SVM Clasificación iris

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
%matplotlib inline
```

In [3]:

```
iris = datasets.load_iris()
iris
```

Out[3]:

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
        [5.1, 3.8, 1.5, 0.3],
        [5.4, 3.4, 1.7, 0.2],
        [5.1, 3.7, 1.5, 0.4],
        [4.6, 3.6, 1. , 0.2],
        [5.1, 3.3, 1.7, 0.5],
        [4.8, 3.4, 1.9, 0.2],
        [5. , 3. , 1.6, 0.2],
        [5. , 3.4, 1.6, 0.4],
        [5.2, 3.5, 1.5, 0.2],
        [5.2, 3.4, 1.4, 0.2],
        [4.7, 3.2, 1.6, 0.2],
        [4.8, 3.1, 1.6, 0.2],
        [5.4, 3.4, 1.5, 0.4],
        [5.2, 4.1, 1.5, 0.1],
        [5.5, 4.2, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.2],
        [5. , 3.2, 1.2, 0.2],
        [5.5, 3.5, 1.3, 0.2],
        [4.9, 3.6, 1.4, 0.1],
        [4.4, 3. , 1.3, 0.2],
        [5.1, 3.4, 1.5, 0.2],
        [5. , 3.5, 1.3, 0.3],
        [4.5, 2.3, 1.3, 0.3],
        [4.4, 3.2, 1.3, 0.2],
        [5. , 3.5, 1.6, 0.6],
        [5.1, 3.8, 1.9, 0.4],
        [4.8, 3. , 1.4, 0.3],
        [5.1, 3.8, 1.6, 0.2],
        [4.6, 3.2, 1.4, 0.2],
        [5.3, 3.7, 1.5, 0.2],
        [5. , 3.3, 1.4, 0.2],
        [7. , 3.2, 4.7, 1.4],
```

```
[6.4, 3.2, 4.5, 1.5],
[6.9, 3.1, 4.9, 1.5],
[5.5, 2.3, 4. , 1.3],
[6.5, 2.8, 4.6, 1.5],
[5.7, 2.8, 4.5, 1.3],
[6.3, 3.3, 4.7, 1.6],
[4.9, 2.4, 3.3, 1. ],
[6.6, 2.9, 4.6, 1.3],
[5.2, 2.7, 3.9, 1.4],
[5. , 2. , 3.5, 1. ],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1. ],
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2. ],
[6.4, 2.7, 5.3, 1.9],
[6.8, 3. , 5.5, 2.1],
[5.7, 2.5, 5. , 2. ],
[5.8, 2.8, 5.1, 2.4],
[6.4, 3.2, 5.3, 2.3],
[6.5, 3. , 5.5, 1.8],
```

```
        [7.7, 3.8, 6.7, 2.2],
        [7.7, 2.6, 6.9, 2.3],
        [6. , 2.2, 5. , 1.5],
        [6.9, 3.2, 5.7, 2.3],
        [5.6, 2.8, 4.9, 2. ],
        [7.7, 2.8, 6.7, 2. ],
        [6.3, 2.7, 4.9, 1.8],
        [6.7, 3.3, 5.7, 2.1],
        [7.2, 3.2, 6. , 1.8],
        [6.2, 2.8, 4.8, 1.8],
        [6.1, 3. , 4.9, 1.8],
        [6.4, 2.8, 5.6, 2.1],
        [7.2, 3. , 5.8, 1.6],
        [7.4, 2.8, 6.1, 1.9],
        [7.9, 3.8, 6.4, 2. ],
        [6.4, 2.8, 5.6, 2.2],
        [6.3, 2.8, 5.1, 1.5],
        [6.1, 2.6, 5.6, 1.4],
        [7.7, 3. , 6.1, 2.3],
        [6.3, 3.4, 5.6, 2.4],
        [6.4, 3.1, 5.5, 1.8],
        [6. , 3. , 4.8, 1.8],
        [6.9, 3.1, 5.4, 2.1],
        [6.7, 3.1, 5.6, 2.4],
        [6.9, 3.1, 5.1, 2.3],
        [5.8, 2.7, 5.1, 1.9],
        [6.8, 3.2, 5.9, 2.3],
        [6.7, 3.3, 5.7, 2.5],
        [6.7, 3. , 5.2, 2.3],
        [6.3, 2.5, 5. , 1.9],
        [6.5, 3. , 5.2, 2. ],
        [6.2, 3.4, 5.4, 2.3],
        [5.9, 3. , 5.1, 1.8]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),
 'frame': None,
 'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
 'DESCR': '.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data Set Chara
cteristics:**\n\n    :Number of Instances: 150 (50 in each of three classes)\n    :Number of 
