Suport Vector Machine (SVM)

¿Qué aprenderemos hoy?

- **SVM kernel lineal**
- **SVM kernel no lineal**

Support Vector Machine

Con **SVM** nuestro objetivo de optimización es **maximizar el margen**. El margen lo podemos definir como la **distancia** entre el hiperplano de separación (frontera de decisión) y las muestras de entrenamiento más cercanas a este hiperplano, que son los llamados **Vectores de Soporte**.

In [ ]:
```
from IPython.display import Image

Image(filename='Imagenes_Clase_05/5_3.png', width=700)
```

Para tener una idea de la maximización de los márgenes, veamos con más detalle los **hiperplanos positivos y negativos** que son paralelos a la frontera de decisión:

$$w_0 + w^T x_{pos} = 1 \quad (1)$$
$$w_0 + w^T x_{neg} = -1 \quad (2)$$

Restando (1) y (2):

$$w^T \left(x_{pos} - x_{neg}\right) = 2$$

Podemos normalizar dividiendo por la norma del vector w, $\|w\| = \sqrt{\sum_{j=1}^{m} w_j^2}$:

$$\frac{w^T \left(x_{pos} - x_{neg}\right)}{\|w\|} = \frac{2}{\|w\|}$$

El lado izquierdo de la ecuación anterior puede interpretarse como la distancia entre el hiperplano positivo y el negativo, y es el llamado margen que queremos maximizar.

La función objetivo de **SVM** se convierte en la maximización de $\frac{2}{\|w\|}$ bajo la restricción de que las muestras sean clasificadas correctamente:

$$w_0 + w^T x^{(i)} \geq 1 \text{ si } y^{(i)} = 1$$
$$w_0 + w^T x^{(i)} \leq -1 \text{ si } y^{(i)} = -1$$
$$\text{Para } i = 1 ... N$$

Estas dos ecuaciones dicen básicamente que todas las muestras negativas deben caer en un lado del hiperplano negativo, mientras que todas las muestras positivas deben caer detrás del hiperplano positivo:

$$y^{(i)} \left(w_0 + w^T x^{(i)}\right) \geq 1 \ \forall i$$

En la práctica, sin embargo, es más fácil minimizar $\frac{1}{2}\|w\|^2$.

Inclusión variable de holgura (clasificación de margen suave):

$$w_0 + w^T x^{(i)} \geq 1 - \xi^{(i)} \text{ si } y^{(i)} = 1$$
$$w_0 + w^T x^{(i)} \leq -1 + \xi^{(i)} \text{ si } y^{(i)} = -1$$
$$\text{Para } i = 1 ... N$$

Formulación final:

$$\min \ \frac{1}{2}\|w\|^2 + C \sum_i \xi^{(i)}$$
$$s.a.$$
$$w_0 + w^T x^{(i)} \geq 1 - \xi^{(i)} \text{ si } y^{(i)} = 1$$
$$w_0 + w^T x^{(i)} \leq -1 + \xi^{(i)} \text{ si } y^{(i)} = -1$$
$$\text{Para } i = 1 ... N$$

A través del parámetro C, podemos controlar la penalización por clasificación errónea. Los valores grandes de C corresponden a grandes penalizaciones por el error, mientras que seremos menos estrictos con los errores de clasificación si elegimos valores más pequeños para C.

In [ ]:
```
Image(filename='Imagenes_Clase_05/5_4.png', width=600)
```

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
```

In [ ]:
```
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data[:, [2, 3]]
name_columns = ['sepal_length','sepal_width','petal_length','petal_width']
variable_names=list(name_columns[2:4])
print(variable_names)
y = iris.target
name_clases=np.unique(y)
print('Etiquetas de Clase:', name_clases)
```

In [ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=1, stratify=y)

print('Cantidad de etiquetas en y:', np.bincount(y))
print('Cantidad de etiquetas en y_train:', np.bincount(y_train))
print('Cantidad de etiquetas en y_test:', np.bincount(y_test))
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

In [ ]:
```
%matplotlib inline
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt


def plot_decision_regions(X, y, classifier, plot_object,test_idx=None,
                          only_train=False, only_test=False,
                          clases_names=['clase 0','clase 1'),
                          resolution=0.02):

    markers = ('s', 'o', '^', 'v', 'x')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)


    plot_object.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)


    if not(only_test):
        for idx, cl in enumerate(np.unique(y)):
            plot_object.scatter(x=X[y == cl, 0],y=X[y == cl, 1],
                                alpha=0.8,c=colors[idx],
                                marker=markers[idx],label=clases_names[idx],
                                edgecolor='black')

    if not(test_idx is None) and not(only_train):
        X_test, y_test = X[test_idx, :], y[test_idx]
        for idx, cl in enumerate(np.unique(y_test)):
            plot_object.scatter(x=X_test[y_test == cl, 0],
                                y=X_test[y_test == cl, 1],
                                alpha=0.8,c=colors[idx],marker=markers[idx],
                                label=clases_names[idx],edgecolor='black')
```

In [ ]:
```
X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))
```

In [ ]:
```
from sklearn.svm import SVC

svm = SVC(kernel='linear', C=100., random_state=1)
svm.fit(X_train_std, y_train)

plot_decision_regions(X_combined_std, y_combined,classifier=svm,
                      plot_object=plt, test_idx=range(105, 150),
                      only_train=True,clases_names=name_clases)
plt.xlabel(f'{variable_names[0]} [cm]')
plt.ylabel(f'{variable_names[1]} [cm]')
plt.title('Linear SVM - sklearn (train set)')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```

