Support Vector Machines and Kernels

Professor: Rogério Martins Gomes

Alunos: Aritana Noara Costa Santos e Victor Augusto Januário da Cruz

Introduction

We will be using the wine quality data set for these exercises. This data set contains various chemical properties of wine, such as acidity, sugar, pH, and alcohol. It also contains a quality metric (3-9, with highest being better) and a color (red or white). The name of the file is `Wine_Quality_Data.csv`.

In [1]:
```
from __future__ import print_function
import os
data_path = ['data']
```

Question 1

- Import the data.
- Create the target variable `y` as a 1/0 column where 1 means red.
- Create a `pairplot` for the dataset.
- Create a bar plot showing the correlations between each column and `y`
- Pick the most 2 correlated fields (using the absolute value of correlations) and create `X`
- Use MinMaxScaler to scale `X`. Note that this will output a np.array. Make it a DataFrame again and rename the columns appropriately.

In [2]:
```
import pandas as pd
import numpy as np

filepath = os.sep.join(data_path + ['Wine_Quality_Data.csv'])
data = pd.read_csv(filepath, sep=',')
```

In [3]:
```
y = (data['color'] == 'red').astype(int)
fields = list(data.columns[:-1])  # everything except "color"
correlations = data[fields].corrwith(y)
correlations.sort_values(inplace=True)
correlations
```

Out[3]:
```
total_sulfur_dioxide   -0.700357
free_sulfur_dioxide    -0.471644
residual_sugar         -0.348821
citric_acid            -0.187397
quality                -0.119323
alcohol                -0.032970
pH                      0.329129
density                 0.390645
fixed_acidity           0.486740
sulphates               0.487218
chlorides               0.512678
volatile_acidity        0.653036
dtype: float64
```

In [4]:
```
pip install seaborn
```

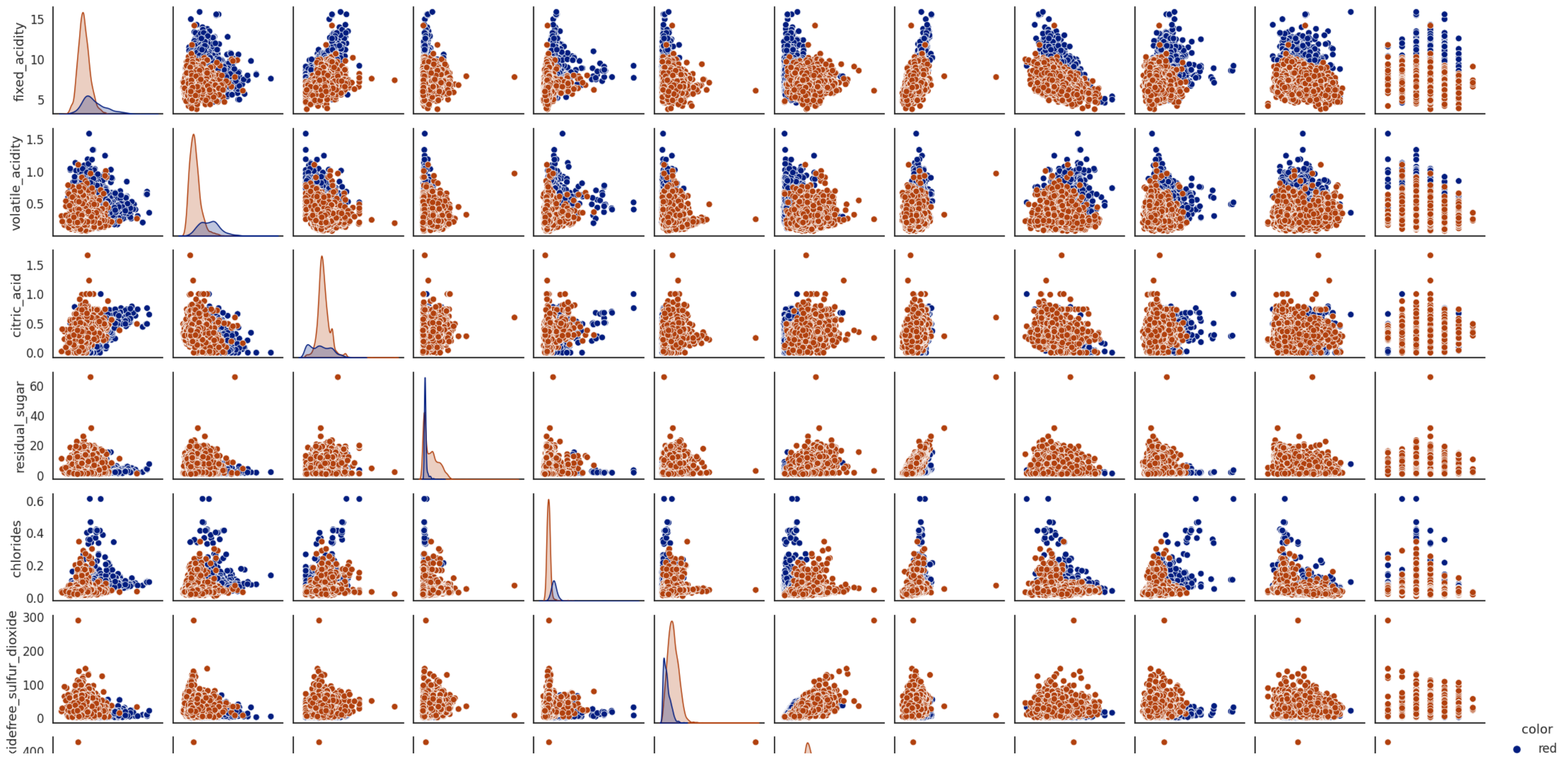
```
Requirement already satisfied: seaborn in /home/aritana/my_jupyter_notebook/my_jupyter_notebook/lib/python3.8/site-pack
ages (0.11.1)
Requirement already satisfied: numpy>=1.15 in /home/aritana/my_jupyter_notebook/my_jupyter_notebook/lib/python3.8/site-
packages (from seaborn) (1.20.3)
Requirement already satisfied: pandas>=0.23 in /home/aritana/my_jupyter_notebook/my_jupyter_notebook/lib/python3.8/site
-packages (from seaborn) (1.2.4)
Requirement already satisfied: matplotlib>=2.2 in /home/aritana/my_jupyter_notebook/my_jupyter_notebook/lib/python3.8/s
ite-packages (from seaborn) (3.4.2)
Requirement already satisfied: scipy>=1.0 in /home/aritana/my_jupyter_notebook/my_jupyter_notebook/lib/python3.8/site-p
ackages (from seaborn) (1.6.3)
Requirement already satisfied: pillow>=6.2.0 in /home/aritana/my_jupyter_notebook/my_jupyter_notebook/lib/python3.8/sit
e-packages (from matplotlib>=2.2->seaborn) (8.2.0)
Requirement already satisfied: python-dateutil>=2.7 in /home/aritana/my_jupyter_notebook/my_jupyter_notebook/lib/python
3.8/site-packages (from matplotlib>=2.2->seaborn) (2.8.1)
Requirement already satisfied: pyparsing>=2.2.1 in /home/aritana/my_jupyter_notebook/my_jupyter_notebook/lib/python3.8/
site-packages (from matplotlib>=2.2->seaborn) (2.4.7)
Requirement already satisfied: cycler>=0.10 in /home/aritana/my_jupyter_notebook/my_jupyter_notebook/lib/python3.8/site
-packages (from matplotlib>=2.2->seaborn) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /home/aritana/my_jupyter_notebook/my_jupyter_notebook/lib/python3.
8/site-packages (from matplotlib>=2.2->seaborn) (1.3.1)
Requirement already satisfied: pytz>=2017.3 in /home/aritana/my_jupyter_notebook/my_jupyter_notebook/lib/python3.8/site
-packages (from pandas>=0.23->seaborn) (2021.1)
Requirement already satisfied: six in /home/aritana/my_jupyter_notebook/my_jupyter_notebook/lib/python3.8/site-packages
```

