<u>**Machine Learning Lab 7**</u>

Support Vector Machines

Decision A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimentional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side. Basically support vector machine works on the principle of separation of classes. What SVM does is, it finds out a line/hyper-parameter(in multidimensional space) that separates out classes.

The dataset

The dataset used to perform this experiment is the wine quality dataset, it is a combination of data on two types of wine variants, namely red wine and white wine, of the portuguese “Vinho Verde” wine. The dataset contains information on the parameters for fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol.

Experiment

In this experiment I used the sklearn’s support vector classifier (SVC) algorithm to predict the quality of a wine.

Using the pandas library in I loaded the red wine and white wine datasets into the memory from their respective csv files and then merged the two datasets into one single pandas dataframe.

Using the pandas.Dataframe.describe() function in pandas I calculated the various statistical measures of each of the columns of the dataset.

For performing the experiment I started with plotting the scatter plot for each of the features in the dataset with every other feature, this helped to find if there were any features which were linearly separable. In the case of my dataset they were not.

Next I divided the dataset into training and testing portions using the train_test_split functionality in sklearn

I setup the grid search for SVC with RBF(Radial Basis function) kernel and the value of gamma ranging from 1e-4 to 0.5, and the value of parameter C ranging from 1 to 1000. I ran the grid search and for the parameters of C=1, and gamma =0.5, I achieve the best performance with a precision score of 0.84.

Next for visualization purposes I selected two most important features from the dataset. In this case the feature importance was decided based on the feature importance values calculated in the previous lab assignments using random forests.

Thus the two features of fixed acidity and volatile acidity were selected and an SVC was trained on this model and the decision boundaries were predicted on the same.

The code and plots can be found in the accompanying jupyter notebook.

Support Vector Machines

November 1, 2018

1 Lab 7

2 Support Vector Machines

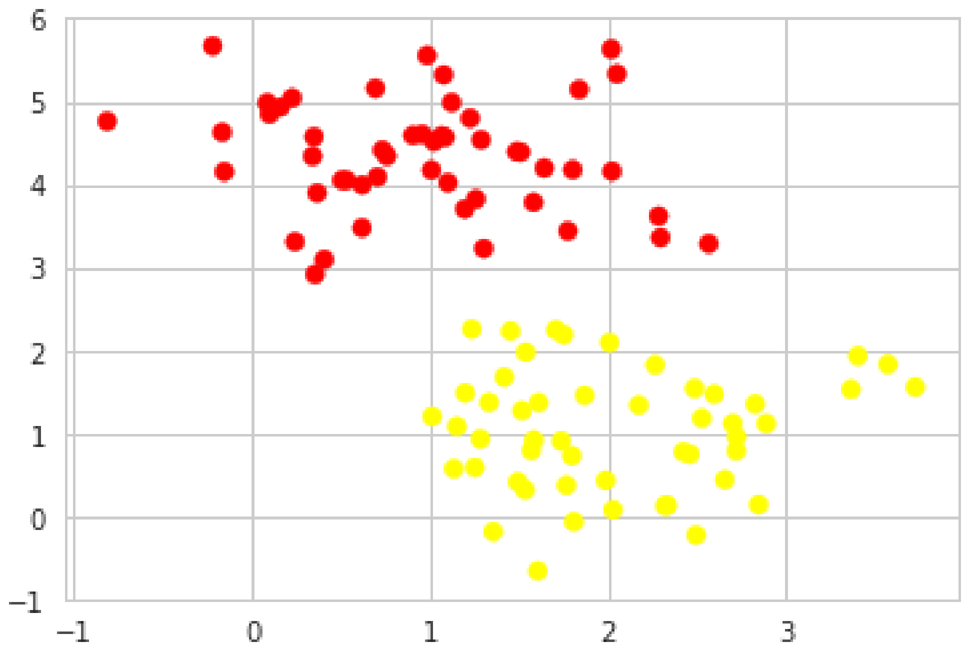
2.1 Submitted to: Prof. Sweetlin Hemlatha

