobre polinomios de las variables originales.

  El núcleo polinómico no solo analiza las características dadas de las muestras de entrada para determinar su similitud, sino también las combinaciones de las muestras de entrada.

  Polynomial kernel : K(xi , xj) = (γxiT xj + r)d , γ > 0

- Radial Basis Function (RBF)

  El kernel de función de base radial es un kernel de propósito general. Se utiliza cuando no tenemos conocimiento previo sobre los datos.

$$k(x,y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$$

- Sigmoide

  El kernel sigmoide tiene su origen en las redes neuronales. Podemos usarlo como proxy para redes neuronales. El núcleo sigmoide está dado por la siguiente ecuación:

  Núcleo sigmoideo: k (x, y) = tanh(αxTy + c

Caso Práctico

Vamos a utilizar un data caso práctico de la plataforma de Kaggle sobre el precio de la movilización . Dejo aquí el enlace al caso.

Lo primero que se debe de hacer es la limpieza de los datos y la división del dataset

Se da la casualidad que este conjunto de datos ya está limpio y no existen valores nulos.

```
# checking if there is any missing value
df.isnull().sum().max()
# df.columns
```

```
0
```

Realizamos la división de dataframe para entrenamiento y para test en este caso vamos a hacer un 20% para comprobar la fiabilidad y un 80 % para su entrenamiento.

```
X_train,X_test,Y_train,Y_test = train_test_split(X_t,y_t,test_size=.20,random_state=42)
print("shape of X Train :"+str(X_train.shape))
print("shape of X Test :"+str(X_test.shape))
print("shape of Y Train :"+str(Y_train.shape))
print("shape of Y Test :"+str(Y_test.shape))
```

Entrenamiento del modelo.

SVM Linear

Vamos a realizar varias pruebas con cada posibilidad de uso de SVC.

En el primero vamos a usar el kernel “Linear” y un bucle con distintas posibilidades de “C”

```
for this_C in [1,3,5,10,40,60,80,100]:
    clf = SVC(kernel='linear',C=this_C).fit(X_train,Y_train)
    scoretrain = clf.score(X_train,Y_train)
    scoretest  = clf.score(X_test,Y_test)
    print("Linear SVM value of C:{}, training score :{:2f} , Test Score: {:2f} \n".format(this_C,scoretrain,scoretest))
```

```
Linear SVM value of C:1, training score :0.953750 , Test Score: 0.960000

Linear SVM value of C:3, training score :0.961875 , Test Score: 0.977500

Linear SVM value of C:5, training score :0.968125 , Test Score: 0.975000

Linear SVM value of C:10, training score :0.977500 , Test Score: 0.967500

Linear SVM value of C:40, training score :0.981250 , Test Score: 0.962500

Linear SVM value of C:60, training score :0.981250 , Test Score: 0.962500

Linear SVM value of C:80, training score :0.981875 , Test Score: 0.970000

Linear SVM value of C:100, training score :0.980625 , Test Score: 0.967500
```

Para el siguiente entrenamiento vamos a aplicar una validación cruzada y dejamos fijo el valor de “C”.

```
from sklearn.model_selection import cross_val_score,StratifiedKFold,LeaveOneOut
clf1 = SVC(kernel='linear',C=20).fit(X_train,Y_train)
scores = cross_val_score(clf1,X_train,Y_train,cv=5)
strat_scores = cross_val_score(clf1,X_train,Y_train,cv=StratifiedKFold(5,random_state=10,shuffle=True))
#Loo = LeaveOneOut()
#Loo_scores = cross_val_score(clf1,X_train,Y_train,cv=Loo)
print("The Cross Validation Score :"+str(scores))
print("The Average Cross Validation Score :"+str(scores.mean()))
print("The Stratified Cross Validation Score :"+str(strat_scores))
print("The Average Stratified Cross Validation Score :"+str(strat_scores.mean()))
#print("The LeaveOneOut Cross Validation Score :"+str(Loo_scores))
#print("The Average LeaveOneOut Cross Validation Score :"+str(Loo_scores.mean()))
```

```
The Cross Validation Score :[0.95015576 0.96261682 0.94392523 0.92789969 0.97169811]
The Average Cross Validation Score :0.9512591238085129
The Stratified Cross Validation Score :[0.95327103 0.96884735 0.95015576 0.96551724 0.95597484]
The Average Stratified Cross Validation Score :0.9587532454897574
```

En el siguiente ejemplo vamos a usar una característica de Sklearn que se llama “DummyClassifier” esto realiza la opción de realizar predicciones que ignoran las características de entrada.