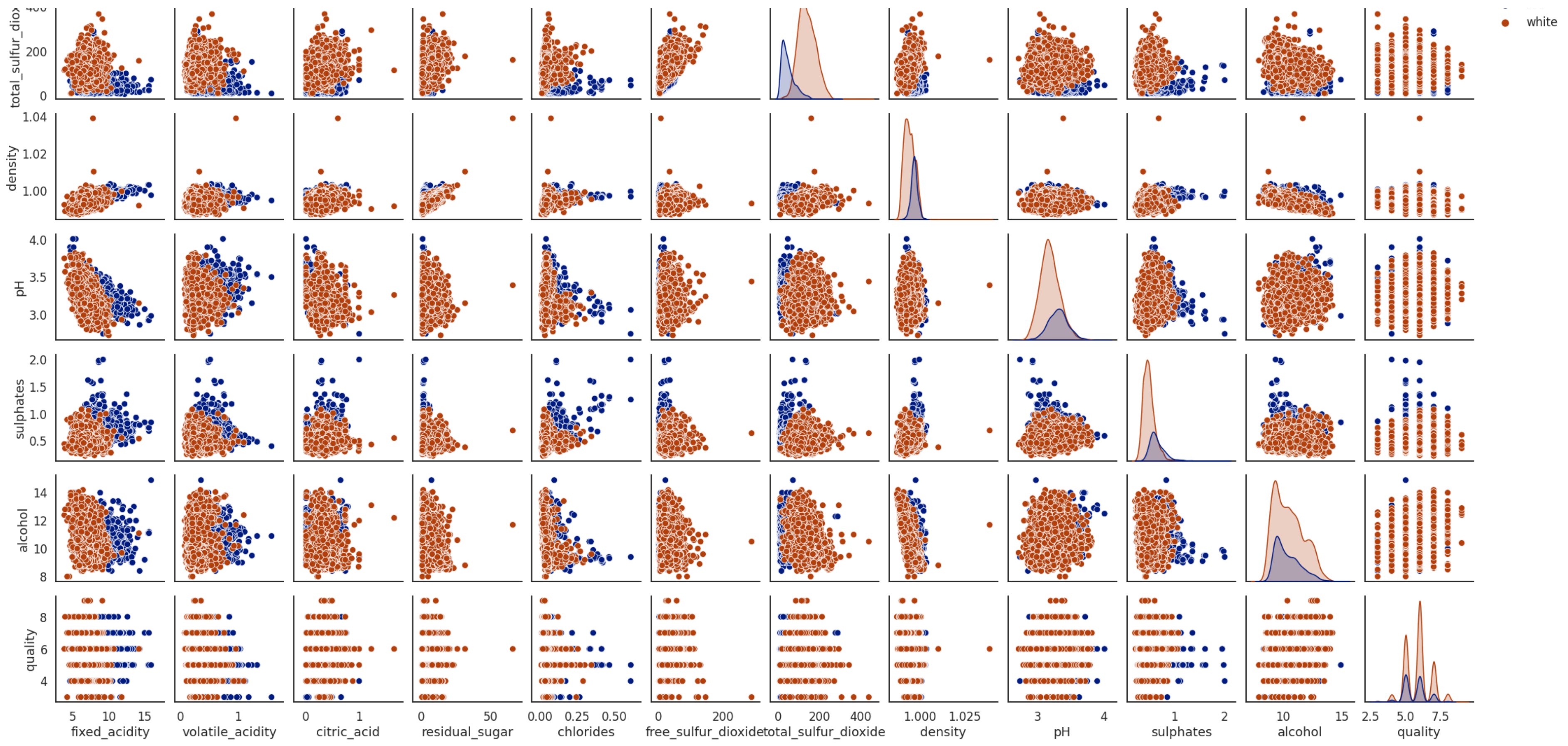
```
(from cycler>=0.10->matplotlib>=2.2->seaborn) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [5]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

sns.set_context('talk')
sns.set_palette('dark')
sns.set_style('white')
```

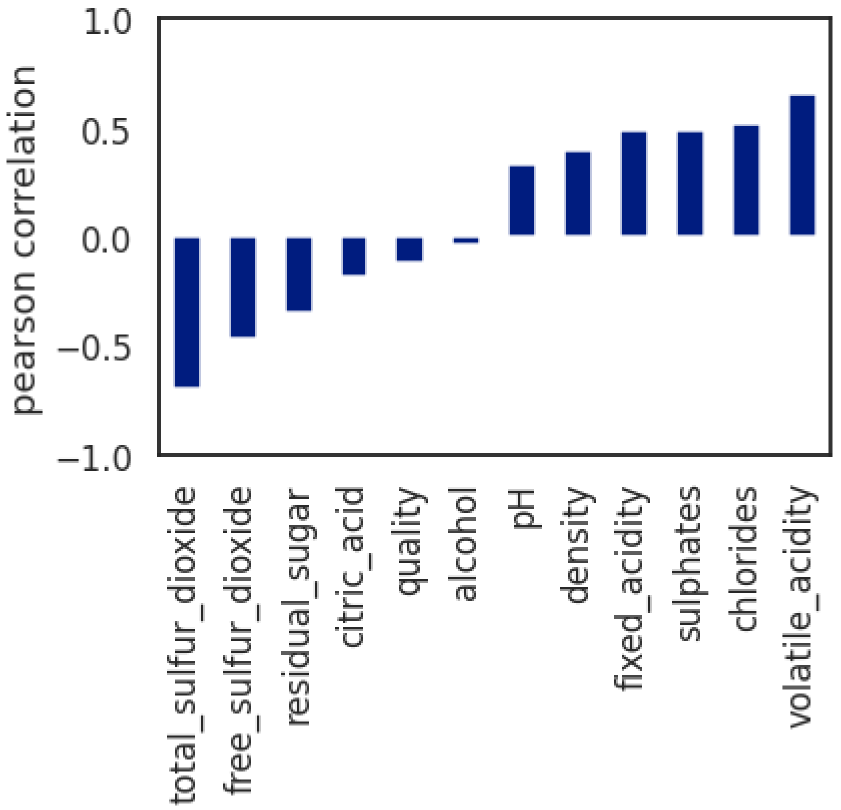
In [12]:
```
sns.pairplot(data, hue='color')
```