2.2 Submitted by: Prateek Singh (15BCE1091)

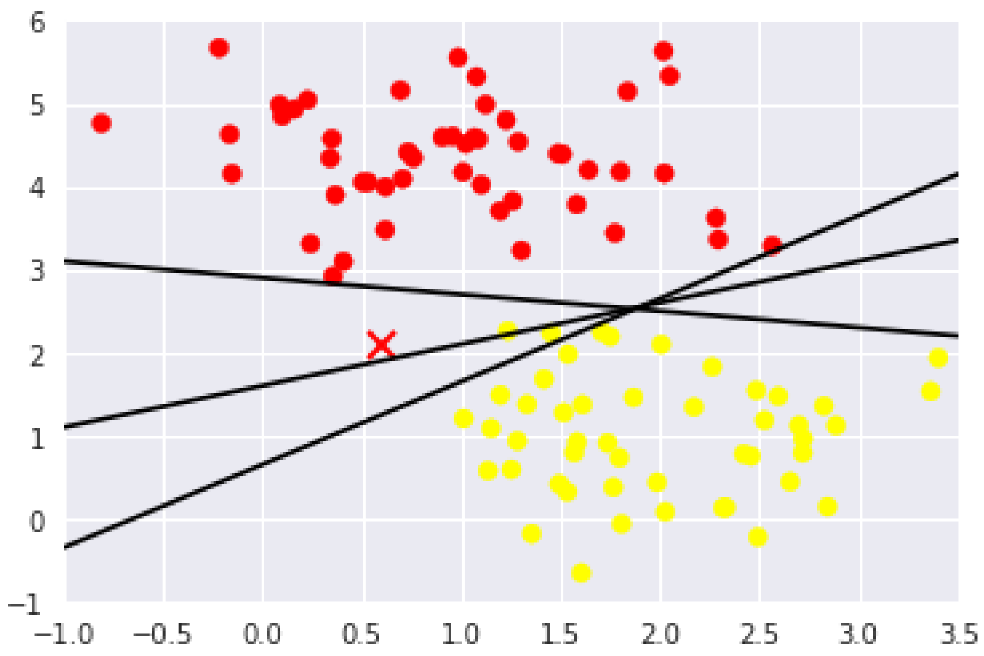
```
In [16]: import numpy as np
         import pandas as pd
         import seaborn as sns
         from scipy import stats
         from sklearn.svm import SVC
         from sklearn.utils import shuffle
         from sklearn.metrics import classification_report
         from sklearn.model_selection import train_test_split
         from sklearn.model_selection import GridSearchCV
         from sklearn.datasets.samples_generator import make_blobs

         import matplotlib.pyplot as plt
         from matplotlib.colors import ListedColormap
         %matplotlib inline

         sns.set(style='whitegrid', context='notebook', font_scale=1)
```

```
In [2]: X, y =  make_blobs(n_samples=100, centers=2, random_state =0, cluster_std=0.70)
        plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
```

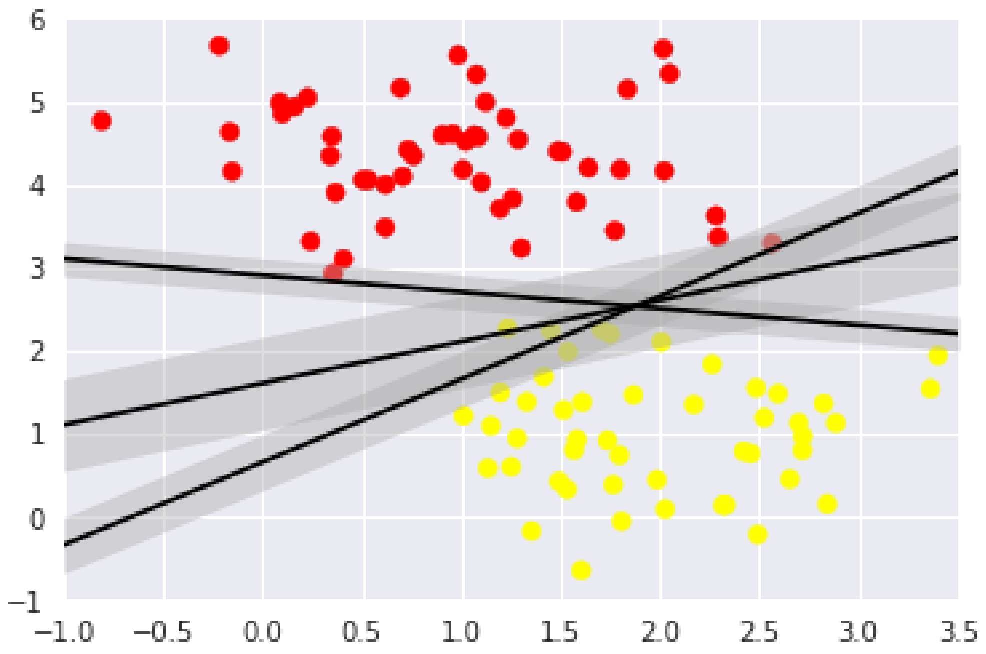
```
Out[2]: <matplotlib.collections.PathCollection at 0x7fd48abf7e10>
```



```
In [3]: xfit = np.linspace(-1, 3.5)
        plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
        plt.plot([0.6], [2.1], 'x', color='red', markeredgewidth=2, markersize=10)

        for m, b in [(1, 0.65), (0.5, 1.6), (-0.2, 2.9)]:
            plt.plot(xfit, m*xfit +b, '-k')

        plt.xlim(-1, 3.5);
```

