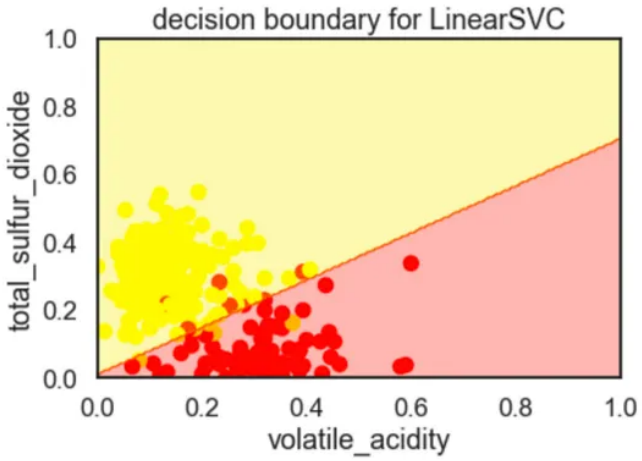
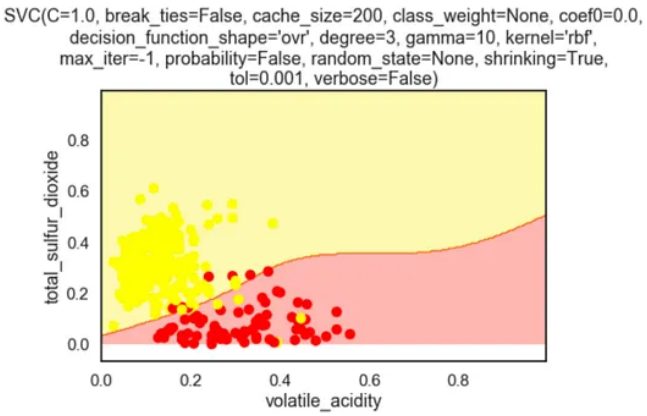
Using Different Types of Kernel in Support Vector Machines

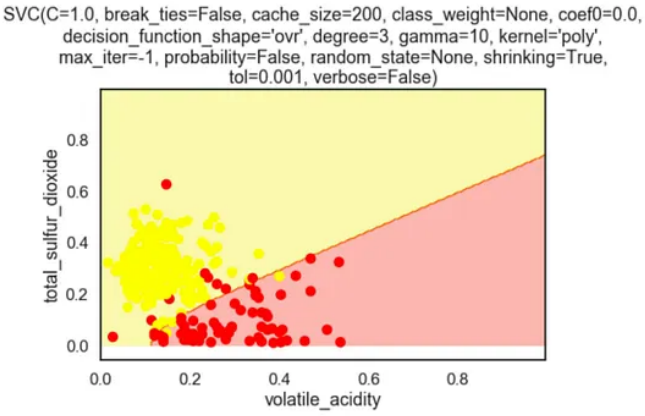
A

Abhishek Gupta · Follow

7 min read · Aug 26, 2020







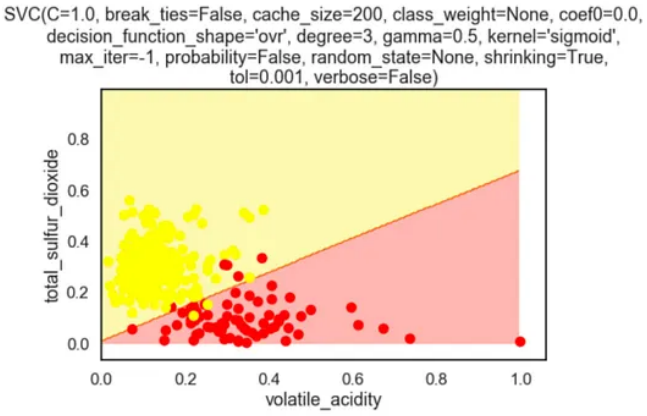


fig 1. Different Types of kernel in SVM

In my previous blog I had explained about Support Vector Machine(SVM). As we know that there are many types of kernel, but the main goal of this post is to see how beautifully the decision boundary changes after applying different types of kernel (fig 1). And at last we will see the classification report. In this post we will take a close look at Linear SVC, Gaussian rbf kernel SVC, Polynomial kernel SVC and at last Sigmoid kernel SVC and also how to plot and visualise all this kernels.

Along with the code let's see what is the mathematical term of each kernel.

Before we begin let's see the dataset. In this I have use "Wine_Quality" dataset which 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 of wine i.e. red or white.

First let's import essential modules of python i.e. "Numpy", "Pandas", "Matplotlib", "Seaborn".

