Actividad 2: Laboratorio De Aprendizaje Automático, UNIR

- Realizado por: Nicolás Felipe Trujillo Montero

In [1]:

```
#######################################################
# Tratamiento de Datos
#######################################################
import pandas as pd
import numpy as np

#######################################################
# Gráficos
#######################################################
import matplotlib.pyplot as plt
from matplotlib import style
from mlxtend.plotting import plot_decision_regions
import seaborn as sns

#######################################################
# Preprocesado y modelado
#######################################################
# Métodos de comprobacion de resultados
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV, KFold
from sklearn.metrics import classification_report, accuracy_score
from sklearn.svm import SVC # Support Vector Machine

# Red neuronal
from sklearn.neural_network import MLPClassifier

# Preprocesado
from sklearn.preprocessing import StandardScaler #z = (x - u) / s


#######################################################
# Configuración matplotlib
#######################################################
%matplotlib inline
plt.rcParams['image.cmap'], plt.rcParams['figure.dpi'], plt.rcParams['savefig.bbox'] = "bwr", "100", "tight"
style.use('ggplot') or plt.style.use('ggplot')

#######################################################
# Configuración warnings
#######################################################
import warnings
warnings.filterwarnings('ignore')
```

1.- Análisis Descriptivo de los Datos

In [2]:

```
# Leemos el fichero y mostramos los tipos
df_base = pd.read_csv("data.csv")
df_base.dtypes
```

Out[2]:

```
danceability        float64
energy              float64
key                   int64
loudness            float64
mode                  int64
speechiness         float64
acousticness        float64
instrumentalness    float64
liveness            float64
valence             float64
tempo               float64
duration_ms           int64
time_signature        int64
liked                 int64
dtype: object
```

a) Comentar de manera general qué se puede observar

Pues nos encontramos ante una función que lee de un DataFrame, nos dice el tipo de datos que contiene cada columna y el tipo del Dataframe que es de tipo objeto. En este caso, todas las variables son númericas de tipo punto flotante de 64 bits (punto flotante es equivalente al conjunto de los números reales), excepto key, mode, duration_ms, time_signature y liked que son númericas de tipo entero de 64 bits.

El significado de las variables es (obtenido de la página de kaggle):

- Acousticness: Una medida de confianza de 0.0 a 1.0 de si la pista es acústica. 1.0 representa una alta confianza en que la pista es acústica.

- Danceability: Danceability describe qué tan adecuada es una pista para bailar basada en una combinación de elementos musicales que incluyen tempo, estabilidad rítmica, fuerza de ritmo y regularidad general. Un valor de 0.0 es menos bailable y 1.0 es más bailable.
- Duration_ms: La duración de la pista en milisegundos.
- Energy: La energía es una medida de 0.0 a 1.0 y representa una medida perceptiva de intensidad y actividad. Por lo general, las pistas enérgicas se sienten rápidas, altas y ruidosas. Por ejemplo, el death metal tiene una alta energía, mientras que un preludio de Bach tiene una puntuación baja en la escala. Las características perceptivas que contribuyen a este atributo incluyen rango dinámico, volumen percibido, timbre, velocidad de inicio y entropía general.
- Instrumentalness: Predice si una pista no contiene voces. Los sonidos "Ooh" y "aah" son tratados como instrumentales en este contexto. Las pistas de rap o palabra hablada son claramente "vocales". Cuanto más cerca esté el valor de instrumentalidad de 1.0, mayor será la probabilidad de que la pista no contenga contenido vocal. Los valores superiores a 0,5 están destinados a representar pistas instrumentales, pero la confianza es mayor a medida que el valor se acerca a 1,0.
- Key: La clave en la que se encuentra la pista. Los enteros se asignan a tonos utilizando la notación estándar de clase de tono. Por ejemplo, 0 = C, 1 = C♯/D, 2 = D♭, y así sucesivamente.
- Liveness: Detecta la presencia de una audiencia en la grabación. Los valores de vida más altos representan una mayor probabilidad de que la pista se haya realizado en vivo. Un valor superior a 0,8 proporciona una gran probabilidad de que la pista esté activa.
- Loudness: La sonoridad total de una pista en decibelios (dB). Los valores de sonoridad se promedian en toda la pista y son útiles para comparar el volumen relativo de las pistas. El volumen es la cualidad de un sonido que es el correlato psicológico primario de la fuerza física (amplitud). Los valores típicos oscilan entre -60 y 0 db.
- Mode: Modo indica la modalidad (mayor o menor) de una pista, el tipo de escala de la que se deriva su contenido melódico. Mayor está representado por 1 y menor es 0.
- Speechiness: Speechiness detecta la presencia de palabras habladas en una pista. Cuanto más exclusivamente como el discurso sea la grabación (por ejemplo, programa de entrevistas, audiolibro, poesía), más cerca de 1.0 estará el valor del atributo. Los valores superiores a 0,66 describen pistas que probablemente están hechas completamente de palabras habladas. Los valores entre 0,33 y 0,66 describen pistas que pueden contener música y voz, ya sea en secciones o en capas, incluidos casos como la música rap. Los valores por debajo de 0,33 probablemente representan música y otras pistas no similares al habla.
- Tempo : El tempo total estimado de una pista en pulsaciones por minuto (BPM). En terminología musical, el tempo es la velocidad o el ritmo de una pieza dada y deriva directamente de la duración promedio del compás.
- Time_signature : Una firma de tiempo total estimada de una pista. La firma de tiempo (metro) es una convención notacional para especificar cuántos latidos hay en cada barra (o medida).
- Valencia: Una medida de 0.0 a 1.0 que describe la positividad musical transmitida por una pista. Las pistas con alta valencia suenan más positivas (por ejemplo, feliz, alegre, eufórica), mientras que las pistas con baja valencia suenan más negativas (por ejemplo, triste, deprimido, enojado).