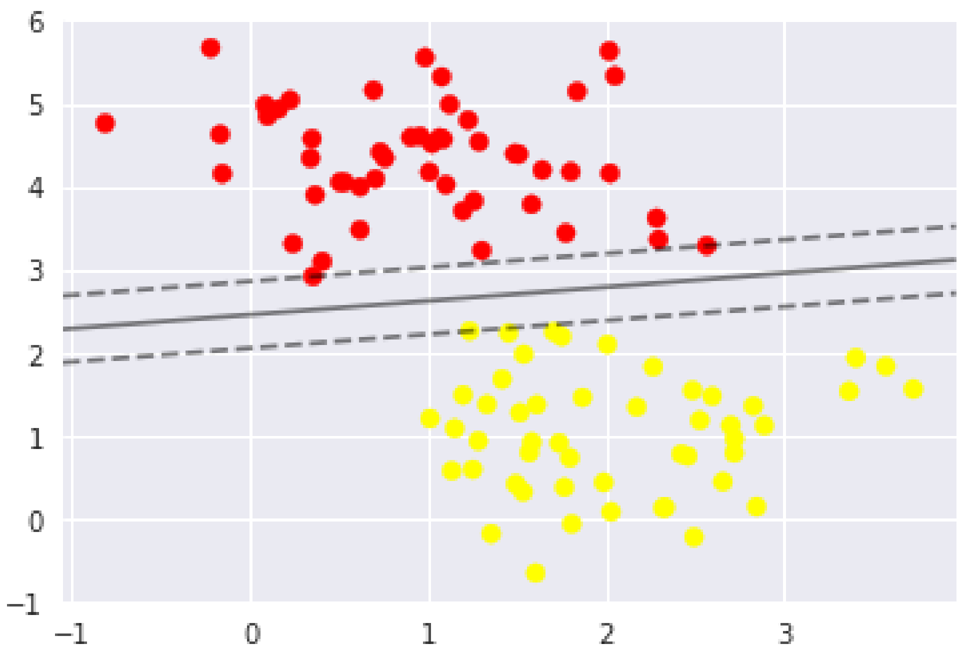


```
In [4]: xfit = np.linspace(-1, 3.5)
        plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')

        for m, b, d in [(1, 0.65, 0.33), (0.5, 1.6, 0.55), (-0.2, 2.9, 0.2)]:
            yfit = m*xfit + b
            plt.plot(xfit, yfit, '-k')
            plt.fill_between(xfit, yfit-d, yfit+d, edgecolor='none', color='#AAAAAA', alpha=0.

        plt.xlim(-1, 3.5);
```