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

Now after importing the data we will be creating the target variable "y"as a 1|0 column where 1 means red. The fields below in code will contain all columns except "color".

```
filepath = 'Wine_Quality_Data.csv'
data = pd.read_csv(filepath)

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

Now plotting a "bar plot" showing the correlations between each column and y, after visualizing this plot we will pick the two most correlated fields using the absolute value of correlations.

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

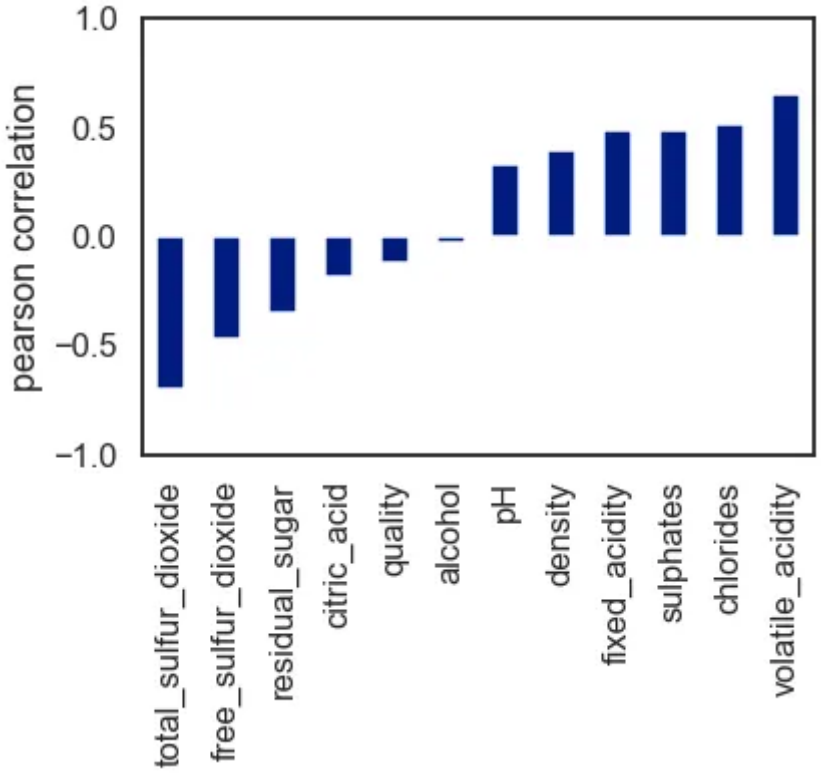


fig 2

As from above (Fig 2). we can see that "total_sulfur_dioxide" and "volatile_acidity" are highly correlated. Now using "MinMaxScaler" to scale X. Since this will return us output in the form of np.array([]) hence we will prepare a DataFrame again and rename the columns appropriately.

```
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])
```

Now before moving to Linear SVC lets see its mathematical formula to learn what is actual happening with our decision boundary.

Linear SVC classifier:

$y = 0$ *if* $w^T x + b < 0$,

$y = 1 \text{ if } w^{\intercal}x + b \geq 0$

Fit the X and Y parameter to support vector machine classifier. Since our original dataset is too large and it produces a crowded plot, we will be picking only 300 samples and corresponding y value to store them in their respective variables.

```
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 'blue')
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');
```

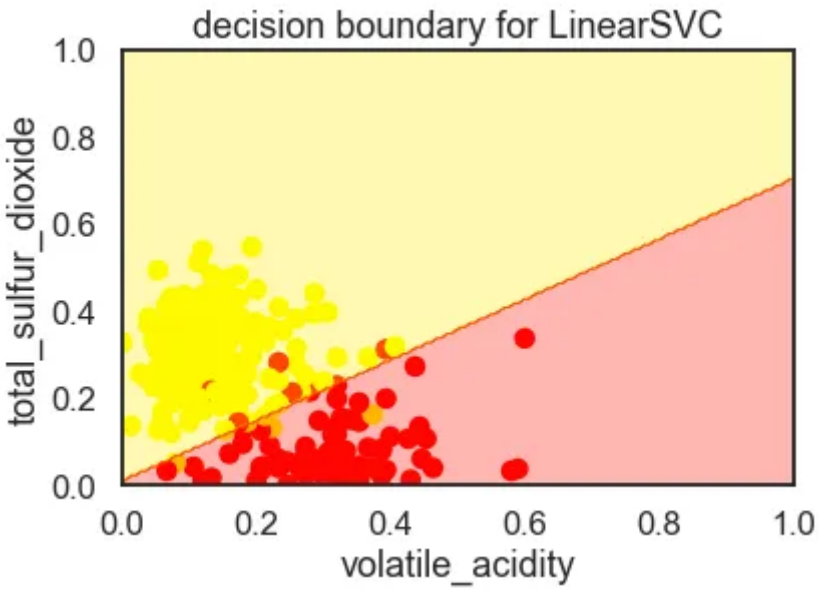


fig 3.

From fig 3. we come to know that the decision boundary is almost at best fit for all X and Y plotted. Now keeping our all X and Y same let us take a look at how the decision boundary changes after applying the "Gaussian rbf kernel".

Gaussian rbf kernel:

Mathematical form of Gaussian RBF : $K(a, b) = exp(-\gamma\|(a - b)^2\|)$

```
def plot_decision_boundary(estimator, X, y):
    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 'blue')
    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=(8, 5))
    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))
```

Plotting the graph for gamma = [.5, 1, 2, 10]