Out[12]:
```
<seaborn.axisgrid.PairGrid at 0x7ff853a18f70>
```





In [7]:
```
ax = correlations.plot(kind='bar')
ax.set(ylim=[-1, 1], ylabel='pearson correlation');
```



In [8]:

```
from sklearn.preprocessing import MinMaxScaler

fields = correlations.map(abs).sort_values().iloc[-2:].index
print(fields)
X = data[fields]
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
X = pd.DataFrame(X, columns=['%s_scaled' % fld for fld in fields])
print(X.columns)
```

```
Index(['volatile_acidity', 'total_sulfur_dioxide'], dtype='object')
Index(['volatile_acidity_scaled', 'total_sulfur_dioxide_scaled'], dtype='object')
```

Question 2

The goal for this question is to look at the decision boundary of a LinearSVC classifier on this dataset. Check out this example in sklearn's documentation.

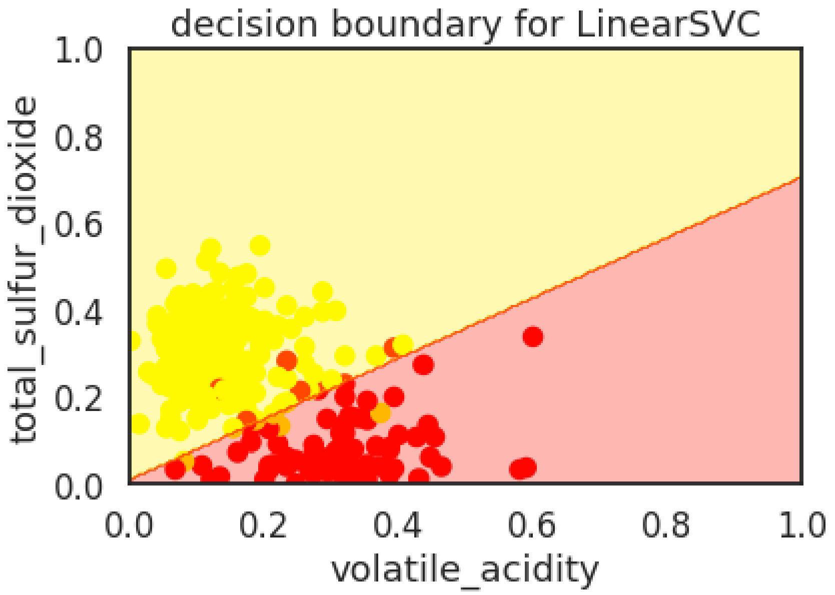
- Fit a Linear Support Vector Machine Classifier to `X` , `y` .
- Pick 300 samples from `X` . Get the corresponding `y` value. Store them in variables `X_color` and `y_color` . This is because original dataset is too large and it produces a crowded plot.
- Modify `y_color` so that it has the value "red" instead of 1 and 'yellow' instead of 0.
- Scatter plot X_color's columns. Use the keyword argument "color=y_color" to color code samples.
- Use the code snippet below to plot the decision surface in a color coded way.

```
x_axis, y_axis = np.arange(0, 1, .005), np.arange(0, 1, .005)
xx, yy = np.meshgrid(x_axis, y_axis)
xx_ravel = xx.ravel()
yy_ravel = yy.ravel()
X_grid = pd.DataFrame([xx_ravel, yy_ravel]).T
y_grid_predictions = *[YOUR MODEL]*.predict(X_grid)
y_grid_predictions = y_grid_predictions.reshape(xx.shape)
ax.contourf(xx, yy, y_grid_predictions, cmap=plt.cm.autumn_r, alpha=.3)
```

Feel free to experiment with different parameter choices for LinearSVC and see the decision boundary.

In [10]:
```
from sklearn.svm import LinearSVC

LSVC = LinearSVC()
LSVC.fit(X, y)

X_color = X.sample(300, random_state=45)
y_color = y.loc[X_color.index]
y_color = y_color.map(lambda r: 'red' if r == 1 else 'yellow')
ax = plt.axes()
ax.scatter(
    X_color.iloc[:, 0], X_color.iloc[:, 1],
    color=y_color, alpha=1)
# -----------
x_axis, y_axis = np.arange(0, 1.005, .005), np.arange(0, 1.005, .005)
xx, yy = np.meshgrid(x_axis, y_axis)
xx_ravel = xx.ravel()
yy_ravel = yy.ravel()
X_grid = pd.DataFrame([xx_ravel, yy_ravel]).T
```

```
y_grid_predictions = LSVC.predict(X_grid)
y_grid_predictions = y_grid_predictions.reshape(xx.shape)
ax.contourf(xx, yy, y_grid_predictions, cmap=plt.cm.autumn_r, alpha=.3)
# -----------
ax.set(
    xlabel=fields[0],
    ylabel=fields[1],
    xlim=[0, 1],
    ylim=[0, 1],
    title='decision boundary for LinearSVC');
```



Question 3

Let's now fit a Gaussian kernel SVC and see how the decision boundary changes.