Attributes: 4 numeric, predictive attributes and the class\n    :Attribute Information:\n     
- sepal length in cm\n        - sepal width in cm\n        - petal length in cm\n        - pe
tal width in cm\n        - class:\n                - Iris-Setosa\n                - Iris-Vers
icolour\n                - Iris-Virginica\n                \n    :Summary Statistics:\n\n   
============== ==== ==== ======= ===== ====================\n                    Min  Max   M
ean    SD   Class Correlation\n    ============== ==== ==== ======= ===== ===================
=\n    sepal length:   4.3  7.9   5.84   0.83    0.7826\n    sepal width:    2.0  4.4   3.05  
0.43   -0.4194\n    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)\n    petal widt
h:    0.1  2.5   1.20   0.76    0.9565  (high!)\n    ============== ==== ==== ======= ===== =
===================\n\n    :Missing Attribute Values: None\n    :Class Distribution: 33.3% fo
r each of 3 classes.\n    :Creator: R.A. Fisher\n    :Donor: Michael Marshall (MARSHALL%PLU@i
o.arc.nasa.gov)\n    :Date: July, 1988\n\nThe famous Iris database, first used by Sir R.A. Fi
sher. The dataset is taken\nfrom Fisher\'s paper. Note that it\'s the same as in R, but not a
s in the UCI\nMachine Learning Repository, which has two wrong data points.\n\nThis is perhap
s the best known database to be found in the\npattern recognition literature.  Fisher\'s pape
r is a classic in the field and\nis referenced frequently to this day.  (See Duda & Hart, for
example.)  The\ndata set contains 3 classes of 50 instances each, where each class refers to 
a\ntype of iris plant.  One class is linearly separable from the other 2; the\nlatter are NOT
linearly separable from each other.\n\n.. topic:: References\n\n   - Fisher, R.A. "The use of
multiple measurements in taxonomic problems"\n     Annual Eugenics, 7, Part II, 179-188 (193
6); also in "Contributions to\n     Mathematical Statistics" (John Wiley, NY, 1950).\n   - Du
da, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.\n     (Q327.D83) Joh
```

```
n Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.\n    - Dasarathy, B.V. (1980) "Nosing Arou
nd the Neighborhood: A New System\n      Structure and Classification Rule for Recognition in
Partially Exposed\n      Environments".  IEEE Transactions on Pattern Analysis and Machine\n
Intelligence, Vol. PAMI-2, No. 1, 67-71.\n    - Gates, G.W. (1972) "The Reduced Nearest Neighb
or Rule".  IEEE Transactions\n      on Information Theory, May 1972, 431-433.\n    - See also:
1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II\n      conceptual clustering syst
em finds 3 classes in the data.\n    - Many, many more ...',
 'feature_names': ['sepal length (cm)',
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 'filename': 'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\sklearn\\datasets\\data\\iris.c
sv'}
```