```
from sklearn.svm import SVC

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

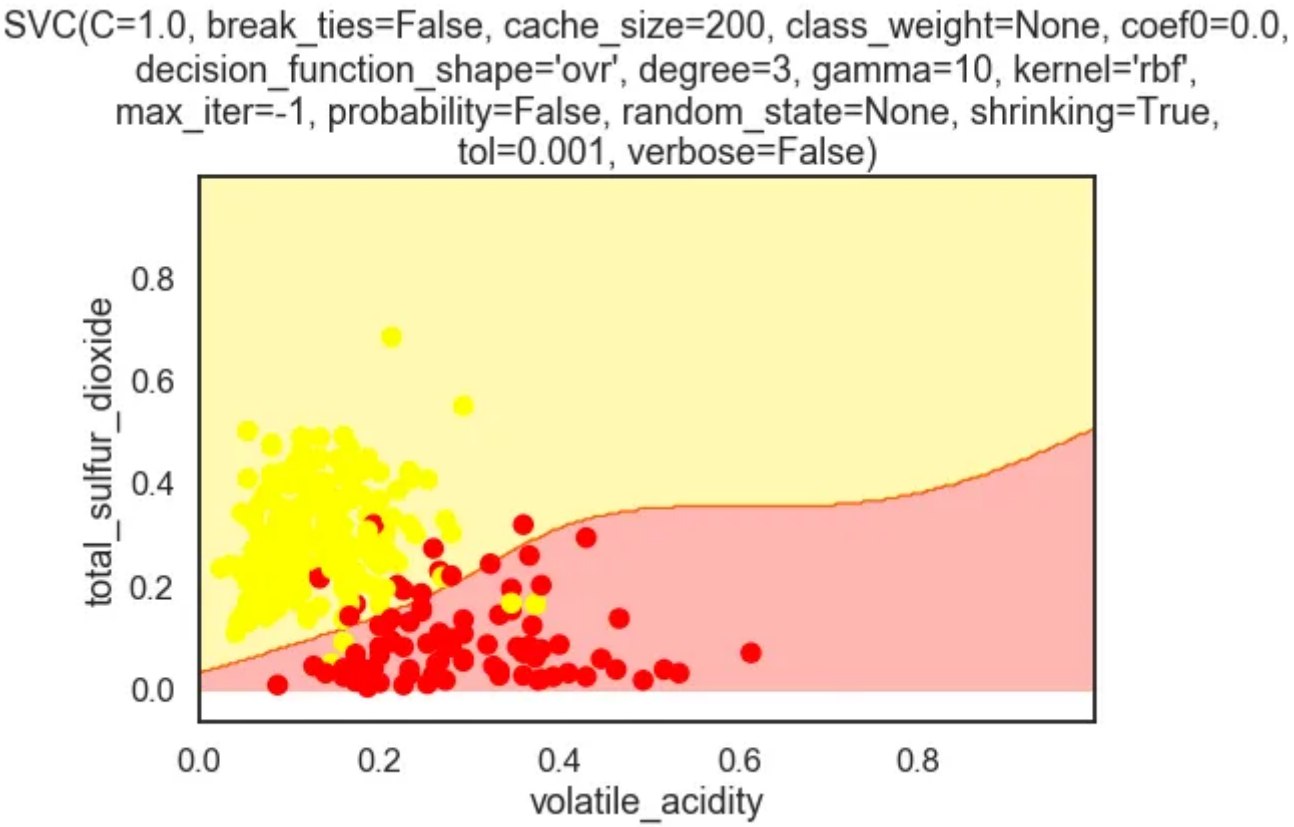


fig 4.

Note, after this there are a total four graphs plotted but for the blog I am picking accurate one.

The plots show models trained with different values of hyperparameters gamma (γ) and C. Increasing gamma makes the bell-shaped curve narrower fig 4. As a result, each instance's range of influence is smaller, the decision boundary ends up being more irregular, wiggling around individual instances. Conversely, a small gamma value makes the bell shaped curve wider instances have a larger range of influence, and the decision boundary ends up smoother. So γ acts like a regularization hyperparameter. If your model is overfitting, you should reduce it, if it is underfitting, you should increase it similar to the C hyperparameter.

Holding gamma constant, for various values of C, plot the decision boundary
CS = [0.1, 1, 10]

```
Cs = [.1, 1, 10]
for C in Cs:
    SVC_Gaussian = SVC(kernel='rbf', gamma=2, C=C)
    plot_decision_boundary(SVC_Gaussian, X, y)
```