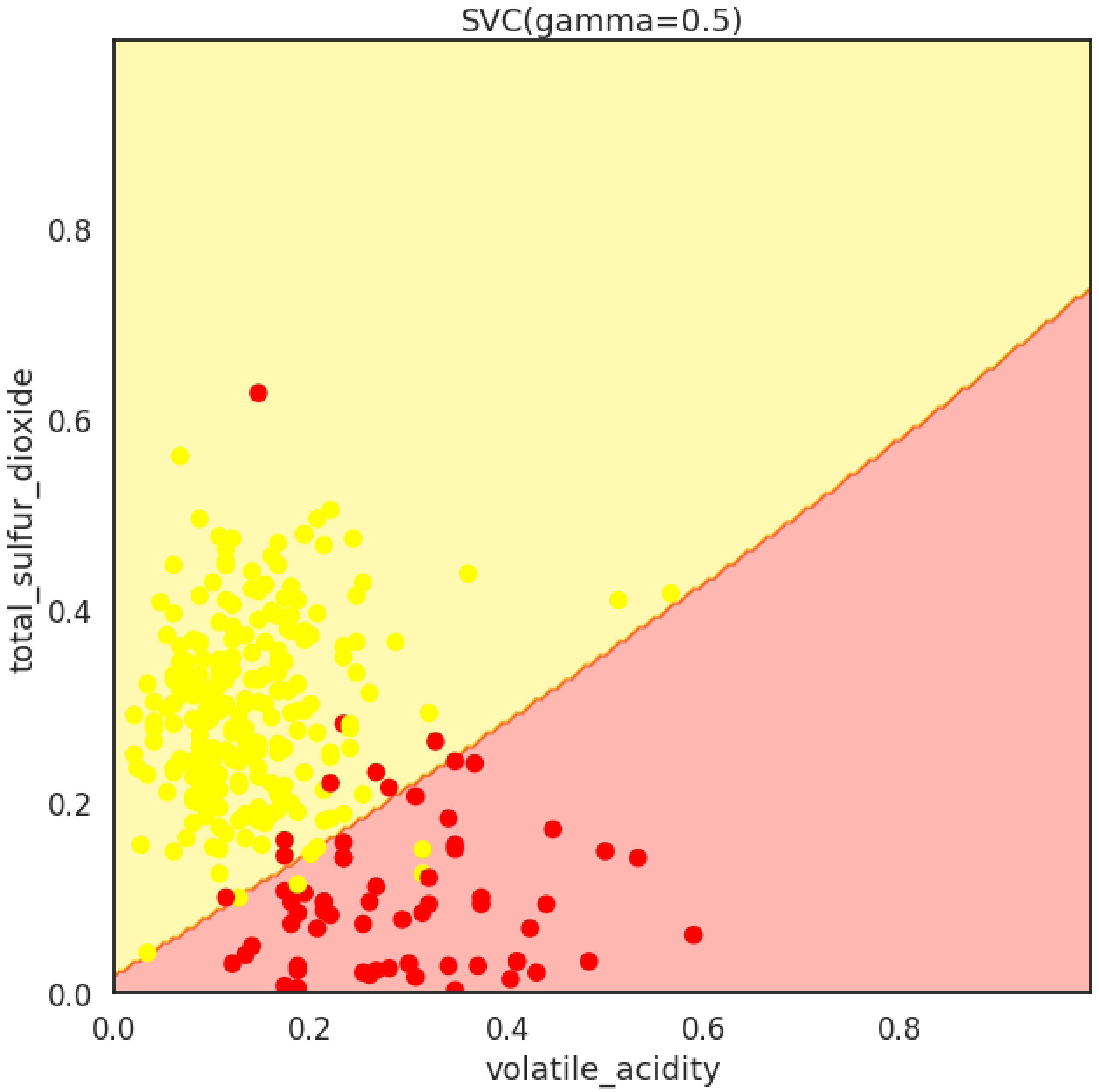
- Consolidate the code snippets in Question 2 into one function which takes in an estimator, `X` and `y`, and produces the final plot with decision boundary. The steps are:
    1. fit model
    2. get sample 300 records from X and the corresponding y's
    3. create grid, predict, plot using ax.contourf