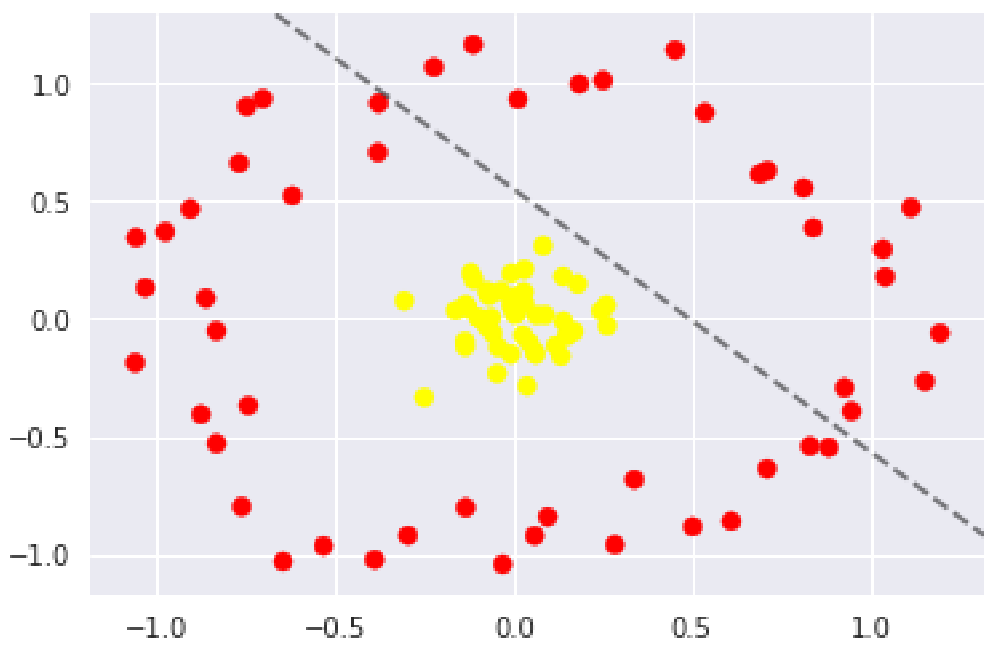


```
In [5]: from sklearn.svm import SVC
        model = SVC(kernel='linear', C=1E10)
        model.fit(X, y)

Out[5]: SVC(C=10000000000.0, cache_size=200, class_weight=None, coef0=0.0,
          decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
          max_iter=-1, probability=False, random_state=None, shrinking=True,
          tol=0.001, verbose=False)

In [6]: def plot_svc_decision_function(model, ax=None, plot_support=True):
            """Plot the decision function for a 2D SVC"""
            if ax is None:
                ax = plt.gca()
            xlim = ax.get_xlim()
            ylim = ax.get_ylim()

            #create grid to evaluate
            x = np.linspace(xlim[0], xlim[1], 30)
            y = np.linspace(ylim[0], ylim[1], 30)
            Y, X = np.meshgrid(y, x)
            xy = np.vstack([X.ravel(), Y.ravel()]).T
            P = model.decision_function(xy).reshape(X.shape)

            #plot decision boundaries and margins
            ax.contour(X, Y, P, colors='k', levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-
            #plot support vector machines
            if plot_support:
                ax.scatter(model.support_vectors_[:,0], model.support_vectors_[:, 1], s=300, l
            ax.set_xlim(xlim)
            ax.set_ylim(ylim)
```

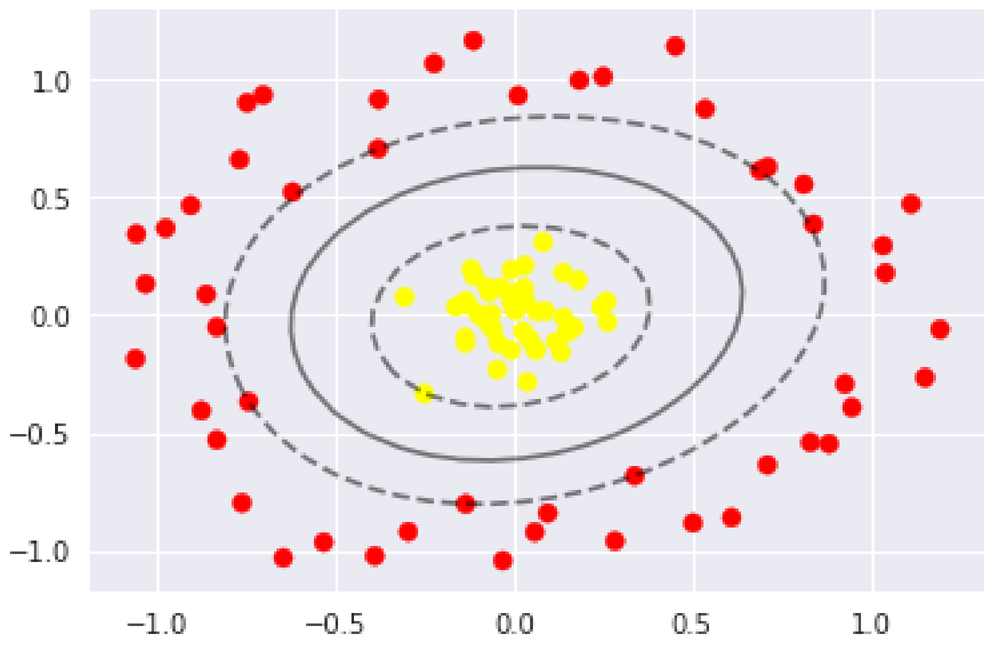
```
In [7]: plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
        plot_svc_decision_function(model);
```



```
In [8]: from sklearn.datasets.samples_generator import make_circles
        X, y = make_circles(100, factor=.1, noise=.1)

        clf = SVC(kernel='linear').fit(X, y)

        plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
        plot_svc_decision_function(clf, plot_support=False);
```



```
In [9]: clf = SVC(kernel='rbf', C=1E6)
        clf.fit(X, y)
        plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
        plot_svc_decision_function(clf, plot_support=False);
```