Y la variable que hay que predecir:

- Liked: 1 para canciones con likes, 0 para canciones con dislike.

In [3]:

```
# Mostramos los principales valores estadísticos de las variables como pueden ser
# la frecuencia absoluta, la media aritmética, la desviación estándar, el elemento mínimo,
# el máximo y los cuartiles
df_base.describe()
```

Out[3]:

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	
count	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	
mean	0.636656	0.638431	5.497436	-9.481631	0.538462	0.148957	0.319093	0.192337	0.148455	0.493632	121.086174	21
std	0.216614	0.260096	3.415209	6.525086	0.499802	0.120414	0.320782	0.346226	0.105975	0.267695	28.084829	7
min	0.130000	0.002400	0.000000	-42.261000	0.000000	0.027800	0.000003	0.000000	0.033100	0.035300	60.171000	7
25%	0.462500	0.533500	2.000000	-9.962000	0.000000	0.056800	0.042200	0.000000	0.084000	0.269000	100.242000	17
50%	0.705000	0.659000	6.000000	-7.766000	1.000000	0.096200	0.213000	0.000008	0.105000	0.525000	124.896000	20
75%	0.799000	0.837500	8.000000	-5.829000	1.000000	0.230500	0.504000	0.097500	0.177000	0.717500	142.460500	24
max	0.946000	0.996000	11.000000	-2.336000	1.000000	0.540000	0.995000	0.969000	0.633000	0.980000	180.036000	65

b) Se pueden observar las estadísticas de las columnas numéricas. ¿Si se tienen 195 observaciones, a qué conclusiones podríamos llegar con estos datos? ¿Podríamos eliminar alguna variable?

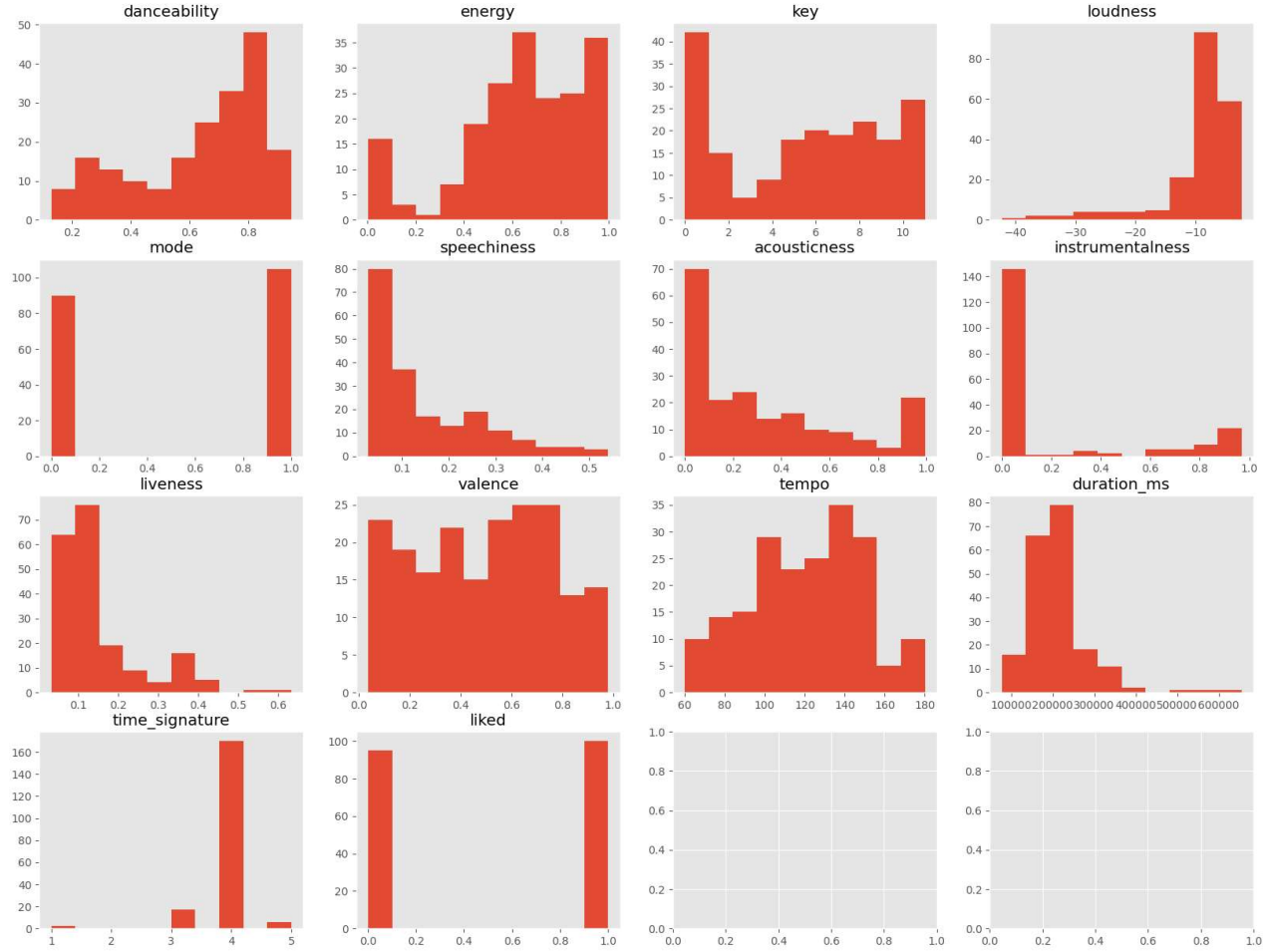
Este dataframe resultante nos indica por cada variable (columna) el número de registros/observaciones que hay, la media aritmética, la desviación estándar, el elemento más pequeño, el elemento que se encuentra en el primer cuartil, el elemento que se encuentra en el segundo cuartil, el elemento que se encuentra en el tercer cuartil y el elemento más grande.