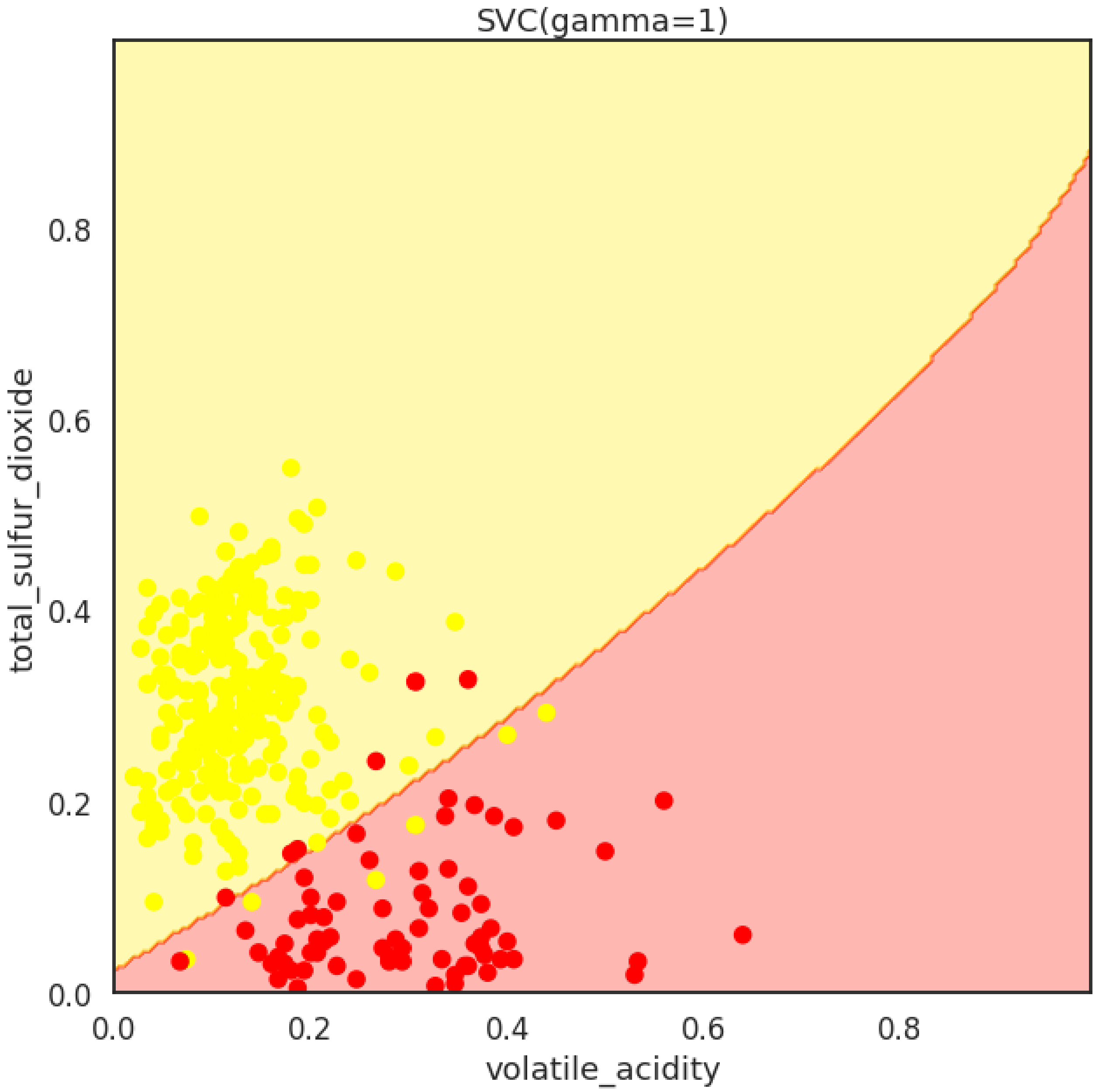
4. add on the scatter plot

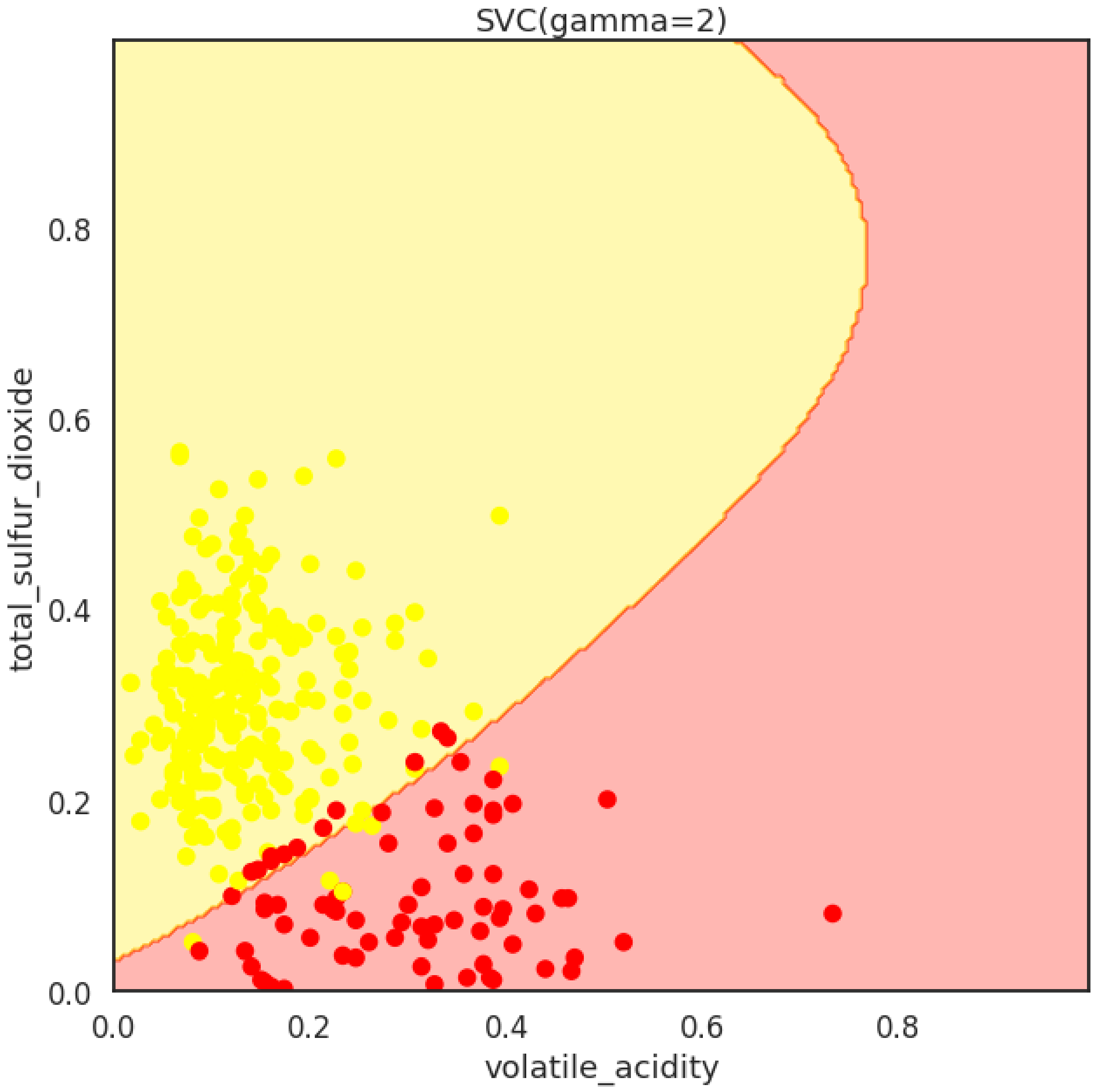
- After copying and pasting code, make sure the finished function uses your input `estimator` and not the LinearSVC model you built.
- For the following values of `gamma`, create a Gaussian Kernel SVC and plot the decision boundary.
  `gammas = [.5, 1, 2, 10]`
- Holding `gamma` constant, for various values of `C`, plot the decision boundary. You may try
  `Cs = [.1, 1, 10]`