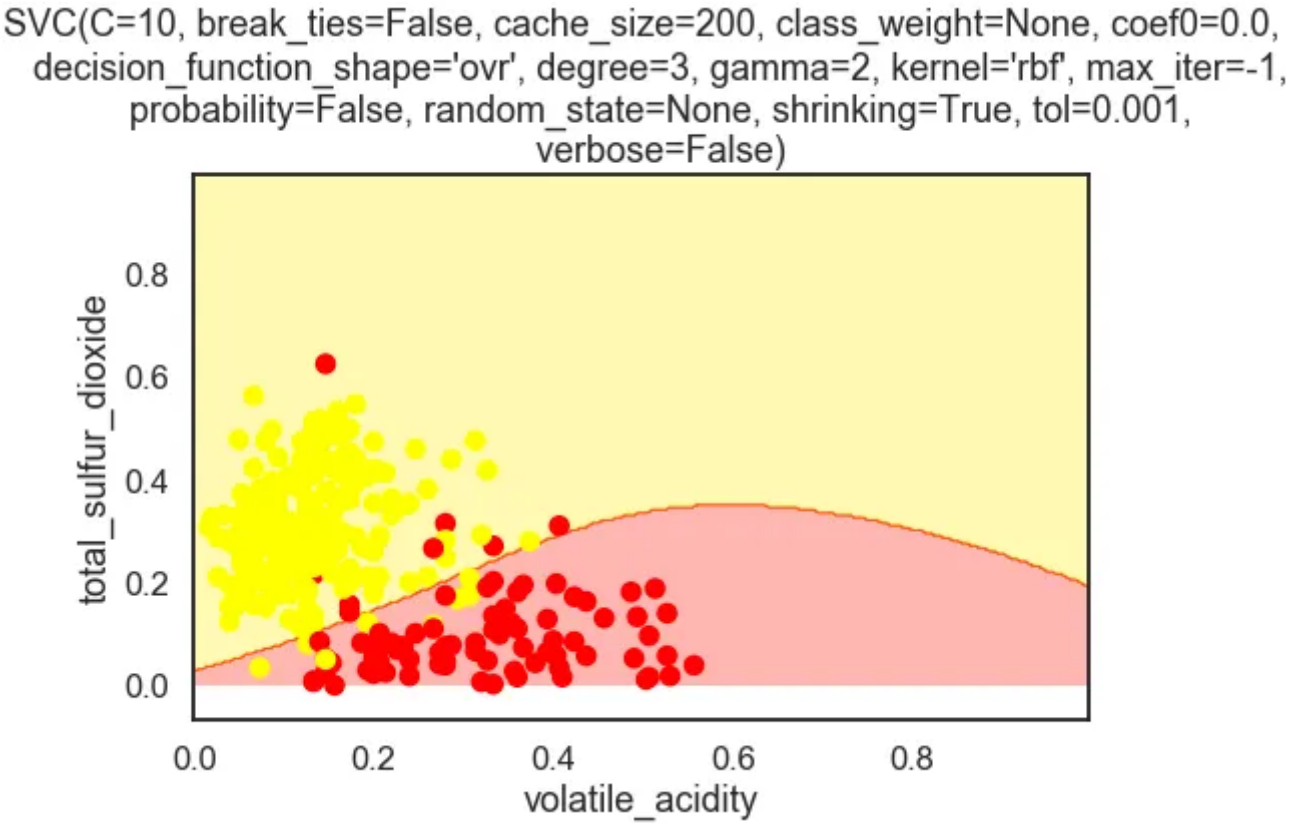


fig 5.

Let us train_test_split our data in order to check our accuracy of rbf kernel and print the best print the best_params, best_estimator.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= 0.30, random_state = 45)

from sklearn.model_selection import GridSearchCV

param_grid = {'C': [0.1, 1, 10],
              'gamma': [10, 2, 1, 0.5],
              'kernel': ['rbf']}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)
grid.fit(X_train, y_train)

print(grid.best_params_)
print(grid.best_estimator_)
```

```
In [16]: print(grid.best_params_)

         print(grid.best_estimator_)

         {'C': 10, 'gamma': 10, 'kernel': 'rbf'}
         SVC(C=10, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
             decision_function_shape='ovr', degree=3, gamma=10, kernel='rbf',
             max_iter=-1, probability=False, random_state=None, shrinking=True,
             tol=0.001, verbose=False)
```

fig 6.

As in above (fig 6.) we can observe that for gaussian rbf the best parameters are C:10, gamma: 10.

```
grid_predictions = grid.predict(X_test)

print(classification_report(y_test, grid_predictions))
```

```
In [17]: grid_predictions = grid.predict(X_test)

         print(classification_report(y_test, grid_predictions))

                       precision    recall  f1-score   support

                    0       0.97      0.97      0.97      1447
                    1       0.92      0.90      0.91       503

             accuracy                           0.95      1950
            macro avg       0.94      0.94      0.94      1950
         weighted avg       0.95      0.95      0.95      1950
```

fig 7.

From fig 7. we can see our "classification _report" in which our accuracy is pretty good i.e. 95%.

Polynomial Kernel SVC:

So now after this let us take a close look how our decision boundary changes in the Polynomial kernel. Using the plot decision boundary function from the above code and degree = 3.

Mathematical form of Polynomial Kernel : K (a, b) = (γ(a)^T*b + r)^ d

```
from sklearn.svm import SVC

gammas = [0.5, 1, 2, 10]
for gamma in gammas:
    SVC_Polynomial = SVC(kernel='poly', gamma=gamma)
    plot_decision_boundary(SVC_Polynomial, X, y)
```

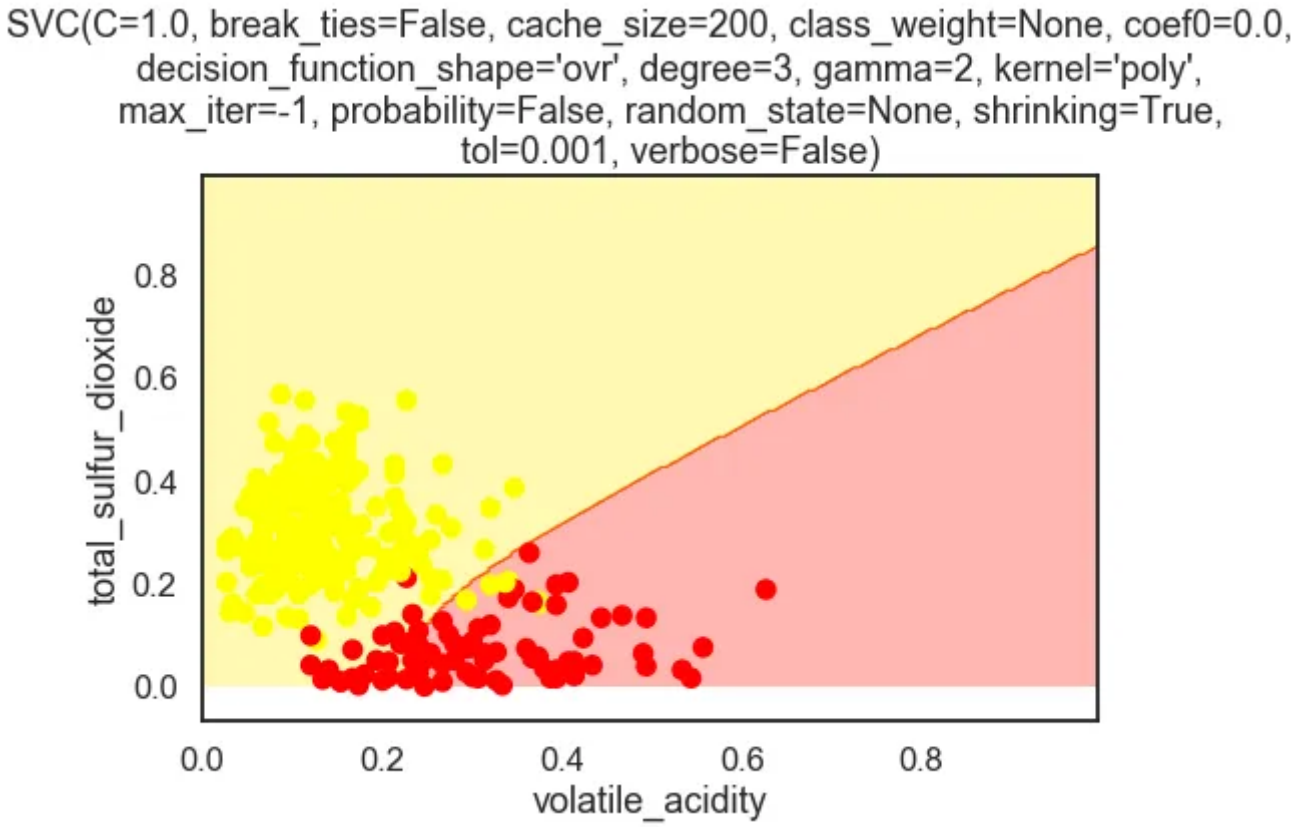


fig 8.

Now plotting by holding gamma constant, for various values of C, plot the decision boundary

CS = [10, 20, 100, 200]