In [120…

```
iris.feature_names
```

Out[120…

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [22]:

```
iris.data.shape
```

Out[22]:

```
(150, 4)
```

In [87]:

```
X = iris.data[:,:2]#Todas las filas y las columnas hasta la 2
Y = iris.target
```

In [88]:

```
##Sólo se realiza en función del ancho y largo de los pétalos.
x_min, x_max = X[:,0].min()-1, X[:,0].max()+1 #Todas las filas columna 0
y_min, y_max = X[:,1].min()-1, X[:,1].max()+1 #Todas las filas columna 1
h = (x_max-x_min)/100

xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

x_plot = np.c_[xx.ravel(), yy.ravel()]
```

In [45]:

```
x_plot #Creamos la parrilla de dibujo, de a 100 saltitos que es h
```

Out[45]:

```
array([[3.3  , 1.   ],
       [3.356, 1.   ],
       [3.412, 1.   ],
       ...,
       [8.732, 5.368],
       [8.788, 5.368],
       [8.844, 5.368]])
```

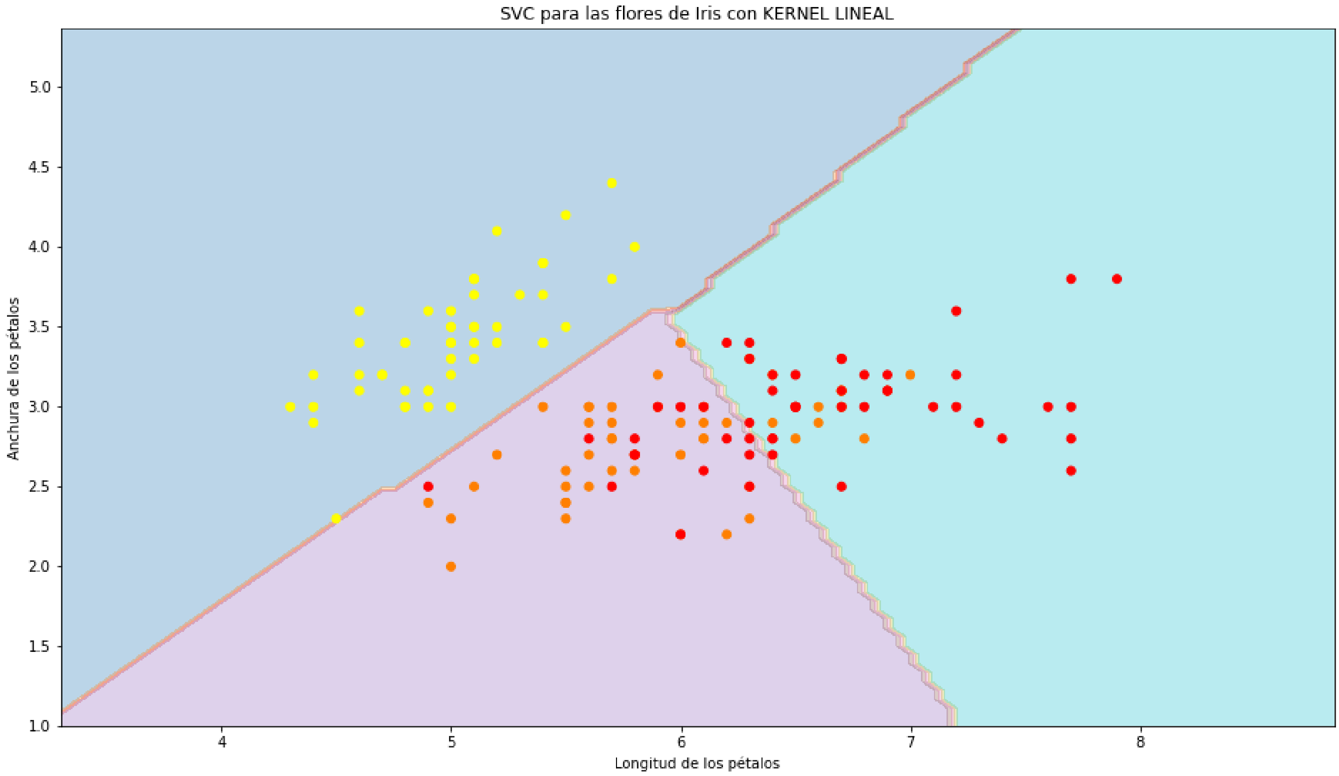
In [46]:

```
C = 1.0
svc = svm.SVC(kernel="linear", C=C, decision_function_shape="ovr").fit(X,Y)
Ypred = svc.predict(x_plot)
Ypred = Ypred.reshape(xx.shape)
```