Gracias a los datos podemos ver como están distribuido los datos o si tienen un sesgo, es decir, si se agrupa la mayoría de los datos en alguna zona. También podemos observar que cada dato tiene un intervalo distinto, por ejemplo, duration_ms se mueve entorno a [77203,655213], y el de loudness se mueve entorno a [-42.261, -2.336].

Si eliminar alguna variable significa quitar la columna entera de nuestro marco de datos, entonces tendríamos que investigar a posteriori si nos interesa quitarla porque a primera vista, teniendo en cuenta la falta de base de conocimientos de música, no creo que haga falta.

In [4]:

```
# Recorremos todas las variables y realizamos un histograma para comprobar
# la distribución de las variables
nvar = df_base.shape[1]
nrow = 4
ncol = 4

fig, ax = plt.subplots(nrow, ncol, figsize=(20, 15))

for irow in range(0, nrow):

    if (irow == nrow-1):
        ncol = ncol-2

    for icol in range(0,ncol):
        ax[irow, icol].hist(df_base[ df_base.columns[irow*nrow + icol] ])
        ax[irow, icol].grid()
        ax[irow, icol].set_title(df_base.columns[irow*nrow + icol])

ncol = ncol+2
```



c) Se muestran los histogramas de cada una de las columnas. ¿Qué se puede decir de la distribución de las variables?

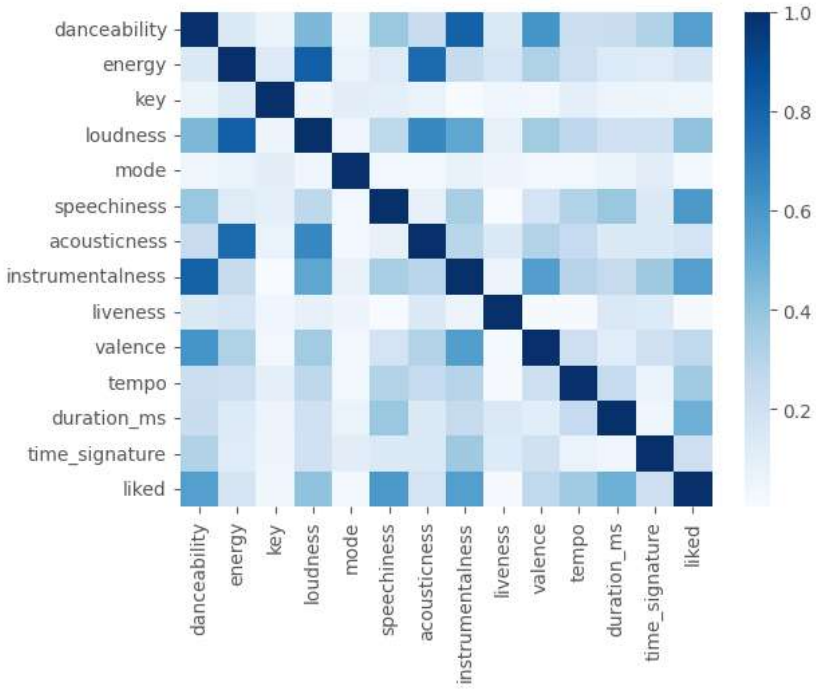
Pues de la distribución en general de las variables no se puede decir algo que compartan todas ya que se pueden apreciar distribuciones "normales sesgadas" y algunas que no siguen una distribución como tal (distribucion aleatoria). Lo que si podemos ver apreciar mejor son los intervalos por los que se mueven los datos.

In [5]:

```
# Realizamos la matriz de correlación
corr_map = df_base.corr()

# Al existir valores negativos, lo que vamos a hacer es ponderar los valores negativos como los positivos, es decir, si
# tenemos x depende de y un -0.5, significa que es inversamente lineal, por lo que vamos a ponderarlo como 0,5 de forma que
# vamos a poder ver más facilmente si una variable depende de otra, independientemente
corr_map_abs = abs(corr_map)
sns.heatmap(corr_map_abs, cmap="Blues")
```

Out[5]:

```
<AxesSubplot: >
```



d) Tenemos el mapa de calor de la matriz de correlaciones, por favor revise cuáles son las variables que mayor correlación tienen y si se puede eliminar alguna columna con base en este mapa de calor.