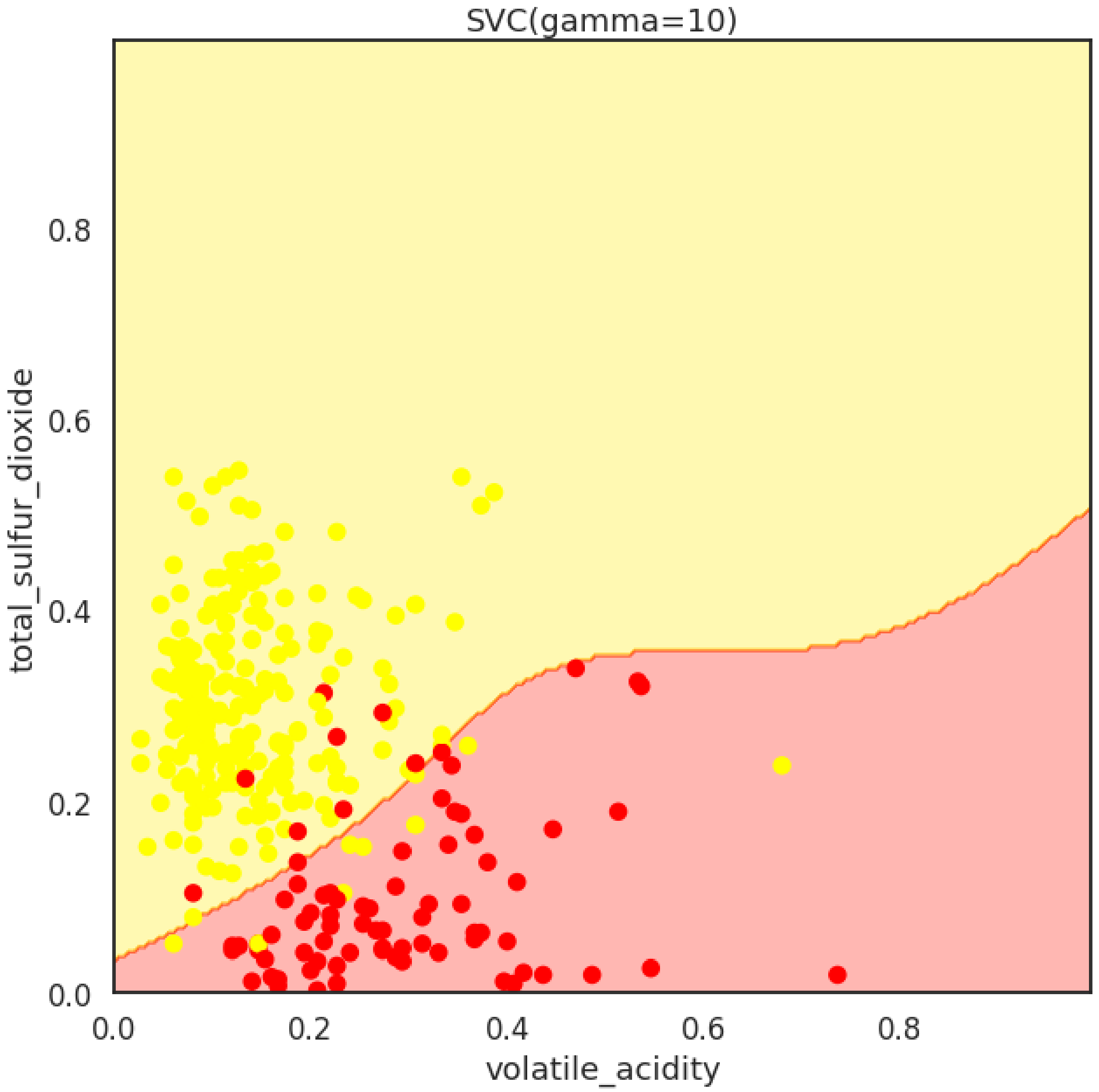
In [13]:
```python
def plotar_barreira_decisao(estimator, X, y):

    #fit um modelo baseado em dados de treino,X:data,y: target
    estimator.fit(X, y) #

    X_color = X.sample(300)
    y_color = y.loc[X_color.index]
    y_color = y_color.map(lambda r: 'red' if r == 1 else 'yellow')
    x_axis, y_axis = np.arange(0, 1, .005), np.arange(0, 1, .005)
    xx, yy = np.meshgrid(x_axis, y_axis)
    xx_ravel = xx.ravel()
    yy_ravel = yy.ravel()
    X_grid = pd.DataFrame([xx_ravel, yy_ravel]).T
    y_grid_predictions = estimator.predict(X_grid)
    y_grid_predictions = y_grid_predictions.reshape(xx.shape)

    fig, ax = plt.subplots(figsize=(10, 10))
    ax.contourf(xx, yy, y_grid_predictions, cmap=plt.cm.autumn_r, alpha=.3)
    ax.scatter(X_color.iloc[:, 0], X_color.iloc[:, 1], color=y_color, alpha=1)
    ax.set(
        xlabel=fields[0],
        ylabel=fields[1],
        title=str(estimator))
```

In [14]:
```python
from sklearn.svm import SVC

# Regularization
gammas = [.5, 1, 2, 10]
for gamma in gammas:
    SVC_Gaussian = SVC(kernel='rbf', gamma=gamma)
    plotar_barreira_decisao(SVC_Gaussian, X, y)
```


SVC(gamma=0.5)
total_sulfur_dioxide
volatile_acidity
0.0
0.2
0.4
0.6
0.8
0.0
0.2
0.4
0.6
0.8

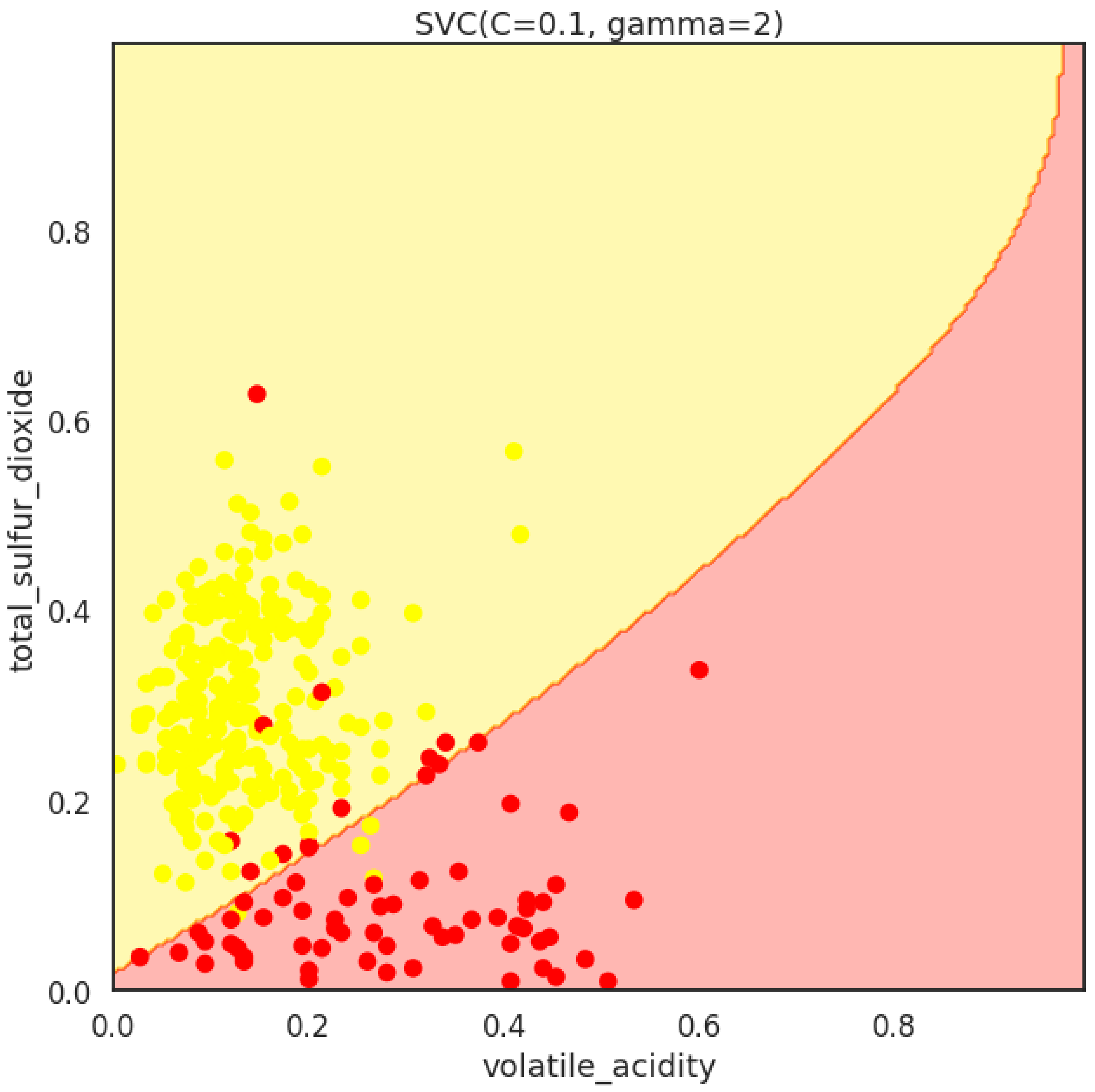

SVC(gamma=1)
total_sulfur_dioxide
volatile_acidity
0.0
0.2
0.4
0.6
0.8
0.0
0.2
0.4
0.6
0.8