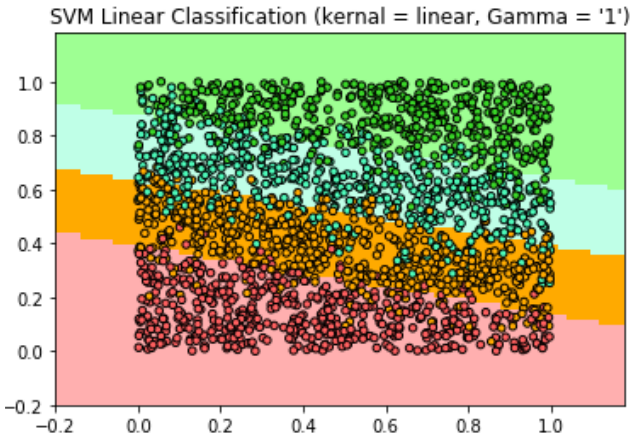
```
from sklearn.dummy import DummyClassifier

for strat in ['stratified', 'most_frequent', 'prior', 'uniform']:
    dummy_maj = DummyClassifier(strategy=strat).fit(X_train,Y_train)
    print("Train Stratergy :{} \n Score :{:.2f}".format(strat,dummy_maj.score(X_train,Y_train)))
    print("Test Stratergy :{} \n Score :{:.2f}".format(strat,dummy_maj.score(X_test,Y_test)))
```

```
Train Stratergy :stratified
 Score :0.24
Test Stratergy :stratified
 Score :0.25
Train Stratergy :most_frequent
 Score :0.26
Test Stratergy :most_frequent
 Score :0.23
Train Stratergy :prior
 Score :0.26
Test Stratergy :prior
 Score :0.23
Train Stratergy :uniform
 Score :0.25
Test Stratergy :uniform
 Score :0.26
```

En el siguiente ejemplo de SVC vamos a aplicarlo a un grafo y aplicando el kernel “Linear” y valor de “C” a 1 esto significa que no tiene vecinos a su alrededor.

```
h = .02  # step size in the mesh
C_param = 1 # No of neighbours
for weights in ['uniform', 'distance']:
    # we create an instance of Neighbours Classifier and fit the data.
    clf1 = SVC(kernel='linear',C=C_param)
    clf1.fit(X, y)

    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    x_min, x_max = X[:, 0].min()-.20, X[:, 0].max()+.20
    y_min, y_max = X[:, 1].min()-.20, X[:, 1].max()+.20
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf1.predict(np.c_[xx.ravel(), yy.ravel()])   # ravel to flatten the into 1D and c_ to concatenate

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cm_bright)

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cm_dark,
                edgecolor='k', s=20)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title("SVM Linear Classification (kernal = linear, Gamma = '%s')"% (C_param))

plt.show()
```

Podemos ver como realiza la separación de de grupos de forma lineal pero también podemos observar como existen algunos puntos que no están es sus respectivos grupos esto se podría mejorar por el valor de margin de SVC.



En el siguiente caso vamos a cambiar a una versión de SVC que solo usa la variable C en su uso.Esta se llama “LinearSVC”.

```
# Linear Support vector machine with only C Parameter
from sklearn.svm import LinearSVC

for this_C in [1,3,5,10,40,60,80,100]:
    clf2 = LinearSVC(C=this_C).fit(X_train,Y_train)
    scoretrain = clf2.score(X_train,Y_train)
    scoretest  = clf2.score(X_test,Y_test)
    print("Linear SVM value of C:{}, training score :{:2f} , Test Score: {:2f} \n".format(this_C,scoretrain,scoretest))
```

```
Linear SVM value of C:1, training score :0.846250 , Test Score: 0.840000

Linear SVM value of C:3, training score :0.864375 , Test Score: 0.855000

Linear SVM value of C:5, training score :0.868125 , Test Score: 0.870000

Linear SVM value of C:10, training score :0.873750 , Test Score: 0.872500

Linear SVM value of C:40, training score :0.856875 , Test Score: 0.847500

Linear SVM value of C:60, training score :0.761250 , Test Score: 0.770000

Linear SVM value of C:80, training score :0.838125 , Test Score: 0.825000

Linear SVM value of C:100, training score :0.843125 , Test Score: 0.845000
```

+ Code + Markdown

Apparently we got better scores with SVC where we defined the kernal as linear than with just LinearSVC

Estos resultados son peores que con el SVC con el kernel en modo lineal.Pero la gran diferencia entre estos dos tipos en que LinearSVC es mucho mas rapido a SVC

```
from sklearn.svm import SVR

svr = SVR(kernel='linear',C=1,epsilon=.01).fit(X_train,Y_train)
print("{:.2f} is the accuracy of the SV Regressor".format(svr.score(X_train,Y_train)))
```

```
0.92 is the accuracy of the SV Regressor
```

- SVM admite regresión lineal y no lineal.
- La regresión SVM intenta ajustar tantas instancias como sea posible en el límite de decisión mientras limita las violaciones de margen.
- El ancho del límite de decisión está controlado por un hiper parámetro **ϵ**.