2.2.1 On my own dataset

```
In [3]: wine_data = pd.read_csv('../Dataset/winequality-white.csv', sep=';')
        red_wine = pd.read_csv('../Dataset/winequality-red.csv', sep=';')
        wine_data.append(red_wine)
        wine_data.head(10)
```

```
Out[3]:    fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
        0            7.0              0.27         0.36            20.7      0.045
        1            6.3              0.30         0.34             1.6      0.049
        2            8.1              0.28         0.40             6.9      0.050
        3            7.2              0.23         0.32             8.5      0.058
        4            7.2              0.23         0.32             8.5      0.058
        5            8.1              0.28         0.40             6.9      0.050
        6            6.2              0.32         0.16             7.0      0.045
        7            7.0              0.27         0.36            20.7      0.045
        8            6.3              0.30         0.34             1.6      0.049
        9            8.1              0.22         0.43             1.5      0.044

           free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
        0                 45.0                 170.0   1.0010  3.00       0.45
        1                 14.0                 132.0   0.9940  3.30       0.49
        2                 30.0                  97.0   0.9951  3.26       0.44
        3                 47.0                 186.0   0.9956  3.19       0.40
        4                 47.0                 186.0   0.9956  3.19       0.40
        5                 30.0                  97.0   0.9951  3.26       0.44
        6                 30.0                 136.0   0.9949  3.18       0.47
        7                 45.0                 170.0   1.0010  3.00       0.45
        8                 14.0                 132.0   0.9940  3.30       0.49
        9                 28.0                 129.0   0.9938  3.22       0.45

           alcohol  quality
        0      8.8        6
        1      9.5        6
        2     10.1        6
        3      9.9        6
        4      9.9        6
        5     10.1        6
        6      9.6        6
        7      8.8        6
        8      9.5        6
        9     11.0        6
```