```
Cs = [10,20,100,200]
for C in Cs:
    SVC_Polynomial = SVC(kernel='poly', gamma=2, C=C)
    plot_decision_boundary(SVC_Polynomial, X, y)
```

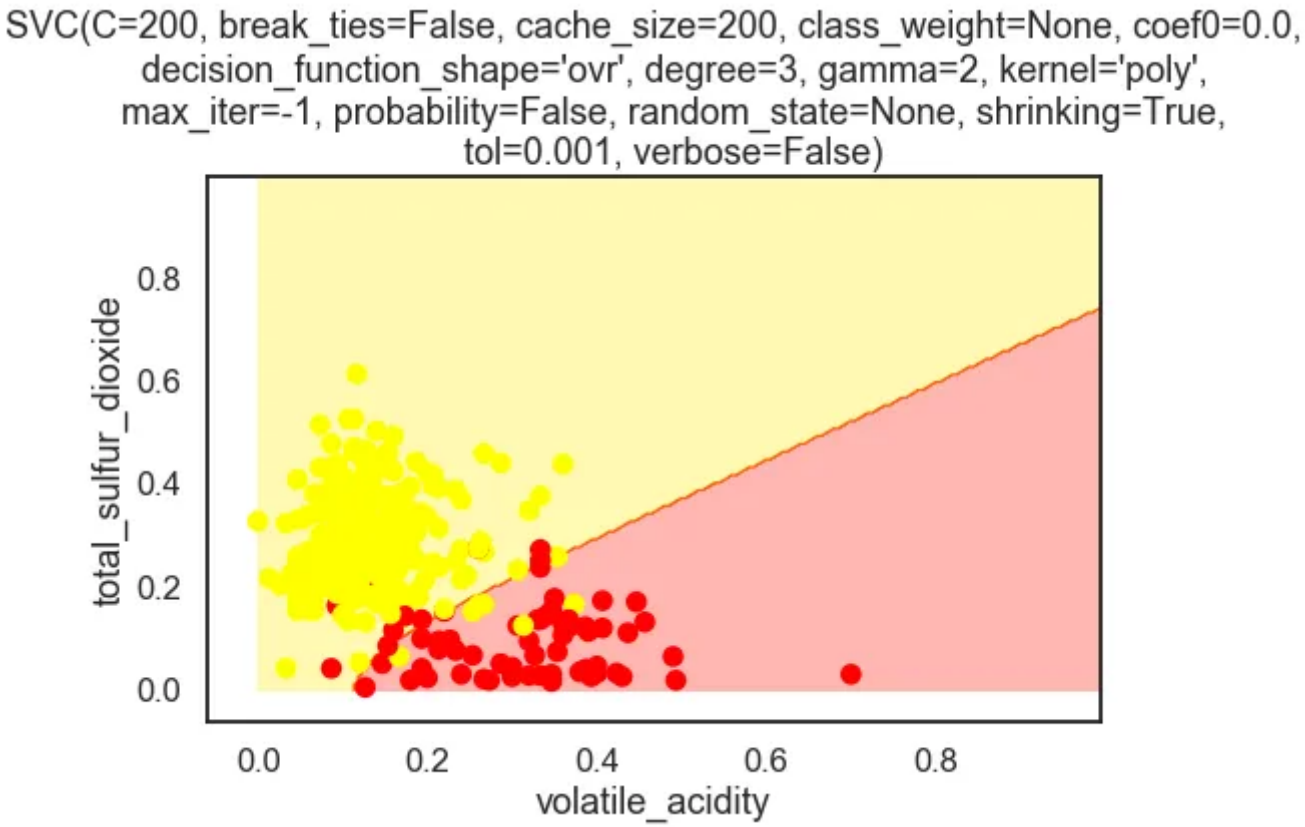


fig 9.

Let us check our accuracy and classification_report of polynomial kernel.