In [48]:

```
plt.figure(figsize=(16,9))
plt.contourf(xx,yy,Ypred, cmap=plt.cm.tab10, alpha = 0.3)
plt.scatter(X[:,0], X[:,1], c=Y, cmap=plt.cm.autumn_r)
plt.xlabel("Longitud de los pétalos")
plt.ylabel("Anchura de los pétalos")
plt.xlim(xx.min(), xx.max())
plt.title("SVC para las flores de Iris con KERNEL LINEAL")
```

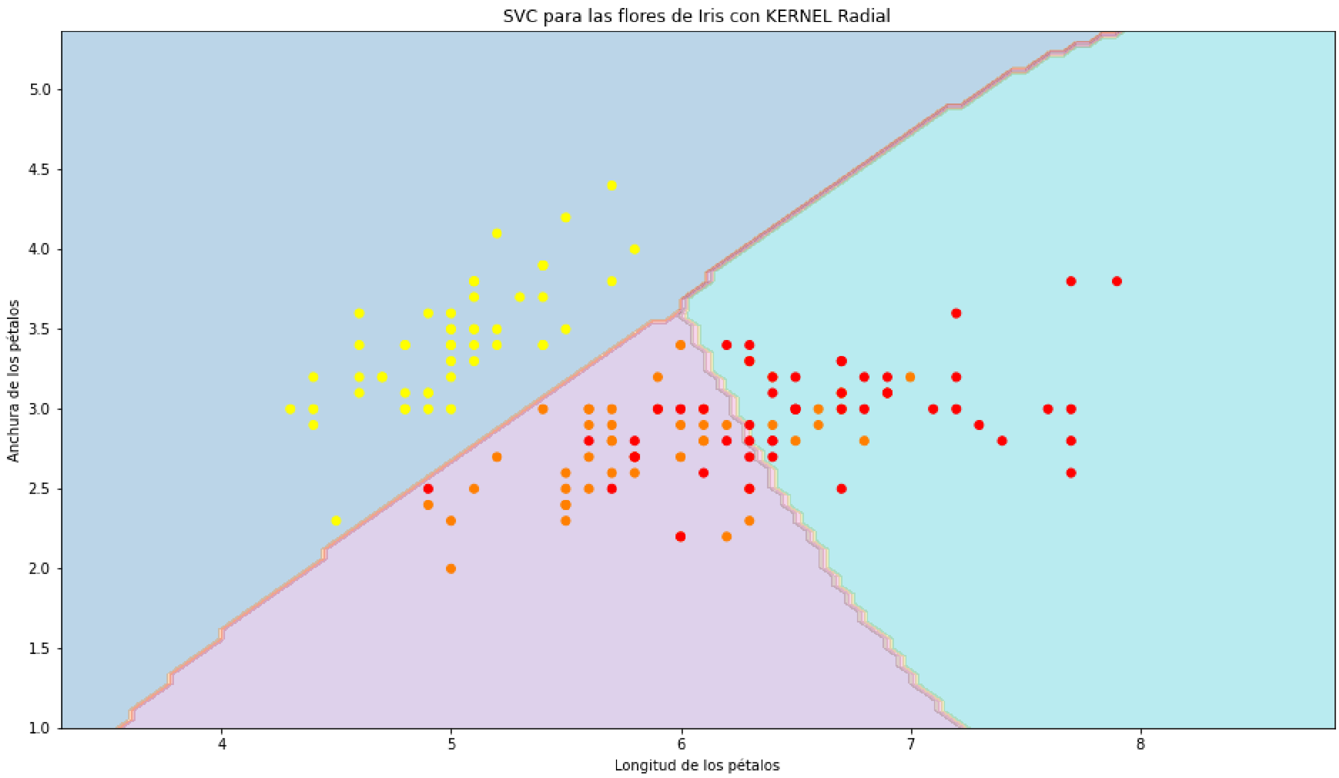
Out[48]: Text(0.5, 1.0, 'SVC para las flores de Iris con KERNEL LINEAL')



In [49]:
```
C = 1.0
svc = svm.SVC(kernel="rbf", C=C, decision_function_shape="ovr").fit(X,Y)
Ypred = svc.predict(x_plot)
Ypred = Ypred.reshape(xx.shape)
```