SVM No Lineal

Un método para manejar relaciones no lineales en nuestro conjunto de datos es usar Kernal polinomial o usar una función de similitud con nuestra SVM.

Usaremos la función de función de base radial gaussiana (RBF) para lo mismo. para manejar esto en Sklearn hay un hiperparámetro Gamma. Consulte la función Gausian RBF para obtener más información.

Técnicamente, el parámetro gamma es el inverso de la desviación estándar del kernel RBF (función gaussiana), que se utiliza como medida de similitud entre dos puntos. Intuitivamente, un valor gamma pequeño define una función gaussiana con una varianza grande. En este caso, dos puntos se pueden considerar similares aunque estén lejos el uno del otro.

```
# SMV with RBF KERNAL AND ONLY C PARAMETER

for this_C in [1,5,10,25,50,100]:
    clf3 = SVC(kernel='rbf',C=this_C).fit(X_train,Y_train)
    clf3train = clf3.score(X_train,Y_train)
    clf3test  = clf3.score(X_test,Y_test)
    print("SVM for Non Linear \n C:{} Training Score : {:2f} Test Score : {:2f}\n".format(this_C,clf3train,clf3test))
```

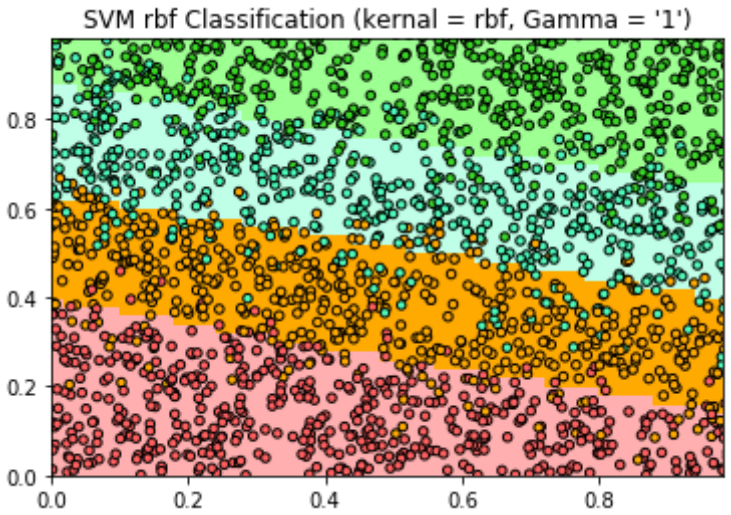
```
SVM for Non Linear
 C:1 Training Score : 0.902500 Test Score : 0.887500

SVM for Non Linear
 C:5 Training Score : 0.957500 Test Score : 0.927500

SVM for Non Linear
 C:10 Training Score : 0.963750 Test Score : 0.927500

SVM for Non Linear
 C:25 Training Score : 0.979375 Test Score : 0.927500

SVM for Non Linear
 C:50 Training Score : 0.986250 Test Score : 0.925000

SVM for Non Linear
 C:100 Training Score : 0.993125 Test Score : 0.920000
```

Por otro lado, un valor gamma grande significa definir una función gaussiana con una varianza pequeña y en este caso, dos puntos se consideran similares solo si están cerca uno del otro.

Ahora vamos a aplicarlo a un un grajo para poder visualizar mejor los datos.

```
h = .02  # step size in the mesh
C_param = 1 # No of neighbours
for weights in ['uniform', 'distance']:
    # we create an instance of Neighbours Classifier and fit the data.
    clf1 = SVC(kernel='rbf',C=C_param)
    clf1.fit(X, y)

    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    x_min, x_max = X[:, 0].min(), X[:, 0].max()
    y_min, y_max = X[:, 1].min(), X[:, 1].max()
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf1.predict(np.c_[xx.ravel(), yy.ravel()])   # ravel to flatten the into 1D and c_ to concatenate

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cm_bright)

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cm_dark,
                edgecolor='k', s=20)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title("SVM Linear Classification (kernal = linear, Gamma = '%s')"% (C_param))

plt.show()
```

Como podemos ver se sigue diferenciando los 4 distintos grupos y que en gran parte de ellos se mezcla bastante con otros grupos.