Gracias a la matriz de correlación (que hemos modificado para ver las variables más correlacionadas, independientemente de si son inversamente o directamente proporcional) podemos ver que la variable danceability y la variable energy son las que mayor correlación tienen con respecto a las demás (por separado)uy correlacionadas. Además, las tres correlaciones más altas (energy-loudness, energy-acousticness y danceability-instrumentalness) tienen bastante sentido siguiendo la definición de éstas (explicadas con anterioridad)

Uno de los momentos clave para eliminar variables para nuestro análisis es la visión de la matriz de correlación. Si en una fila en este caso, sin contar la relación de una variable consigo misma, los valores son muy cercanos a 0 eso significa que la variable no están correlacionadas y son independientes, como en el caso de key, mode o liveness que la mayoría de las correlaciones son por debajo de 0.5, por lo que cabría la posibilidad de eliminarlas (Habría que analizar la existencia de relaciones no lineales ya que la matriz de correlación analiza las relaciones lineales).

2.- Tratamiento de valores faltantes

In [6]:

```
# isnull() me comprueba si algún valor de df_base es NA o None
col_total_nulos = df_base.isnull().sum()
serie_col_nombres = col_total_nulos[col_total_nulos > 0]
display(serie_col_nombres)
```

```
Series([], dtype: int64)
```

No existen valores faltantes

In [7]:

```
(df_base==0).sum(axis=0)
```

Out[7]:

```
danceability         0
energy               0
key                 12
loudness             0
mode                90
speechiness          0
acousticness         0
instrumentalness    78
liveness             0
valence              0
tempo                0
duration_ms          0
time_signature       0
liked               95
dtype: int64
```

a) Podemos ver que existen columnas con ceros. Puede comentar ¿Qué puede estar ocurriendo con este conjunto de datos?

Pues no significa nada relevante como tal, ya que los valores 0 tienen sentido

3.- Entrenamiento de algoritmos

En este apartado aplicaremos los algoritmos descritos.

In [8]:

```
def graficar_accuracy_scores(estimator, train_x, train_y, test_x, test_y, nparts=5, seed=None, jobs=None):
    kfold = KFold(n_splits=nparts,shuffle=True, random_state=seed)
    fig,axes = plt.subplots(figsize=(7, 3))
    axes.set_title("Ratio de éxito(Accuracy)/Nro. Fold")
    axes.set_xlabel("Nro. Fold")
    axes.set_ylabel("Accuracy")
    train_scores = cross_val_score(estimator, train_x,train_y, cv = kfold, n_jobs=jobs, scoring="accuracy")
    test_scores = cross_val_score(estimator, test_x, test_y, cv = kfold, n_jobs=jobs, scoring="accuracy")
    train_sizes = range(1,nparts+1,1)
    axes.grid()
    axes.plot(train_sizes, train_scores, 'o-', color="r",label="Datos Entrenamiento")
    axes.plot(train_sizes, test_scores, 'o-', color="g",label="Datos de Test")
    axes.legend(loc="best")
    return train_scores
```

¿Tratamientos de datos faltantes?

In [9]:

```
df_base_res_1 = df_base.dropna(axis=0)
df_base_res_2 = df_base.dropna(axis=1)

all(df_base_res_1 == df_base_res_2)
```

Out[9]:

```
True
```

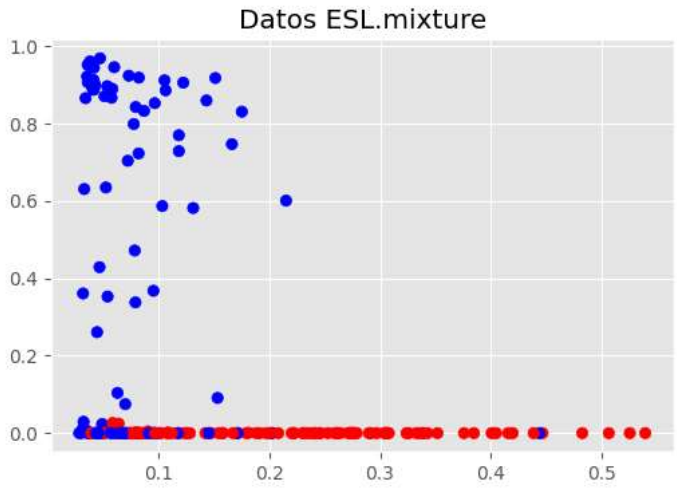
En este caso, no tenemos datos faltantes, ni hemos convertido valores 0 a NaN ya que tienen sentido, así que la complejidad de esta parte recae en realizar pruebas comparando modelos, para escoger el modelo que mejor ratio accuracy/fit consiga. Esto significa que tenemos que conseguir un modelo que no esté sobreajustado, ni subajustado, pero que consiga un accuracy bueno.