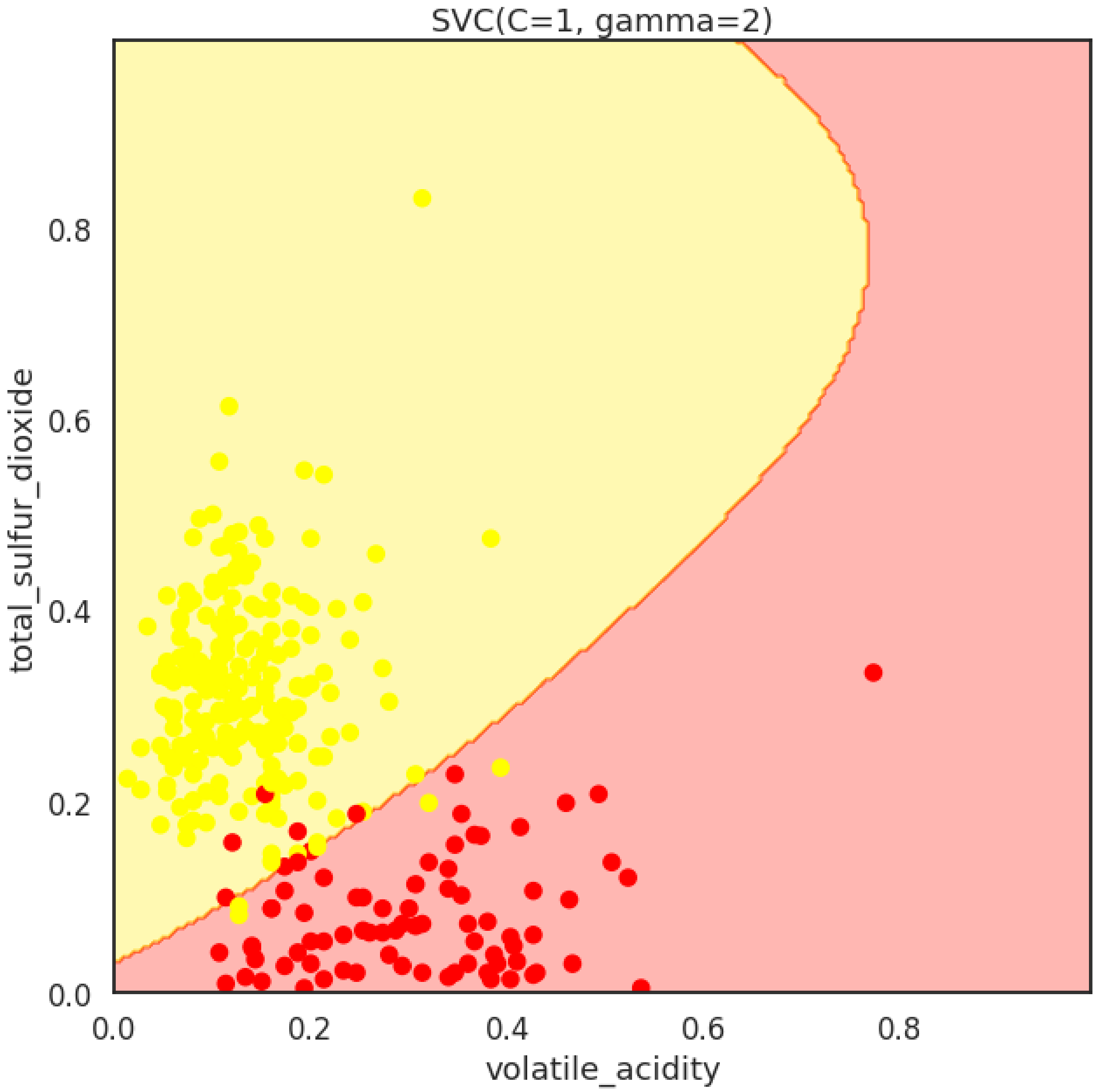

SVC(gamma=2)
total_sulfur_dioxide
0.0
0.2
0.4
0.6
0.8
volatile_acidity
0.0
0.2
0.4
0.6
0.8