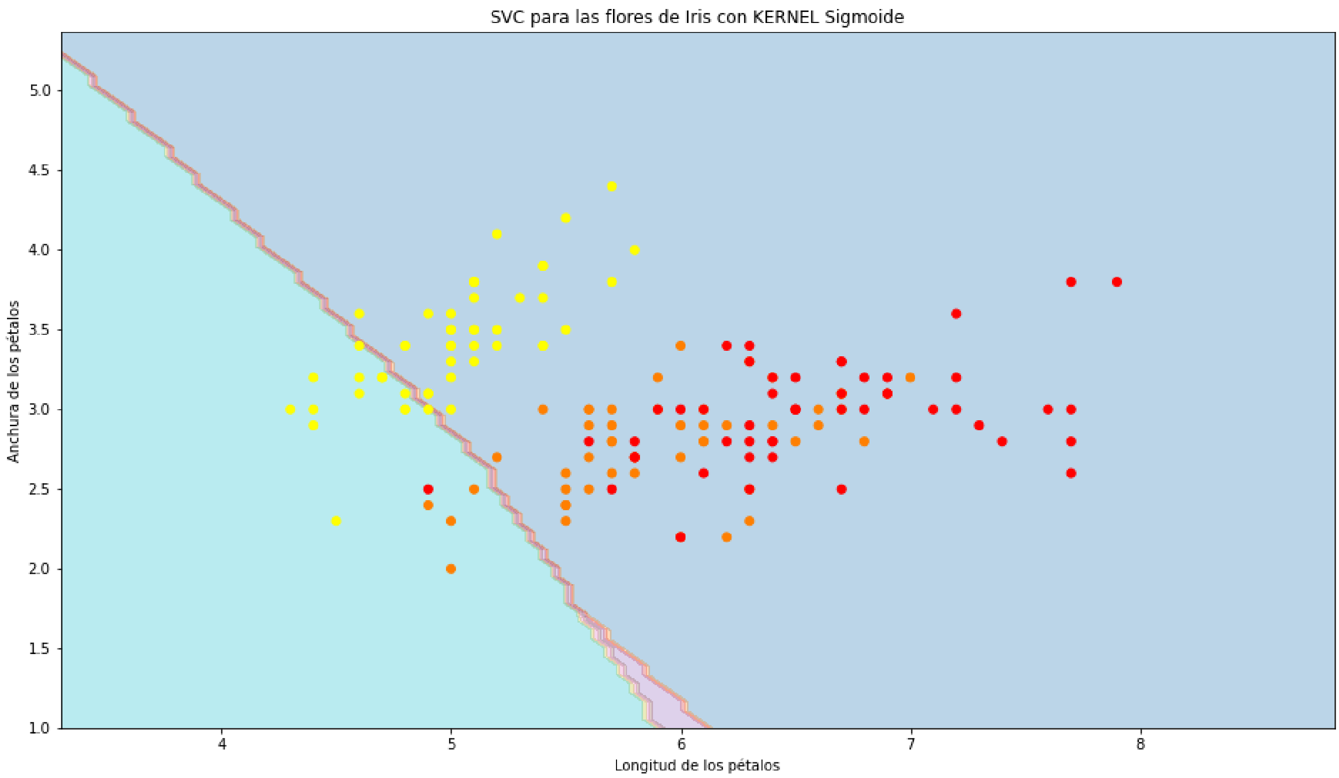
In [50]:
```
plt.figure(figsize=(16,9))
plt.contourf(xx,yy,Ypred, cmap=plt.cm.tab10, alpha = 0.3)
plt.scatter(X[:,0], X[:,1], c=Y, cmap=plt.cm.autumn_r)
plt.xlabel("Longitud de los pétalos")
plt.ylabel("Anchura de los pétalos")
plt.xlim(xx.min(), xx.max())
plt.title("SVC para las flores de Iris con KERNEL Radial")
```

Out[50]: Text(0.5, 1.0, 'SVC para las flores de Iris con KERNEL Radial')