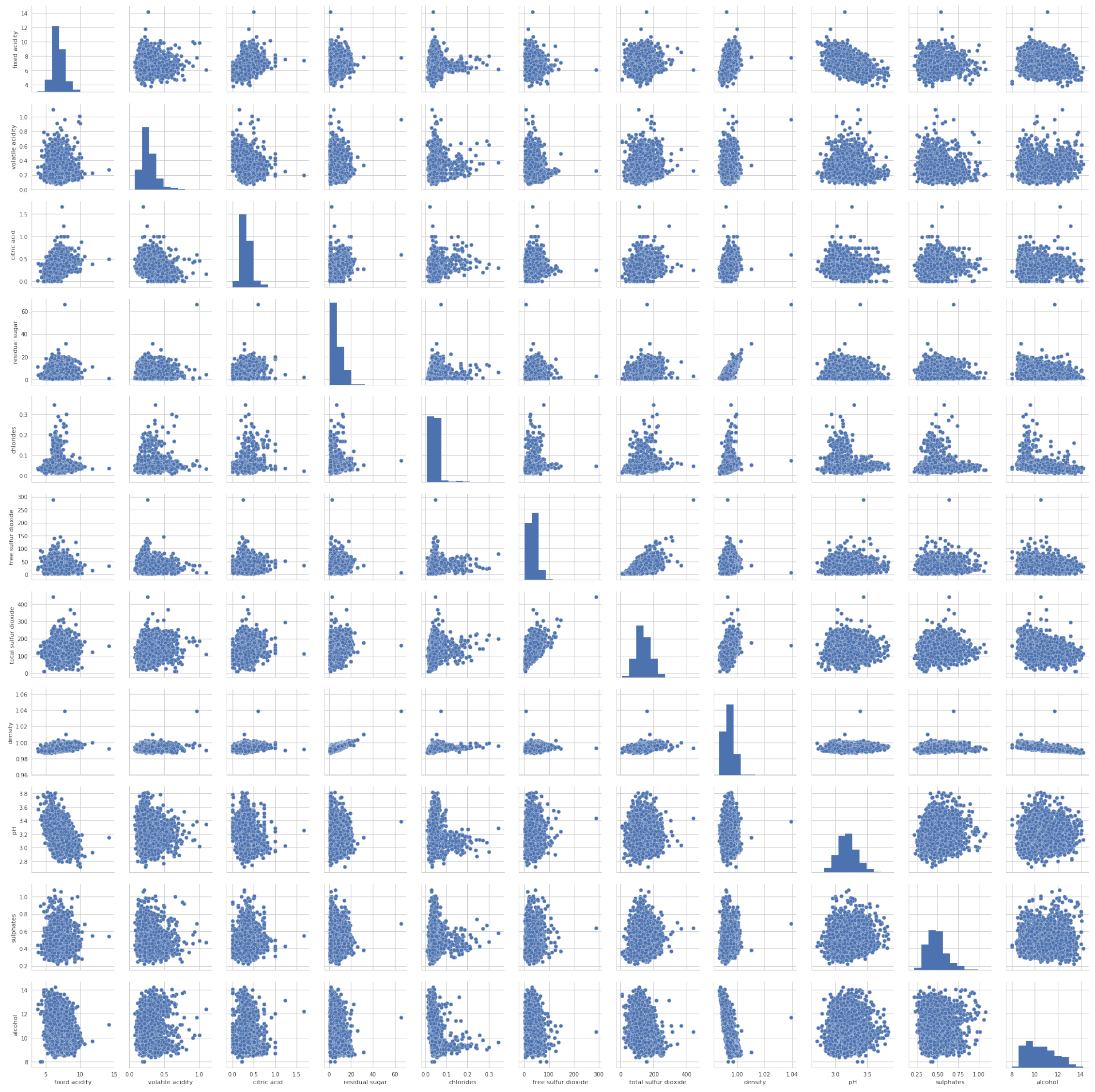
```
In [4]: wine_data.dtypes
```

```
Out[4]: fixed acidity           float64
        volatile acidity        float64
        citric acid             float64
        residual sugar          float64
```

```
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
dtype: object
```

```
In [5]: sns.pairplot(wine_data.iloc[:,:11].dropna(), size=2.5)
```

```
Out[5]: <seaborn.axisgrid.PairGrid at 0x7fd48ac0d550>
```



```
In [6]: X_train, X_test, Y_train, Y_test = train_test_split(wine_data.iloc[:,:11],
                                                             wine_data.iloc[:, 11],
                                                             test_size=0.2,
                                                             random_state=42)
        print('Size of training set: ', len(X_train.axes[0]))
        print('Size of test set: ', len(X_test.axes[0]))
```

```
Size of training set:  3918
Size of test set:  980
```

```
In [7]: parameters = [{'kernel': ['rbf'],
                      'gamma': [1e-4, 1e-3, 0.01, 0.1, 0.2, 0.5],
                       'C': [1, 10, 100, 1000]},
                     {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]
```

```
In [8]: clf = GridSearchCV(SVC(decision_function_shape='ovr'), parameters, cv=5)
        clf.fit(X_train, Y_train)
```

```
Out[8]: GridSearchCV(cv=5, error_score='raise',
               estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
          decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
          max_iter=-1, probability=False, random_state=None, shrinking=True,
          tol=0.001, verbose=False),
               fit_params=None, iid=True, n_jobs=1,
               param_grid=[{'kernel': ['rbf'], 'gamma': [0.0001, 0.001, 0.01, 0.1, 0.2, 0.5],
               pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
               scoring=None, verbose=0)
```

```
In [9]: print("Best parameters set found on development set:")
        print(clf.best_params_)
        print("Grid scores on training set:")
        means = clf.cv_results_['mean_test_score']
        stds = clf.cv_results_['std_test_score']
        for mean, std, params in zip(means, stds, clf.cv_results_['params']):
            print("%0.3f (+/-%0.03f) for %r"
                  % (mean, std * 2, params))
```

```
Best parameters set found on development set:
{'C': 1, 'gamma': 0.5, 'kernel': 'rbf'}
Grid scores on training set:
0.450 (+/-0.016) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.472 (+/-0.018) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.490 (+/-0.025) for {'C': 1, 'gamma': 0.01, 'kernel': 'rbf'}
0.555 (+/-0.008) for {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
0.576 (+/-0.027) for {'C': 1, 'gamma': 0.2, 'kernel': 'rbf'}
0.593 (+/-0.019) for {'C': 1, 'gamma': 0.5, 'kernel': 'rbf'}
0.479 (+/-0.035) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.503 (+/-0.022) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
```