En el siguiente caso vamos a hacer uso de la opción de “gamma” de SVC. También se van a realizar varias combinaciones de valores en cada opción.

```
# SVM WITH RBF KERNAL, C AND GAMMA HYPERPARAMTER
for this_gamma in [.1,.5,.10,.25,.50,1]:
    for this_C in [1,5,7,10,15,25,50]:
        clf3 = SVC(kernel='rbf',C=this_C,gamma=this_gamma).fit(X_train,Y_train)
        clf3train = clf3.score(X_train,Y_train)
        clf3test  = clf3.score(X_test,Y_test)
        print("SVM for Non Linear \n Gamma: {} C:{} Training Score : {:2f} Test Score : {:2f}\n".format(this_gamma,this_C,clf3train,clf3test))
```

```
SVM for Non Linear
 Gamma: 0.5 C:1 Training Score : 0.980625 Test Score : 0.835000

SVM for Non Linear
 Gamma: 0.5 C:5 Training Score : 1.000000 Test Score : 0.850000

SVM for Non Linear
 Gamma: 0.5 C:7 Training Score : 1.000000 Test Score : 0.847500

SVM for Non Linear
 Gamma: 0.5 C:10 Training Score : 1.000000 Test Score : 0.847500

SVM for Non Linear
 Gamma: 0.5 C:15 Training Score : 1.000000 Test Score : 0.847500

SVM for Non Linear
 Gamma: 0.5 C:25 Training Score : 1.000000 Test Score : 0.847500

SVM for Non Linear
 Gamma: 0.5 C:50 Training Score : 1.000000 Test Score : 0.847500

SVM for Non Linear
 Gamma: 1 C:1 Training Score : 0.993125 Test Score : 0.712500

SVM for Non Linear
 Gamma: 1 C:5 Training Score : 1.000000 Test Score : 0.742500

SVM for Non Linear
 Gamma: 1 C:7 Training Score : 1.000000 Test Score : 0.742500

SVM for Non Linear
 Gamma: 1 C:10 Training Score : 1.000000 Test Score : 0.742500

SVM for Non Linear
 Gamma: 1 C:15 Training Score : 1.000000 Test Score : 0.742500

SVM for Non Linear
 Gamma: 1 C:25 Training Score : 1.000000 Test Score : 0.742500

SVM for Non Linear
```

A la hora de buscar la mejor opción de valores tenemos una funcion llamada “GridSearchCV” que realiza la búsqueda exhaustiva sobre valores de parámetros específicos para un estimador y realiza los parámetros del estimador utilizados para aplicar estos métodos se optimizan que mediante una búsqueda de cuadrícula con validación cruzada sobre una cuadrícula de parámetros.

```
# grid search method
from sklearn.model_selection import GridSearchCV

param_grid = {'C': [1,5,7,10,15,25,50],
              'gamma': [.1,.5,.10,.25,.50,1]}
GS = GridSearchCV(SVC(kernel='rbf'),param_grid,cv=5)
```

```
GS.fit(X_train,Y_train)
```

```
GridSearchCV(cv=5, error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'C': [1, 5, 7, 10, 15, 25, 50], 'gamma': [0.1, 0.5, 0.1, 0.25, 0.5, 1]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

+ Code + Markdown

```
print("the parameters {} are the best.".format(GS.best_params_))
print("the best score is {:.2f}.".format(GS.best_score_))
```

```
the parameters {'C': 7, 'gamma': 0.1} are the best.
the best score is 0.91.
```

Máquina SVM kernelizada

Esta opción es la más difícil de realizar ya que se requiere de conocimientos matemáticos para poder realizarla correctamente pero también es la que es capaz de sacar mejores resultados.

Ya que usa opciones como el grado o el epsilon.

```
# Kernalized SVM machine
svr2 = SVR(degree=2,C=100,epsilon=.01).fit(X_train,Y_train)
print("{:.2f} is the accuracy of the SV Regressor".format(svr2.score(X_train,Y_train)))
```

```
0.95 is the accuracy of the SV Regressor
```

También para poder hacer este tipo de opción es necesario tener ciertos conocimientos de los datos y para ello es necesario realizar varios métodos antes.Se recomienda el uso de la validación cruzada para tener mejor conocimiento de los mismos.

Conclusión

El uso de estas herramientas u opciones, brindan una gran variedad de oportunidades de precisión para el entrenamiento de nuestro modelo, para ello es necesario tener gran conocimiento sobre los datos para poder darle un buen uso.