In [ ]:
```
plot_decision_regions(X_combined_std, y_combined,classifier=svm,
                      plot_object=plt,test_idx=range(105, 150),
                      only_test=True,clases_names=name_clases)

plt.xlabel(f'{variable_names[0]} [cm]')
plt.ylabel(f'{variable_names[1]} [cm]')
plt.title('Linear SVM - sklearn (Test set)')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```

In [ ]:
```
# Tres modelos
svm.coef_
```

In [ ]:
```
r_exponents=[-6,-4,-1,0,2,4]
for idx, c in enumerate(r_exponents):
    svm = SVC(kernel='linear', C=10.**c, random_state=1)
    svm.fit(X_train_std, y_train)
    if idx%3==0:
        fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(15, 4))
    plot_decision_regions(X_combined_std, y_combined,classifier=svm,
                          plot_object=axs[idx%3],test_idx=range(105, 150),
                          only_train=True,clases_names=name_clases)
    axs[idx%3].set_xlabel(f'{variable_names[0]} [cm]')
    axs[idx%3].set_ylabel(f'{variable_names[1]} [cm]')
    axs[idx%3].set_title('Linear SVM (Train set): C = '+str(10.**c))
    axs[idx%3].legend(loc='upper left')

    if idx%3==2:
        plt.show()
```

SVM Kernel No Lineal

- Para abordar problemas de clasificación no lineal

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(1)
X_xor = np.random.randn(250, 2)
y_xor = np.logical_xor(X_xor[:, 0] > 0,X_xor[:, 1] > 0)
y_xor = np.where(y_xor, 1, -1)
name_clases_xor=np.unique(y_xor)
print('Etiquetas de Clase:', name_clases_xor)
```

In [ ]:
```
plt.scatter(X_xor[y_xor == 1, 0],X_xor[y_xor == 1, 1],c='b',marker='x',label='1')
plt.scatter(X_xor[y_xor == -1, 0],X_xor[y_xor == -1, 1],c='r',marker='s',label='-1')

plt.xlim([-3, 3])
plt.ylim([-3, 3])
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```

La idea básica detrás de los métodos de kernel (para tratar estos datos linealmente no separables) es crear combinaciones no lineales de las características originales para proyectarlas en un espacio de mayor dimensión a través de una función de mapeo ϕ.

$$\phi(x_1, x_2) = (z_1, z_2, z_3) = (x_1, x_2, x_1^2 + x_2^2)$$

In [ ]:
```
from IPython.display import Image

Image(filename='Imagenes_Clase_05/6_1.png', width=700)
```

Para resolver un problema no lineal usando SVM, tenemos que transformar los datos de entrenamiento en un espacio de características de mayor dimensión a través de una función de mapeo ϕ. Luego, podemos encontrar un modelo SVM lineal para clasificar los datos en este nuevo espacio de características. Así, podemos usar la misma función de mapeo ϕ para transformar datos nuevos (no vistos por el modelo) y clasificarlos usando el modelo SVM lineal.

- Computacionalmente muy costoso.

Se utiliza el Truco del Kernel, que consiste en reemplazar el producto escalar (en el Dual) $x^{(i)T} x^{(j)}$ por $\phi(x^{(i)})^T \phi(x^{(j)})$

- Definamos la Función Kernel:

$$K(x^{(i)T}, x^{(j)}) = \phi(x^{(i)})^T \phi(x^{(j)})$$

- Función de Base Radial (RBF o Kernel Gaussiano):

$$K(x^{(i)}, x^{(j)}) = \exp\left(-\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma^2}\right)$$
$$K(x^{(i)}, x^{(j)}) = \exp\left(-\gamma \|x^{(i)} - x^{(j)}\|^2\right) \in [0, 1]$$
$$\gamma = \frac{1}{2\sigma^2} \text{ es parámetro libre que debe optimizarse.}$$

In [ ]:
```
from sklearn.svm import SVC

svm = SVC(kernel='rbf', random_state=1, gamma=0.10, C=1000000.0)
svm.fit(X_xor, y_xor)
plot_decision_regions(X_xor, y_xor,classifier=svm,plot_object=plt,
                      clases_names=name_clases_xor)

plt.legend(loc='upper left')
plt.tight_layout()
#plt.savefig('images/02_14.png', dpi=300)
plt.show()
```

SVM con datos iris

In [ ]:
```
svm = SVC(kernel='rbf', random_state=1, gamma=0.2, C=1.0)
svm.fit(X_train_std, y_train)

plot_decision_regions(X_combined_std, y_combined,classifier=svm,
                      plot_object=plt,test_idx=range(105, 150),
                      only_train=True,clases_names=name_clases)

plt.xlabel(f'{variable_names[0]} [cm]')
plt.ylabel(f'{variable_names[1]} [cm]')
plt.title('Linear SVM - sklearn (Train set)')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```