In [51]:
```
C = 1.0
svc = svm.SVC(kernel="sigmoid", C=C, decision_function_shape="ovr").fit(X,Y)
Ypred = svc.predict(x_plot)
Ypred = Ypred.reshape(xx.shape)
plt.figure(figsize=(16,9))
plt.contourf(xx,yy,Ypred, cmap=plt.cm.tab10, alpha = 0.3)
plt.scatter(X[:,0], X[:,1], c=Y, cmap=plt.cm.autumn_r)
plt.xlabel("Longitud de los pétalos")
plt.ylabel("Anchura de los pétalos")
plt.xlim(xx.min(), xx.max())
plt.title("SVC para las flores de Iris con KERNEL Sigmoide")
```

Out[51]:
```
Text(0.5, 1.0, 'SVC para las flores de Iris con KERNEL Sigmoide')
```

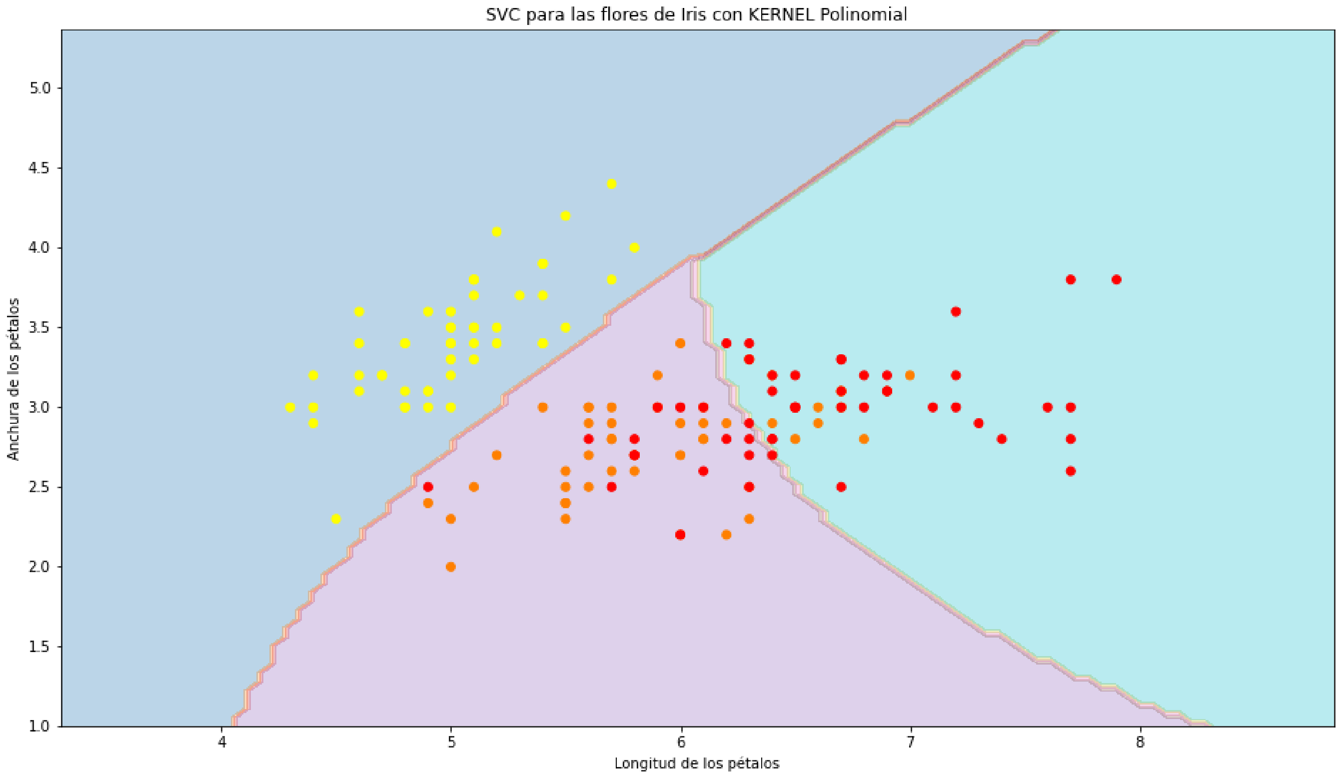


In [52]:
```
C = 1.0
```

```
svc = svm.SVC(kernel="poly", C=C, decision_function_shape="ovr").fit(X,Y)
Ypred = svc.predict(x_plot)
Ypred = Ypred.reshape(xx.shape)
plt.figure(figsize=(16,9))
plt.contourf(xx,yy,Ypred, cmap=plt.cm.tab10, alpha = 0.3)
plt.scatter(X[:,0], X[:,1], c=Y, cmap=plt.cm.autumn_r)
plt.xlabel("Longitud de los pétalos")
plt.ylabel("Anchura de los pétalos")
plt.xlim(xx.min(), xx.max())
plt.title("SVC para las flores de Iris con KERNEL Polinomial")
```

Out[52]:

```
Text(0.5, 1.0, 'SVC para las flores de Iris con KERNEL Polinomial')
```



In [53]:

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report
from sklearn.utils import shuffle
```

In [54]:

```
X, Y =shuffle(X,Y, random_state=0)
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.25, random_state=0)
```

In [55]:

```
parametres =[
    {
        'kernel':['rbf'],
        'gamma':[1e-4, 1e-3, 0.1, 0.2, 0.5],
        'C':[1,10,100,1000]
    },
    {
        'kernel':['linear'],
        'C':[1,10,100,1000]
    }
]
```

In [56]:

```
clf = GridSearchCV(svm.SVC(decision_function_shape='ovr'), param_grid=parametres, cv=5)
clf.fit(X,Y)
```

Out[56]:

```
GridSearchCV(cv=5, estimator=SVC(),
```

```
                    param_grid=[{'C': [1, 10, 100, 1000],
                                 'gamma': [0.0001, 0.001, 0.1, 0.2, 0.5],
                                 'kernel': ['rbf']},
                                {'C': [1, 10, 100, 1000], 'kernel': ['linear']}])
```

In [57]:

```python
clf.best_params_
```

Out[57]:

```
{'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
```

In [59]:

```python
means = clf.cv_results_['mean_test_score']
stds = clf.cv_results_['std_test_score']
params = clf.cv_results_['params']