```
param_grid = {'C': [10, 20, 100, 200],
              'gamma': [10, 2, 1, 0.5],
              'kernel': ['poly']}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)

grid.fit(X_train, y_train)

print(grid.best_params_)

print(grid.best_estimator_)
```

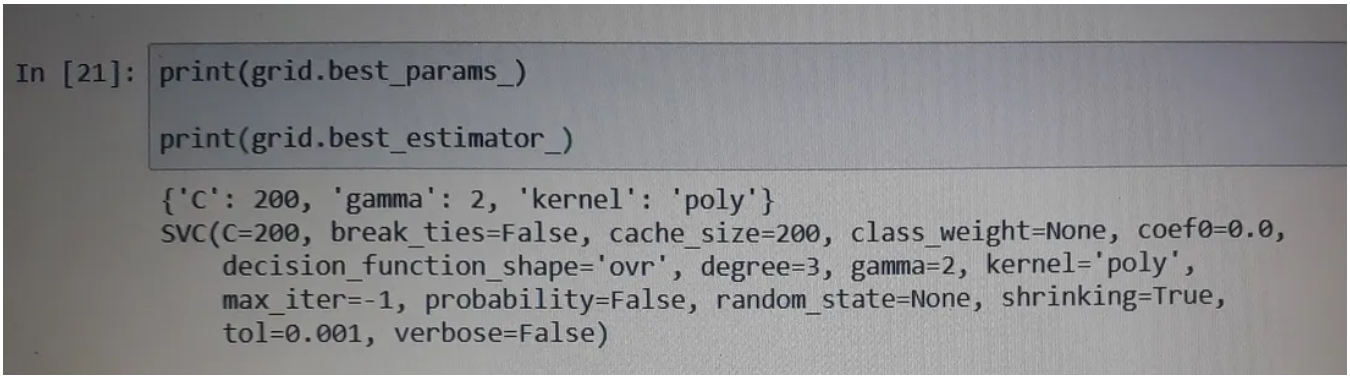


fig 10.

```
grid_predictions = grid.predict(X_test)

print(classification_report(y_test, grid_predictions))
```

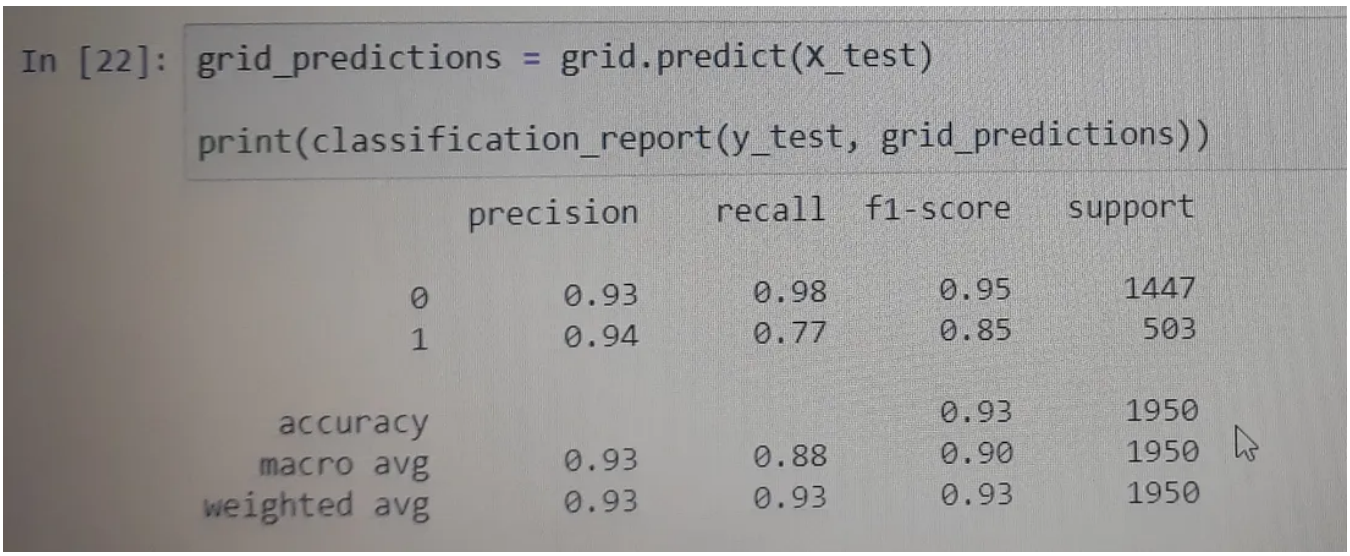


fig 11.

As we can see in (fig 11.) that here our accuracy is 93% and being dropped by 2% compared to Gaussian rbf kernel (fig 11.). Also best estimated gamma is 2 and C value is 200 in (fig 10.).

At last we will now see how the decision boundary changes while using sigmoid kernel.

Sigmoid kernel SVC:

*Mathematical form of Sigmoid kernel : K (a, b) = tanh (γ(a)^T*b + r)*