In [10]:

```
#########################################################
# Variables Globales
#########################################################
# Dimension del hiperplano
svm_dim = 2
endog_name = "liked"
endog = df_base[endog_name]
exog_names = corr_map_abs["liked"].sort_values(ascending=False)[range(1,1+svm_dim)]
exog = df_base[exog_names.keys()]

X_train, X_test, y_train, y_test = train_test_split(
                                        exog,
                                        endog,
                                        train_size   = 0.8,
                                        random_state = 1234,
                                        shuffle      = True
                                    )
```

In [11]:

```
fig, ax = plt.subplots(figsize=(6,4))
ax.scatter(df_base["speechiness"], df_base["instrumentalness"], c=df_base["liked"]);
ax.set_title("Datos ESL.mixture");
```

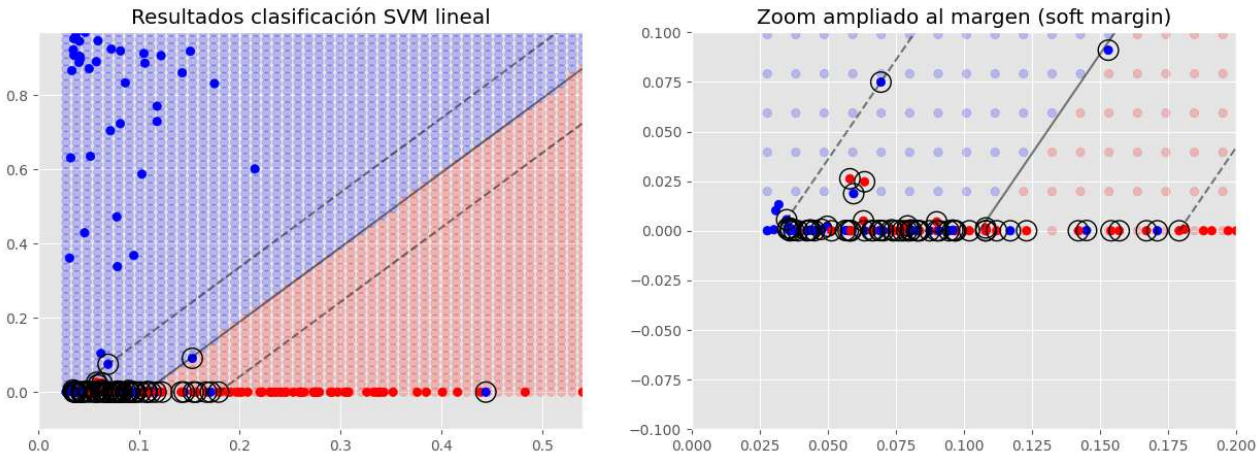


3.1.- Máquinas de soporte vectorial

Lo más óptimo es usar las variables con más correlación ya que si usamos todas, el plot se nos va a las 13 dimensiones (13 = nº de variables), por lo que vamos a clasificar liked, con respecto a instrumentalness y speechiness.

In [12]:

```
#########################################################
# SVM Lineal
#########################################################

# Representación gráfica de los límites de clasificación

# Grid de valores, guardamos en "x" y en "y" las rectas que marcaran las fronteras
x = np.linspace(np.min(X_train["speechiness"]), np.max(X_train["speechiness"]), 50)
y = np.linspace(np.min(X_train["instrumentalness"]), np.max(X_train["instrumentalness"]), 50)
Y, X = np.meshgrid(y, x)
grid = np.vstack([X.ravel(), Y.ravel()]).T

# Creamos el modelo SVM Lineal
modelo = SVC(C = 100, kernel = 'linear', random_state=1234)
modelo.fit(X_train, y_train)

# Predicción valores grid
pred_grid = modelo.predict(grid)

fig, ax = plt.subplots(1,2,figsize=(15,5))
ax[0].scatter(grid[:,0], grid[:,1], c=pred_grid, alpha = 0.2)
ax[0].scatter(X_train["speechiness"], X_train["instrumentalness"], c=y_train, alpha = 1)

# Vectores soporte
ax[0].scatter(
    modelo.support_vectors_[:, 0],
    modelo.support_vectors_[:, 1],
    s=200, linewidth=1,
    facecolors='none', edgecolors='black'
)

# Hiperplano de separación
ax[0].contour(
    X,
    Y,
    modelo.decision_function(grid).reshape(X.shape),
    colors = 'k',
    levels = [-1, 0, 1],
    alpha  = 0.5,
    linestyles = ['--', '-', '--']
)

ax[0].set_xlim(left=0.0)
ax[0].set_ylim(bottom=-0.1)

ax[0].set_title("Resultados clasificación SVM lineal");

ax[1].scatter(grid[:,0], grid[:,1], c=pred_grid, alpha = 0.2)
ax[1].scatter(X_train["speechiness"], X_train["instrumentalness"], c=y_train, alpha = 1)

# Vectores soporte
ax[1].scatter(
    modelo.support_vectors_[:, 0],
    modelo.support_vectors_[:, 1],
    s=200, linewidth=1,
    facecolors='none', edgecolors='black'
)

# Hiperplano de separación
ax[1].contour(
    X,
    Y,
    modelo.decision_function(grid).reshape(X.shape),
    colors = 'k',
    levels = [-1, 0, 1],
    alpha  = 0.5,
    linestyles = ['--', '-', '--']
)

ax[1].set_xlim(left=0.0, right=0.2)
ax[1].set_ylim(bottom=-0.1, top = 0.1)
ax[1].set_title("Zoom ampliado al margen (soft margin)");
```



En este caso, se ha creado un hiperplano que trata de separar de forma lineal en dos campos los pares speechiness-instrumentalness etiquetados, de manera que en los puntos que caigan en la zona roja deberían ser puntos rojos según el modelo y los puntos azules iguales.

Además, se ha aplicado un soft margin paralelos a la recta de separación a ambos lados, para que los puntos que estén entre ambos márgenes no se sepan con claridad como clasificarlos.