for m, s, p in zip(means, stds, params):
    print("%0.3f (+/-%0.3f) para %r" %(m, 2*s, p))
```

```
0.747 (+/-0.124) para {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.747 (+/-0.124) para {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.807 (+/-0.129) para {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
0.787 (+/-0.124) para {'C': 1, 'gamma': 0.2, 'kernel': 'rbf'}
0.780 (+/-0.116) para {'C': 1, 'gamma': 0.5, 'kernel': 'rbf'}
0.747 (+/-0.124) para {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.747 (+/-0.124) para {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.773 (+/-0.098) para {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
0.780 (+/-0.116) para {'C': 10, 'gamma': 0.2, 'kernel': 'rbf'}
0.767 (+/-0.126) para {'C': 10, 'gamma': 0.5, 'kernel': 'rbf'}
0.747 (+/-0.124) para {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.813 (+/-0.124) para {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.780 (+/-0.080) para {'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
0.773 (+/-0.078) para {'C': 100, 'gamma': 0.2, 'kernel': 'rbf'}
0.767 (+/-0.084) para {'C': 100, 'gamma': 0.5, 'kernel': 'rbf'}
0.813 (+/-0.124) para {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.760 (+/-0.107) para {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.780 (+/-0.080) para {'C': 1000, 'gamma': 0.1, 'kernel': 'rbf'}
0.767 (+/-0.084) para {'C': 1000, 'gamma': 0.2, 'kernel': 'rbf'}
0.753 (+/-0.124) para {'C': 1000, 'gamma': 0.5, 'kernel': 'rbf'}
0.773 (+/-0.098) para {'C': 1, 'kernel': 'linear'}
0.767 (+/-0.094) para {'C': 10, 'kernel': 'linear'}
0.767 (+/-0.094) para {'C': 100, 'kernel': 'linear'}
0.767 (+/-0.094) para {'C': 1000, 'kernel': 'linear'}
```

La función **GridSearchCV(svm.SVC(decision_function_shape='ovr'), param_grid=parametres, cv=5)** lo que hace es crear tantos modelos parámetros en la grilla existan. Ejemplo, en el modelo *RBF* tenemos 1(kernel).5(gammas).3(C) = 20 modelos diferentes. Para el *Linear* tenemos 1(kernel).4(C) = 4 modelos diferentes. Luego, obitene los mejores parámetros en base al score más elevado. Nosotros pdemos imprimir en pantalla todos los parámetros si los llamamos uno por uno para ir viendo cuál es el mejor. O bien utilizar el método *clf.bestparams* que ya los hace automáticamente.

In [60]:

```python
y_pred = clf.predict(X_test)
```

In [61]:

```python
print(classification_report(Y_test, y_pred, target_names=["Setosa", "Versicolor", "Virginica'
```

```
              precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00        11
  Versicolor       0.60      0.82      0.69        11
   Virginica       0.83      0.62      0.71        16

    accuracy                           0.79        38
   macro avg       0.81      0.81      0.80        38
```

```
weighted avg       0.81      0.79      0.79        38
```

Recordar, con el método **classification_report** podemos contrastar las Y reales con las Y obtenidas, y ver los parámetros principales, como vemos la presición es el cociente de VP/VP+FP y el Recall = VP/VP+FN las setosas han sido perfectament clasificadas, mientras más cerca de 1 mejor es el grado de presición.

Resumen final de la clasificación de Iris

In [202…

```python
def svm_iris( Npred=[2,3,4], Ndiv=[10,100,1000], C=1.0, gamma = 0.01, kernel ="rbf"):
    """Suppor Vector Machine para IRIS"""
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    from sklearn import svm, datasets
    %matplotlib inline
    # Data sets disponibles en sklearn: 'iris', 'boston', 'wine', 'breast_cancer'

    #Cargo data set de la librería de sklear.datasets
    df= datasets.load_iris()

    #Particiono las filas y columnas de X y de Y
    X = df.data[:,Npred-2:Npred]#Todas las filas y las columnas desde Npred-2 hasta Npred
    Y = df.target

    ##Eligo el mínimo y máximo de los valores del dataset y trazo grilla.
    x_min, x_max = X[:,0].min()-1, X[:,0].max()+1
    y_min, y_max = X[:,1].min()-1, X[:,1].max()+1
    h = (x_max-x_min)/Ndiv

    #Divido los rangos en 100 divisiones, creo los ejes
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

    #Realizo la grilla juntando las variables
    x_plot = np.c_[xx.ravel(), yy.ravel()]

    #Realizo modelo SVC y creo ploteo
    svc = svm.SVC(kernel=kernel, C=C, gamma=gamma, decision_function_shape="ovr").fit(X,Y)
    Ypred = svc.predict(x_plot)
    Ypred = Ypred.reshape(xx.shape)
    plt.figure(figsize=(16,9))
    plt.contourf(xx,yy,Ypred, cmap=plt.cm.tab10  , alpha = 0.3)
    plt.scatter(X[:,0], X[:,1], c=Y, cmap=plt.cm.autumn_r)
    plt.xlabel("" +str(df.feature_names[Npred]))
    plt.ylabel("" +str(df.feature_names[Npred-1]))

    plt.xlim(xx.min(), xx.max())
    plt.title("SVC para las flores de Iris con KERNEL "+kernel)
```

In [203…

```python
from ipywidgets import interact, fixed
```

In [204…

```python
interact(svm_iris, C=[0.01,0.1,1,10,100,1000, 10000, 1E6, 1E10],
        gamma=[1e-5, 1e-4, 1e-3, 1e-2, 0.1, 0.2, 0.5, 0.99],
        kernel=["rbf", "linear", "sigmoid", "poly"])
```

Out[204…

```
<function __main__.svm_iris(Npred=[2, 3, 4], Ndiv=[10, 100, 1000], C=1.0, gamma=0.01, kernel
='rbf')>
```