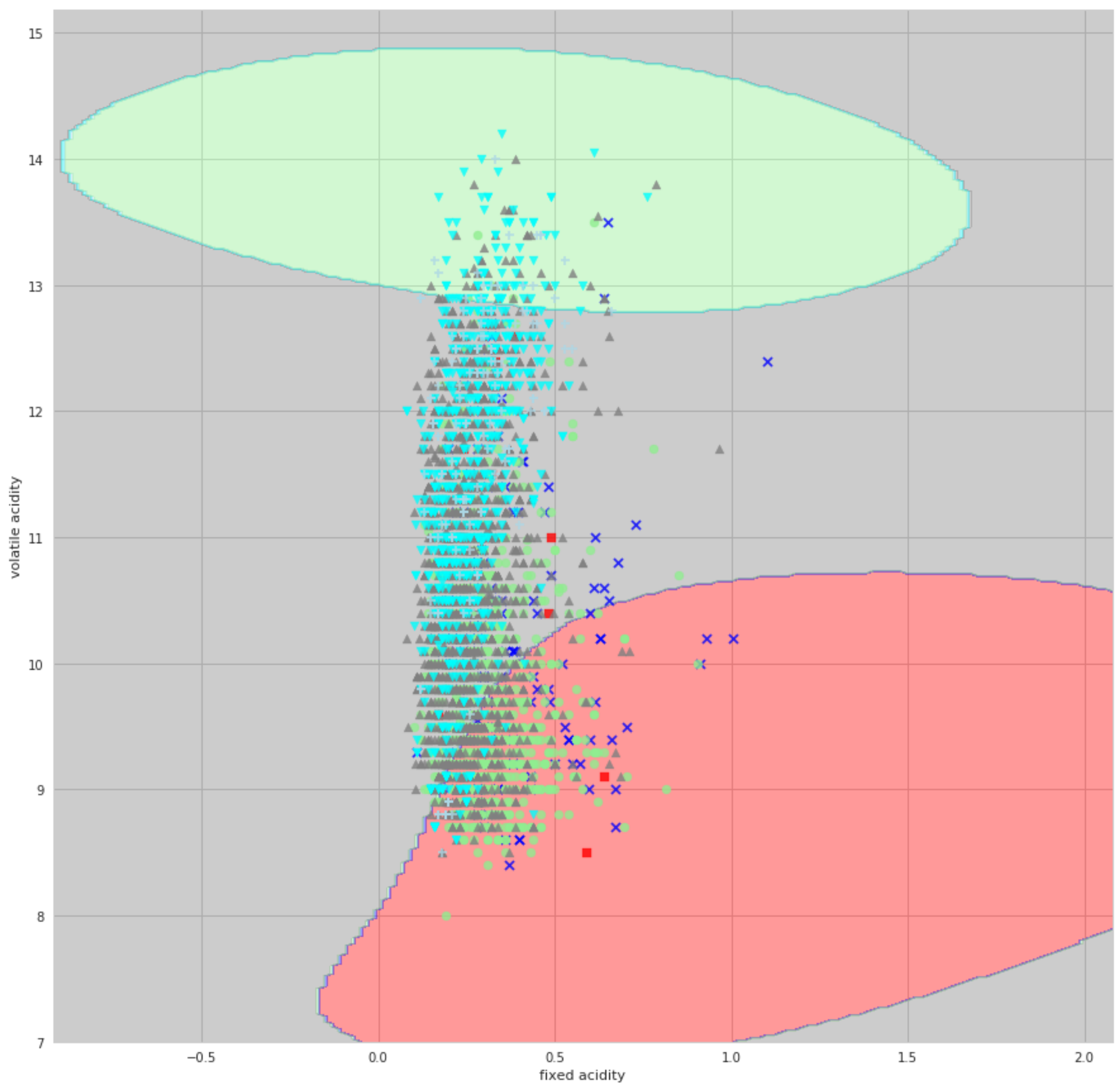
```
0.537 (+/-0.016) for {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
0.556 (+/-0.018) for {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
0.573 (+/-0.044) for {'C': 10, 'gamma': 0.2, 'kernel': 'rbf'}
0.589 (+/-0.029) for {'C': 10, 'gamma': 0.5, 'kernel': 'rbf'}
0.517 (+/-0.019) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.526 (+/-0.027) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.554 (+/-0.006) for {'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
0.555 (+/-0.021) for {'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
0.573 (+/-0.044) for {'C': 100, 'gamma': 0.2, 'kernel': 'rbf'}
0.589 (+/-0.029) for {'C': 100, 'gamma': 0.5, 'kernel': 'rbf'}
0.528 (+/-0.018) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.551 (+/-0.029) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.566 (+/-0.036) for {'C': 1000, 'gamma': 0.01, 'kernel': 'rbf'}
0.556 (+/-0.021) for {'C': 1000, 'gamma': 0.1, 'kernel': 'rbf'}
0.573 (+/-0.044) for {'C': 1000, 'gamma': 0.2, 'kernel': 'rbf'}
0.589 (+/-0.029) for {'C': 1000, 'gamma': 0.5, 'kernel': 'rbf'}
0.526 (+/-0.015) for {'C': 1, 'kernel': 'linear'}
0.526 (+/-0.018) for {'C': 10, 'kernel': 'linear'}
0.534 (+/-0.024) for {'C': 100, 'kernel': 'linear'}
0.522 (+/-0.033) for {'C': 1000, 'kernel': 'linear'}
```

```
In [10]: print("Detailed classification report:")
         print("The model is trained on the full development set.")
         print("The scores are computed on the full evaluation set.")
         #data_train, data_test, label_train, label_test

         y_true, y_pred = Y_test, clf.predict(X_test)
         print(classification_report(y_true, y_pred))
         print()
```

```
Detailed classification report:
The model is trained on the full development set.
The scores are computed on the full evaluation set.
             precision    recall  f1-score   support

          3       0.00      0.00      0.00         5
          4       1.00      0.08      0.15        25