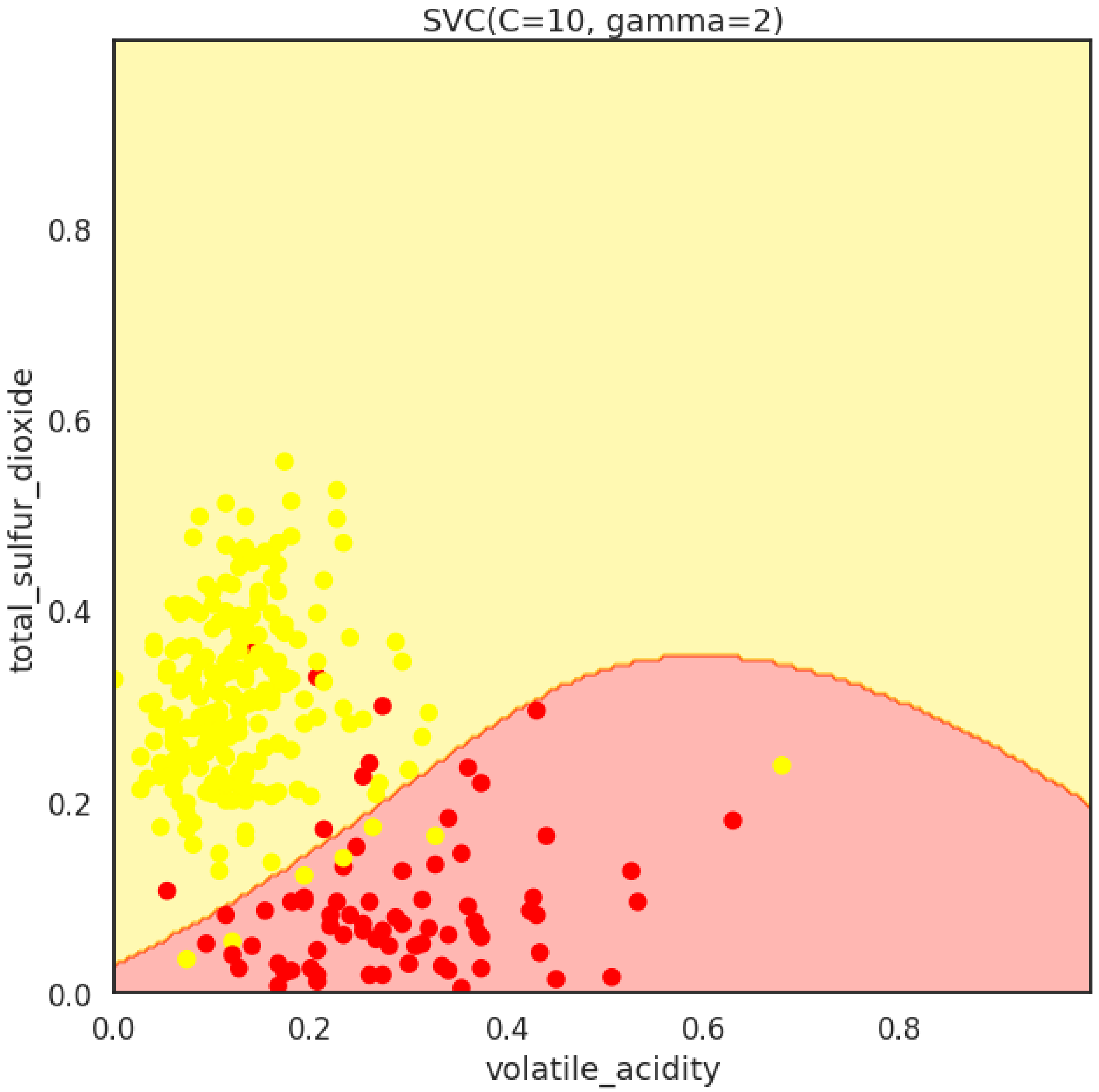


```
In [15]: Cs = [.1, 1, 10]
         for C in Cs:
```

```
    SVC_Gaussian = SVC(kernel='rbf', gamma=2, C=C)
    plotar_barreira_decisao(SVC_Gaussian, X, y)
```




SVC(C=1, gamma=2)
total_sulfur_dioxide
volatile_acidity
0.0
0.2
0.4
0.6
0.8


SVC(C=10, gamma=2)
total_sulfur_dioxide
volatile_acidity
0.0
0.2
0.4
0.6
0.8

Conclusões:

SVM é baseado na ideia de encontrar um hyperplano que melhor separa as features em domínios diferentes. Utilizado para problemas de classificação e regressão que possuem datasets pequenos pois o processamento é demorado.

As terminologias básicas utilizadas: Os pontos próximos ao hiperplano são chamados pontos de suporte do vetor e a distância dos vetores ao hiperplano chamamos margem. Basicamente, para encontrar o vetor w e escalar b que representam o hiperplano, devemos estimar os parâmetros w e b da fronteira de decisão a partir do conjunto de treino de forma a obter o hiperplano com margem máxima satisfazendo condições de linearidade:

$$\left.\begin{array}{ll} wx_i + b \geq 1 & se \quad y_i = 1 \\ wx_i + b \leq -1 & se \quad y_i = -1 \end{array}\right\} y_i(wx_i + b) \geq 1 \;\; (i = 1, 2, \cdots, N)$$

Pelos gráficos acima, podemos notar que nosso problema não é linearmente separável. E podemos separá-los pelo SVM por meio de uma dimensão maior. Criando um diferente sistema de coordenadas e fazer as transformações de um espaço para outro na coordenadas dos vetores. Isso se faz pelo produto escalar entre os pares dos vetores. O Kernel é uma maneira de computar o produto escalar entre dois vetores x e y em uma dimensão maior no espaço, também chamado de "generalized dot product", em inglês.

Na questão 3 foi utilizado o Gaussian Kernel SVC o que garante um otimal glogal preditor que minimiza ambas estimativas e erros de aproximação do classificador.

Create a Gaussian Kernel SVC and plot the decision boundary. `gammas = [.5, 1, 2, 10]`

O paramêtro gama define quão distante a influência de um exemplo simples de treino alcança. Com valores pequenos valores significando "longe" e altos valores "perto". Podem também serem considerados como o inverso do raio de influência das amostras selecionadas pelo vetor de suporte. Em outras palavras, o gamma pode ser visto como o inverso do desvio padao do RBF Kernel, logo, pequenos gamas definem uma funão gaussiana com alta variância, neste caso, dois pontos podem ser considerados similares mesmo longe entre si. Por outro lado, alto gamma indica baixa variância e logo dois pontos podem ser considerados similares se estão próximos entre si. Como podemos ver nos gráficos acima condizennte com os gammas descritos abaixo.

1. .5
2. 1
3. 2
4. 10

Question 4

In this question, we will compare the fitting times between SVC vs Nystroem with rbf kernel.

Jupyter Notebooks provide a useful magic function **`%timeit`** which executes a line and prints out the time it took to fit. If you type **`%%timeit`** in the beginning of the cell, then it will run the whole cell and output the running time.

- Re-load the wine quality data if you made changes to the original.
- Create `y` from data.color, and `X` from the rest of the columns.
- Use `%%timeit` to get the time for fitting an SVC with rbf kernel.
- Use `%%timeit` to get the time for the following: fit_transform the data with Nystroem and then fit a SGDClassifier.

Nystroem+SGD will take much shorter to fit. This difference will be more pronounced if the dataset was bigger.

- Make 5 copies of X and concatenate them
- Make 5 copies of y and concatenate them
- Compare the time it takes to fit the both methods above

In [16]:

```
from sklearn.kernel_approximation import Nystroem
from sklearn.svm import SVC
from sklearn.linear_model import SGDClassifier

y = data.color == 'red'
X = data[data.columns[:-1]]

kwargs = {'kernel': 'rbf'}
svc = SVC(**kwargs)
nystroem = Nystroem(**kwargs)
sgd = SGDClassifier()
```

In [17]:

```
%%timeit
svc.fit(X, y)
```

```
485 ms ± 80.1 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [18]:
```
%%timeit
X_transformed = nystroem.fit_transform(X)
sgd.fit(X_transformed, y)
```

```
254 ms ± 12.5 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [19]:
```
X2 = pd.concat([X]*5)
y2 = pd.concat([y]*5)

print(X2.shape)
print(y2.shape)
```

```
(32485, 12)
(32485,)
```

In [20]:
```
%timeit svc.fit(X2, y2)
```

```
12.8 s ± 646 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [21]:
```
%%timeit
X2_transformed = nystroem.fit_transform(X2)
sgd.fit(X2_transformed, y2)
```

```
558 ms ± 30.5 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```