```
gammas = [0.5, 1, 2, 10]
for gamma in gammas:
    SVC_Sigmoid = SVC(kernel='sigmoid', gamma=gamma)
    plot_decision_boundary(SVC_Sigmoid, X, y)
```

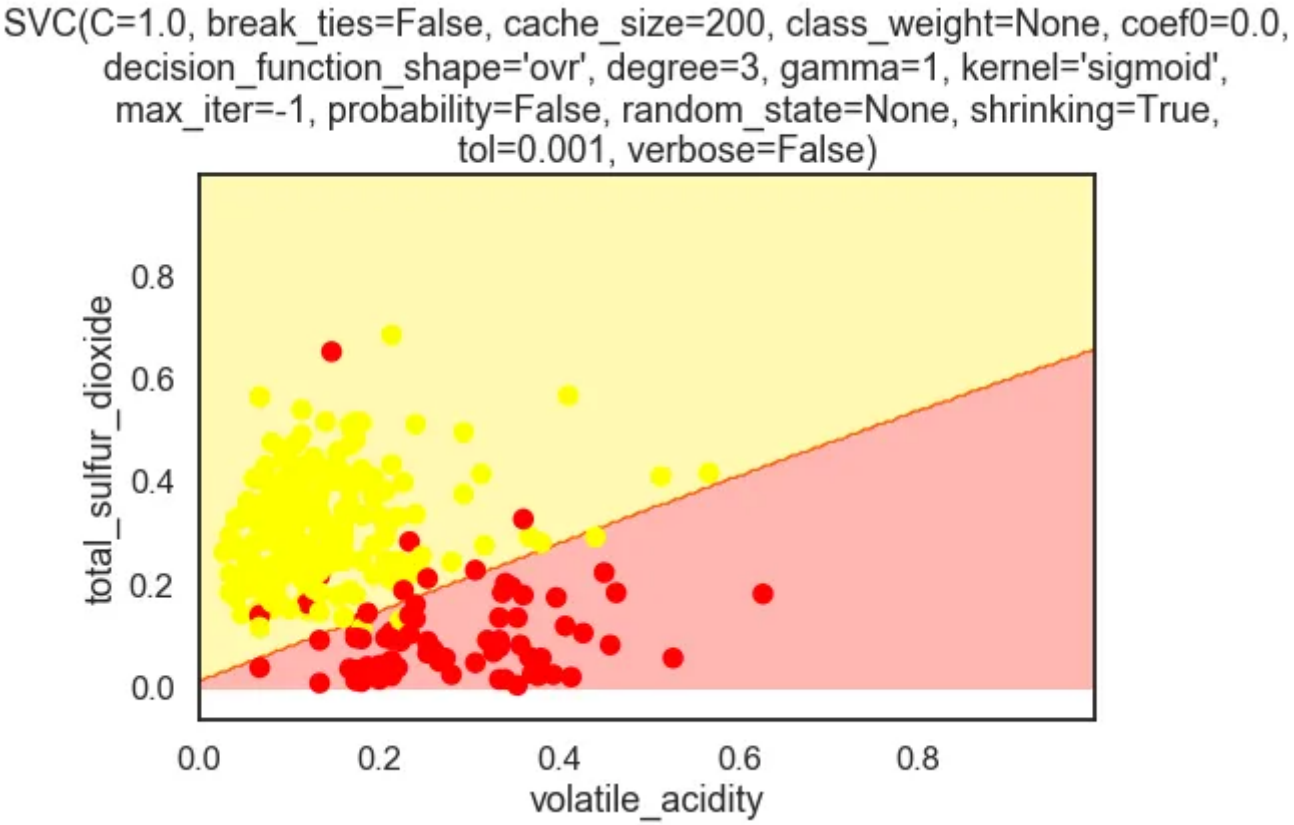


fig 12.

```
Cs = [10,20,100,200]
for C in Cs:
    SVC_Sigmoid = SVC(kernel='sigmoid', gamma=2, C=C)
    plot_decision_boundary(SVC_Sigmoid, X, y)
```

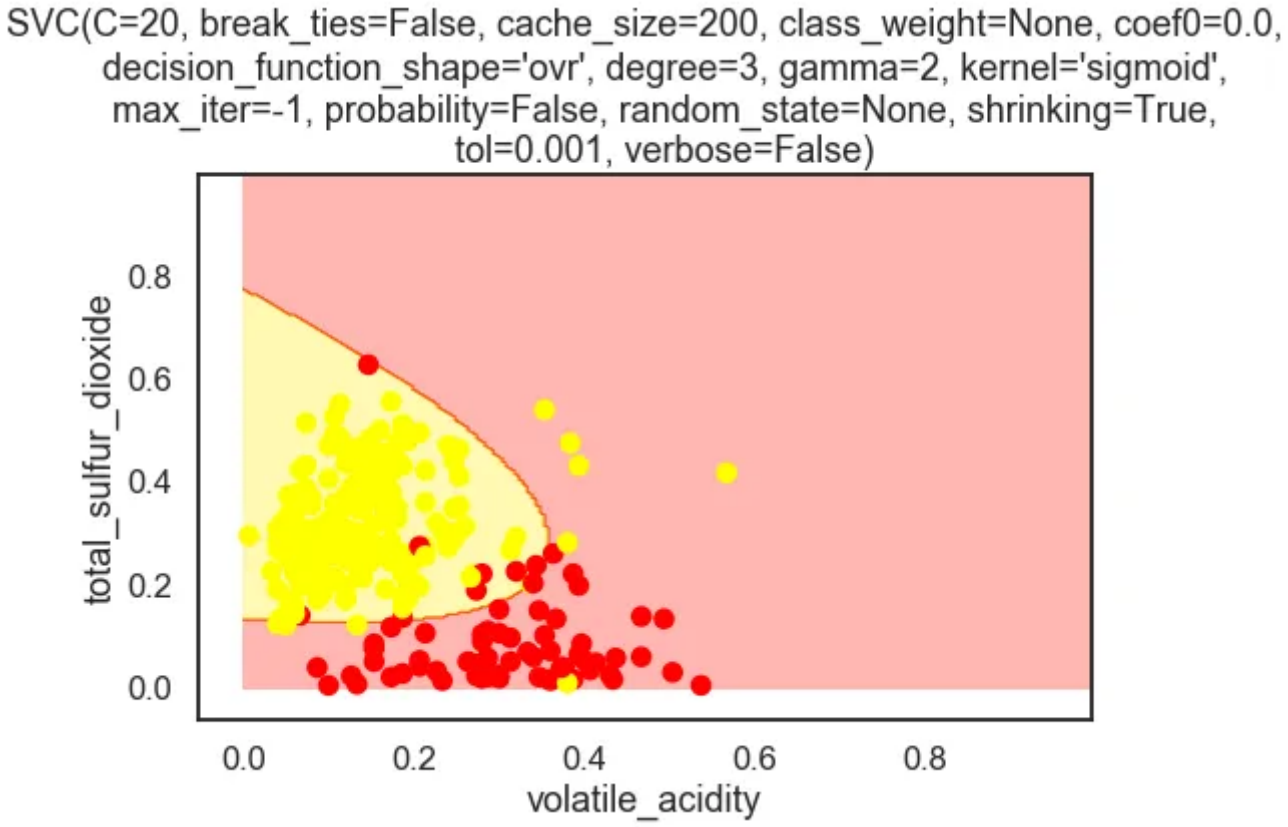


fig 13.

Let us now see what is the accuracy and classification_report of the sigmoid kernel.