          5       0.83      0.39      0.53       291
          6       0.55      0.95      0.70       432
          7       0.91      0.39      0.55       192
          8       1.00      0.37      0.54        35

avg / total       0.73      0.63      0.60       980



/home/prateek/anaconda3/envs/dltf/lib/python3.6/site-packages/sklearn/metrics/classification.py
```

```
  'precision', 'predicted', average, warn_for)
```

```
In [21]: def plot_decision_surface(X, y, classifier, test_idx=None, resolution=0.02):

             markers = ('s', 'x', 'o', '^', 'v', '+', '.')
             colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan', 'lightblue', 'lightgreen')
             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), np.arange(x2_min, x

             Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
             Z = Z.reshape(xx1.shape)

             plt.figure(figsize=(15,15))
             plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
             plt.xlim(xx1.min(), xx1.max())
             plt.ylim(xx2.min(), xx2.max())
             plt.xlabel('fixed acidity')
             plt.ylabel('volatile acidity')

             X_test, y_test = X[test_idx, :], y[test_idx]

             for idx, cl in enumerate(np.unique(y)):
                 plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                 alpha=0.8, c=cmap(idx),
                 marker=markers[idx], label=cl)
                 if test_idx:
                     X_test, y_test = X[test_idx, :], y[test_idx]
                     plt.scatter(X_test[:, 0], X_test[:, 1], c='',
                     alpha=1.0, linewidth=1, marker='o',
                     s=55, label='test set')

In [22]: svc = SVC(C=1.0, kernel='rbf')
         svc.fit(X_train.iloc[:, [1, 10]], Y_train)

Out[22]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

In [23]: a = np.array(X_train.iloc[:, [1, 10]])
         plot_decision_surface(X=a, y = np.array(Y_train.values), classifier=svc)
```


15
14
13
12
11
10
9
8
7
−0.5
0.0
0.5
1.0
1.5
2.0
fixed acidity
volatile acidity