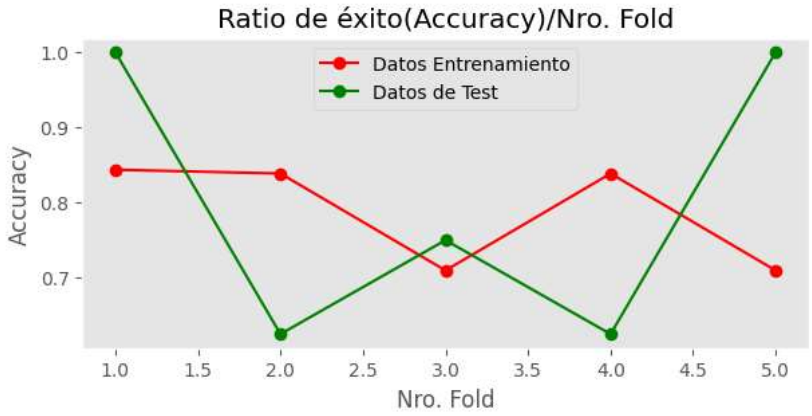
In [13]:

```
Y_test_pred = modelo.predict(X_test)
print(classification_report(y_test, Y_test_pred))

predicciones = modelo.predict(X_test)
accuracy = accuracy_score(
            y_true    = y_test,
            y_pred    = predicciones,
            normalize = True
           )
print("")
print(f"El accuracy de test es: {100*accuracy}%")
```

```
              precision    recall  f1-score   support

           0       0.87      0.95      0.91        21
           1       0.94      0.83      0.88        18

    accuracy                           0.90        39
   macro avg       0.90      0.89      0.90        39
weighted avg       0.90      0.90      0.90        39


El accuracy de test es: 89.74358974358975%
```

In [14]:

```
graficar_accuracy_scores(modelo, X_train, y_train, X_test, y_test, nparts=5, seed=1234, jobs=-1);
```



Comentario Support Vector Machine:

- **Classification report**: La etiqueta 0 es mejor etiquetada que la etiqueta 1 ya que las medidas de precision (exactitud), f1-score y recall de medias son mayores que la de la etiqueta 1. Además, podemos observar como el accuracy es de un 90% cosa que nos puede llevar a la conclusión de que existe un sobreajuste, pero veremos en el siguiente punto que no.
- **Graficar_accuracy_scores**: Los gráficos nos indican que, para el modelo entrenado, el primer y cuarto fold nos da un mejor ratio accuracy para los datos de entrenamiento, y para los datos de validación cruzada, nos da un mejor resultado el primer y quinto fold. Esto significa que los valores del conjunto de test están dispersos entre sí, por eso para la gráfica de los datos de test varía en accuracy según el fold que miremos, mientras que los valores del conjunto de entrenamiento

están cercanos, por eso la gráfica de los datos del conjunto de entrenamiento varían muy poco en accuracy según el fold que miremos. Además, como de 5 fold, se cumple que los fold 2-4 del conjunto de entrenamiento tienen mayor accuracy que los 2-4 fold del conjunto de test, y para el 1-3-5 es al contrario, podemos llegar a la conclusión de que no existe ni sobreajuste, ni subajuste.

Posibles mejoras para el SVM "modelo = SVC(C = 100, kernel = 'linear', random_state=1234)":

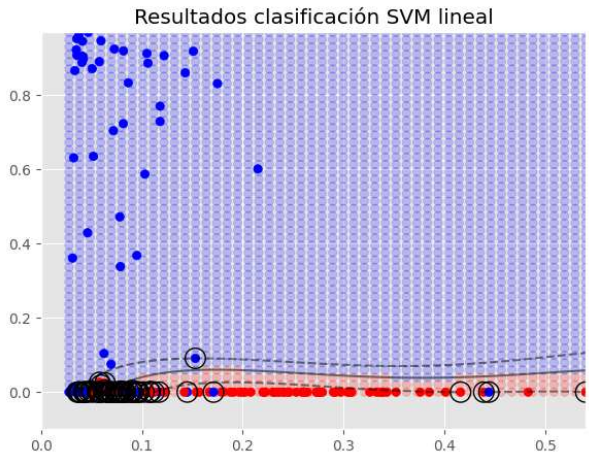
- Parámetro C: Delimita la amplitud del margen, a mayor C, menor amplitud habrá del margen.
- Parámetro kernel: Dice la forma del hiperplano que va a tomar, según la documentación existen 'linear', 'poly', 'rbf', 'sigmoid' y 'precomputed', es decir, tipos lineales o radiales. Dependiendo de el que escojamos, se creará una recta, un polinomio, ...
- Parámetro random_state: Controla la semilla de los números aleatorios que se forman cuando se realiza la estimación probabilística.
- Realizar una normalización con StandardScaler

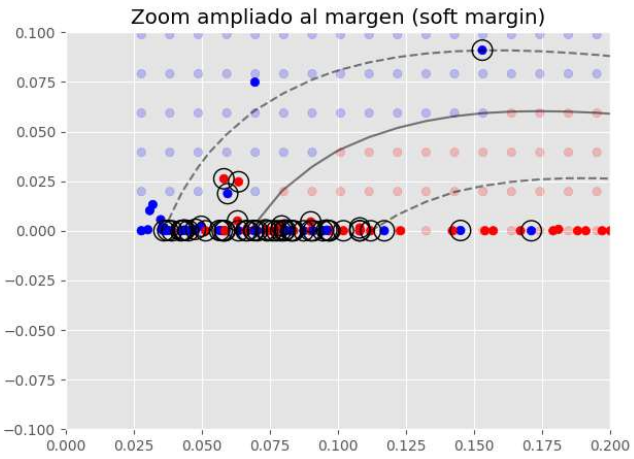
¿Cambiaría algo para realizar alguna mejora?