```
param_grid = {'C': [10, 20, 100, 200],
              'gamma': [10, 2, 1, 0.5],
              'kernel': ['sigmoid']}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)

grid.fit(X_train, y_train)

print(grid.best_params_)

print(grid.best_estimator_)
```

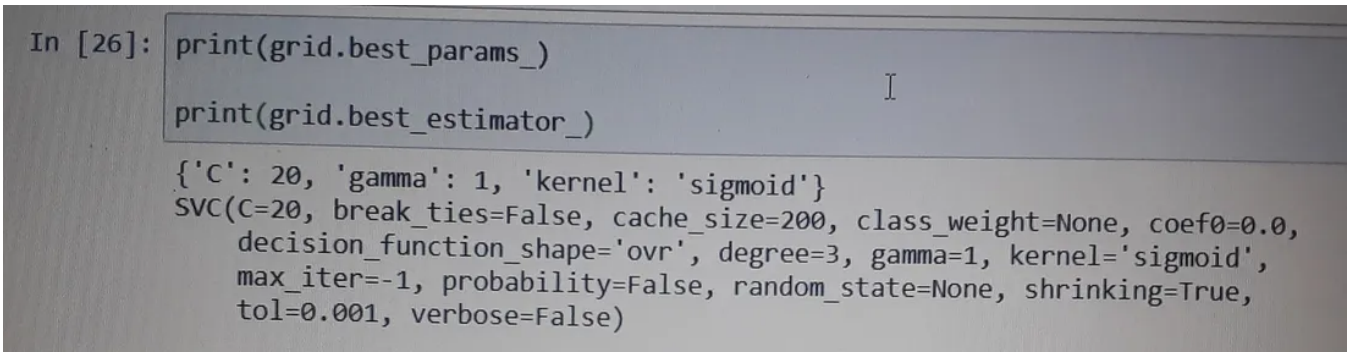


fig 14.

```
grid_predictions = grid.predict(X_test)

print(classification_report(y_test, grid_predictions))
```

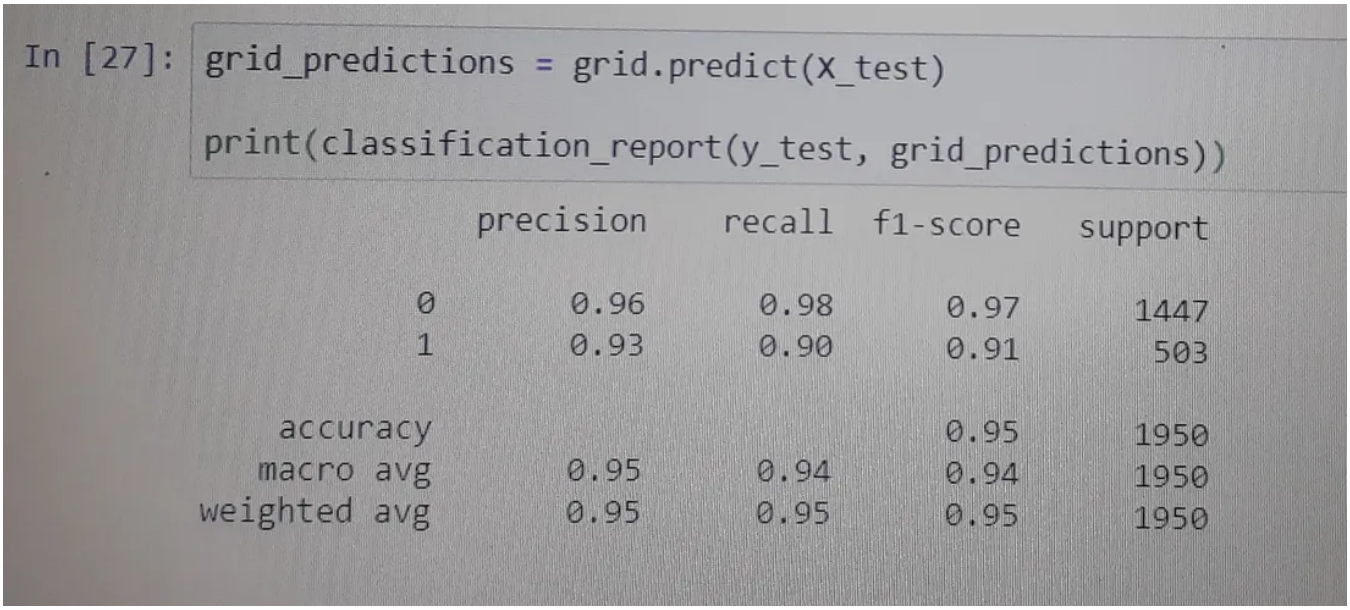


fig 15.

Ok so now in this we again get the accuracy of 95%. With the best parameter as C=20 and gamma = 1.

Conclusion:

So in this post we conclude with how decision boundaries change when we apply different types of kernel. And also we saw the mathematical formula of each but to completely understand the mathematics behind this I would suggest go and read books there are n numbers of books available in market. Then we compared the accuracy of each kernel in which all were pretty good.

For Source Code please visit >>

https://github.com/Abhishek0917/Studious_ML/blob/master/Support%20Vector%20Machine%20and%20Kernel%20Types%20(2).ipynb

Thank You!!