In [ ]:
```
svm = SVC(kernel='rbf', random_state=1, gamma=0.2, C=1.0)
svm.fit(X_train_std, y_train)

plot_decision_regions(X_combined_std, y_combined,classifier=svm,
                      plot_object=plt,test_idx=range(105, 150),
                      only_test=True,clases_names=name_clases)

plt.xlabel(f'{variable_names[0]} [cm]')
plt.ylabel(f'{variable_names[1]} [cm]')
plt.title('Linear SVM - sklearn (Test set)')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```

In [ ]:
```
r_exponents=[-6,-4,-1,0,2,4]
for idx, c in enumerate(r_exponents):
    svm = SVC(kernel='rbf', gamma=0.2, C=10.**c, random_state=1)
    svm.fit(X_train_std, y_train)
    if idx%3==0:
        fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(15, 4))
    plot_decision_regions(X_combined_std, y_combined,classifier=svm,
                          plot_object=axs[idx%3],test_idx=range(105, 150),
                          only_train=True,clases_names=name_clases)
    axs[idx%3].set_xlabel(f'{variable_names[0]} [cm]')
    axs[idx%3].set_ylabel(f'{variable_names[1]} [cm]')
    axs[idx%3].set_title('SVM non linear (Train set): C = '+str(10.**c))
    axs[idx%3].legend(loc='upper left')

    if idx%3==2:
        plt.show()
```

In [ ]:
```
r_exponents=[-6,-4,-1,0,2,4]
for idx, c in enumerate(r_exponents):
    svm = SVC(kernel='rbf', gamma=0.2, C=10.**c, random_state=1)
    svm.fit(X_train_std, y_train)
    if idx%3==0:
        fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(15, 4))
    plot_decision_regions(X_combined_std, y_combined,classifier=svm,
                          plot_object=axs[idx%3],test_idx=range(105, 150),
                          only_test=True,clases_names=name_clases)
    axs[idx%3].set_xlabel(f'{variable_names[0]} [cm]')
    axs[idx%3].set_ylabel(f'{variable_names[1]} [cm]')
    axs[idx%3].set_title('SVM non linear (Test set): C = '+str(10.**c))
    axs[idx%3].legend(loc='upper left')

    if idx%3==2:
        plt.show()
```

In [ ]:
```
gamma_s=[0.1,0.6, 1.0, 2.0, 10.0,200]
for idx, c in enumerate(gamma_s):
    svm = SVC(kernel='rbf', gamma=c, C=1.0, random_state=1)
    svm.fit(X_train_std, y_train)
    if idx%3==0:
        fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(15, 4))
    plot_decision_regions(X_combined_std, y_combined,classifier=svm,
                          plot_object=axs[idx%3],test_idx=range(105, 150),
                          only_train=True,clases_names=name_clases)
    axs[idx%3].set_xlabel(f'{variable_names[0]} [cm]')
    axs[idx%3].set_ylabel(f'{variable_names[1]} [cm]')
    axs[idx%3].set_title('SVM non linear (Train set): gamma = '+str(c))
    axs[idx%3].legend(loc='upper left')

    if idx%3==2:
        plt.show()
```

In [ ]:
```
gamma_s=[0.1,0.6, 1.0, 2.0, 10.0,200]
for idx, c in enumerate(gamma_s):
    svm = SVC(kernel='rbf', gamma=c, C=1.0, random_state=1)
    svm.fit(X_train_std, y_train)
    if idx%3==0:
        fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(15, 4))
    plot_decision_regions(X_combined_std, y_combined,classifier=svm,
                          plot_object=axs[idx%3],test_idx=range(105, 150),
                          only_test=True,clases_names=name_clases)
    axs[idx%3].set_xlabel(f'{variable_names[0]} [cm]')
    axs[idx%3].set_ylabel(f'{variable_names[1]} [cm]')
    axs[idx%3].set_title('SVM non linear (Test set): gamma = '+str(c))
    axs[idx%3].legend(loc='upper left')

    if idx%3==2:
        plt.show()
```

Preguntas:

- Qué otros Kernel ofrece SVC?
- Realizar análisis anterior con otros datos.