En primera instancia, me pensaba que no hacía falta, pero si observamos en la siguiente celda el kernel rbf, vemos como clasifica mejor.

In [15]:

```
#########################################################
# SVM Radial
#########################################################

# Representación gráfica de los límites de clasificación

# Grid de valores, guardamos en "x" y en "y" las rectas que marcaran las fronteras
x = np.linspace(np.min(X_train["speechiness"]), np.max(X_train["speechiness"]), 50)
y = np.linspace(np.min(X_train["instrumentalness"]), np.max(X_train["instrumentalness"]), 50)
Y, X = np.meshgrid(y, x)
grid = np.vstack([X.ravel(), Y.ravel()]).T

# Creamos el modelo SVM Lineal
modelo = SVC(C = 100, kernel = 'rbf', random_state=1234)
modelo.fit(X_train, y_train)

# Predicción valores grid
pred_grid = modelo.predict(grid)

fig, ax = plt.subplots(1,2,figsize=(15,5))
ax[0].scatter(grid[:,0], grid[:,1], c=pred_grid, alpha = 0.2)
ax[0].scatter(X_train["speechiness"], X_train["instrumentalness"], c=y_train, alpha = 1)

# Vectores soporte
ax[0].scatter(
    modelo.support_vectors_[:, 0],
    modelo.support_vectors_[:, 1],
    s=200, linewidth=1,
    facecolors='none', edgecolors='black'
)

# Hiperplano de separación
ax[0].contour(
    X,
    Y,
    modelo.decision_function(grid).reshape(X.shape),
    colors = 'k',
    levels = [-1, 0, 1],
    alpha  = 0.5,
    linestyles = ['--', '-', '--']
)

ax[0].set_xlim(left=0.0)
ax[0].set_ylim(bottom=-0.1)

ax[0].set_title("Resultados clasificación SVM lineal");

ax[1].scatter(grid[:,0], grid[:,1], c=pred_grid, alpha = 0.2)
ax[1].scatter(X_train["speechiness"], X_train["instrumentalness"], c=y_train, alpha = 1)

# Vectores soporte
ax[1].scatter(
    modelo.support_vectors_[:, 0],
    modelo.support_vectors_[:, 1],
    s=200, linewidth=1,
    facecolors='none', edgecolors='black'
)

# Hiperplano de separación
ax[1].contour(
    X,
    Y,
    modelo.decision_function(grid).reshape(X.shape),
    colors = 'k',
    levels = [-1, 0, 1],
    alpha  = 0.5,
    linestyles = ['--', '-', '--']
)

ax[1].set_xlim(left=0.0, right=0.2)
ax[1].set_ylim(bottom=-0.1, top = 0.1)
ax[1].set_title("Zoom ampliado al margen (soft margin)");
```





In [16]:

```
Y_test_pred = modelo.predict(X_test)
print(classification_report(y_test, Y_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.90      0.90        21
           1       0.89      0.89      0.89        18

    accuracy                           0.90        39
   macro avg       0.90      0.90      0.90        39
weighted avg       0.90      0.90      0.90        39
```

In [17]:

```
Y_test_pred = modelo.predict(X_test)
print(classification_report(y_test, Y_test_pred))

accuracy = accuracy_score(
            y_true    = y_test,
            y_pred    = Y_test_pred,
            normalize = True
           )
print("")
print(f"El accuracy de test es: {100*accuracy}%")
```

```
              precision    recall  f1-score   support

           0       0.90      0.90      0.90        21
           1       0.89      0.89      0.89        18

    accuracy                           0.90        39
   macro avg       0.90      0.90      0.90        39
weighted avg       0.90      0.90      0.90        39


El accuracy de test es: 89.74358974358975%
```

En este caso, no creo que sea necesario aplicar funciones para optimizar parámetros como GridSearchCV, ya que hemos llegado a una muy buena aproximación.

3.2.- Red Neuronal

En este caso, vamos a aplicar una red neuronal para realizar la clasificación. Para ello, primero vamos a escalar los datos para conseguir un mejor modelo.

In [18]:

```
# Variable endógena
endog_name = "liked"

# Escalamos el dataframe
sc = StandardScaler()
scaled_df = pd.DataFrame(sc.fit_transform(df_base[exog_names.keys()]), columns=exog_names.keys())
scaled_df[endog_name] = df_base[endog_name]

endog = scaled_df[endog_name]
exog = scaled_df[exog_names.keys()]

X_train, X_test, y_train, y_test = train_test_split(
                                        exog,
                                        endog,
                                        train_size   = 0.8,
                                        random_state = 1234,
                                        shuffle      = True
                                    )
```

Recordamos los parámetros del clasificador Red Neuronal MPL:

- **Hidden_layer_sizes** : Este parámetro nos permite establecer el número de capas y el número de nodos que deseamos tener en el Clasificador de Redes Neuronales. Cada elemento de la tupla representa el número de nodos en la i-ésima posición donde i es el índice de la tupla. Por lo tanto, la longitud de la tupla indica el número total de capas ocultas en la red.
- **Max_iter**: Denota el número de épocas.
- **Activación**: La función de activación para las capas ocultas.
- **Solver**: este parámetro especifica el algoritmo para la optimización del peso en los nodos.
- **Random_state**: el parámetro permite establecer una semilla para reproducir los mismos resultados

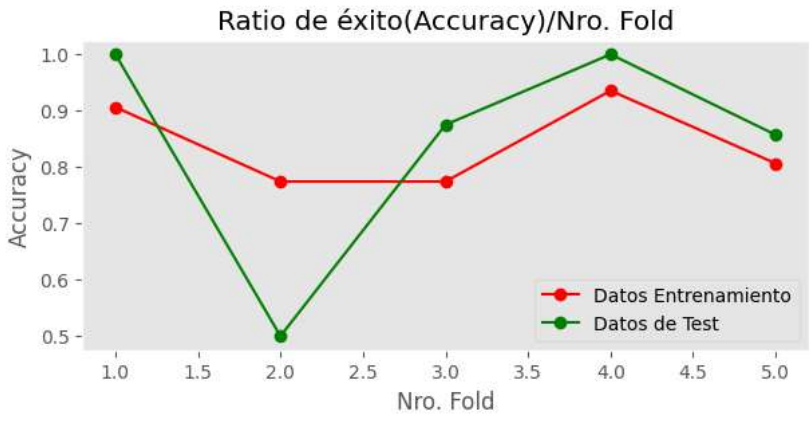
In [19]:

```
# Realizamos el modelo con los parámetros demostrados
classifier = MLPClassifier(hidden_layer_sizes=(150,100,50), max_iter=300,activation = 'relu',solver='sgd',random_state=1234)

# Entrenamos el modelo
classifier.fit(X_train, y_train)

# Predicción sobre las instancias de prueba
y_pred = classifier.predict(X_test)

print(classification_report(y_test, Y_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.90      0.90        21
           1       0.89      0.89      0.89        18

    accuracy                           0.90        39
   macro avg       0.90      0.90      0.90        39
weighted avg       0.90      0.90      0.90        39
```

In [22]:

```
graficar_accuracy_scores(modelo, X_train, y_train, X_test, y_test, nparts=5, seed=1234, jobs=-1);
```



Comentario Neural Network:

- **Classification report**: La etiqueta 0 es mejor etiquetada que la etiqueta 1 ya que las medidas de precision (exactitud), f1-score y recall son mayores que la de la etiqueta 1. Además, podemos observar como el accuracy es de un 90% cosa que nos puede llevar a la conclusión de que existe un sobreajuste.
- **Graficar_accuracy_scores**: Los gráficos nos indican que, para el modelo entrenado, el segundo fold nos da un mejor ratio accuracy para los datos de entrenamiento, y para los datos de validación cruzada, nos da un mejor resultado los demás fold. Esto significa que, como la mayoría de los datos de test tienen mejor ratio, nos sugiere que existe un sobreajuste ya que los datos de test se clasifican mucho mejor que los de entrenamiento.

Posibles mejoras para el SVM "classifier = MLPClassifier(hidden_layer_sizes=(150,100,50), max_iter=300,activation = 'relu',solver='sgd',random_state=1234)":

- Parámetro Hidden_layer_sizes: Podríamos aumentar o reducir las neuronas multicapa, para ver si el modelo varia
- Parámetro Max_iter: Podríamos aumentar el número de épocas para mejorar el resultado
- Parámetro Activation: según la documentación, hay 4 funciones que se podrían usar para los cálculos en las neuronas multicapa. Sin embargo, la función que usamos es la mejor para la clasificación.
- Parámetro Solver: podríamos modificar el parámetro para que la optimización de los pesos sea distinta.
- Parámetro random_state: Modificando la aleatoriedad, podríamos obtener un mejor resultado.

¿Cambiaría algo para realizar alguna mejora?

En primera instancia, no cambiaría ningún parámetro, ya que el accuracy es bueno. Sin embargo, modificaría algún parámetro (diría que hidden_layer_sizes) para modificar el sobreajuste explicado anteriormente.

Comparación de Modelos

Una vez realizados los dos modelos y viendo los resultados, yo escogería la máquina de soporte vectorial ya que con el scatter plot hemos visto que se clasifica de una forma mejor, logrando el mismo accuracy, pero con mejor ajuste ya que no está ni sobreajustado, ni subajustado el modelo.

¿Cuáles serían las ventajas y desventajas?
Pues son modelos complementarios, es decir, la ventaja de SVM es que podemos ver directamente la separación en el plano para saber que parámetros usar ya que se ve de forma gráfica, pero la desventaja es que cuando nos vamos a n-dimensiones no podemos visualizarlo bien. Sin embargo, la NN Multicapa nos permite usar esos n atributos, pero no podemos ver gráficamente el resultado por lo que es más dificil optimizar el modelo.

¿Son útiles los modelos para los datos?
Mientras que estemos en un problema de clasificación y realicemos un análisis inicial correcto, podemos utilizar estos modelos llegando a una buena predicción.

Comentarios Adicionales

- Alguna forma de mejorar sería implementar un sistema de correlaciones para analizar cuales son las variables mejor para pronosticar o sino un algoritmo de PCA.