Support Vector Machines

April 24, 2019

This project illustrates the use of support vector machines algorithm to classify the data.
For the project, I have used Iris flower dataset which contains measurements for 150 iris flowers from three different species.
The three classes in the Iris dataset:

```
Iris-setosa (n=50)
Iris-versicolor (n=50)
Iris-virginica (n=50)
```

The four features of the Iris dataset:

```
sepal length in cm
sepal width in cm
petal length in cm
petal width in cm
```

Importing standard python libraries required for the problem and creating the dataframe for the dataset

```
In [1]: import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt
```

```
In [3]: df = sns.load_dataset('iris')
```

```
In [4]: df.head()
```

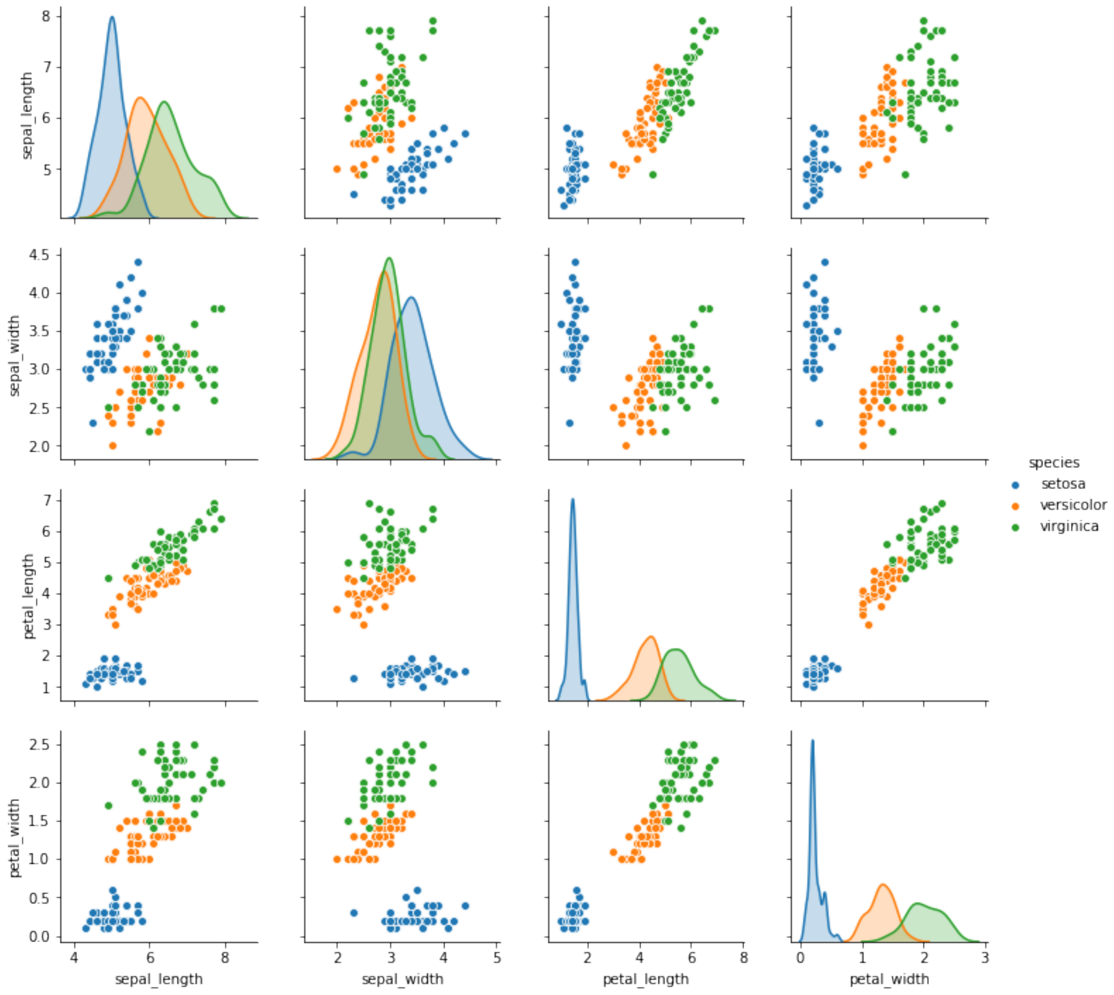
```
Out[4]:    sepal_length  sepal_width  petal_length  petal_width species
        0           5.1          3.5           1.4          0.2  setosa
        1           4.9          3.0           1.4          0.2  setosa
        2           4.7          3.2           1.3          0.2  setosa
        3           4.6          3.1           1.5          0.2  setosa
        4           5.0          3.6           1.4          0.2  setosa
```

```
In [5]: df['species'].nunique()
```

```
Out[5]: 3
```

Creating pairwise plot between all the independent variables looking for insights

```
In [9]: sns.pairplot(df,hue='species')

C:\ProgramData\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a no
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval


Out[9]: <seaborn.axisgrid.PairGrid at 0x18da7b98438>
```



From the above plot, we can observe that speal_width might not be really significant in classifying the flower, which we would be verifying in the later stage after coming up with an effective SVM model.

Splitting the data into training and testing data

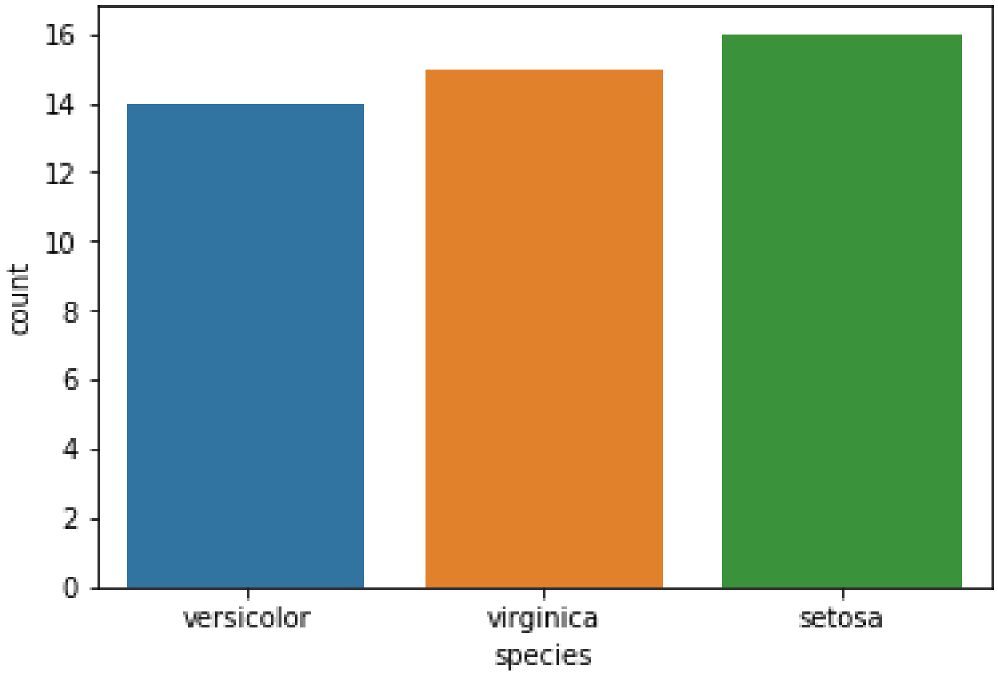
```
In [13]: from sklearn.model_selection import train_test_split

In [14]: df_y = df.pop('species')

In [15]: df_x = df
In [93]: train_x,test_x,train_y,test_y = train_test_split(df_x,df_y,test_size=0.3,random_state=
```

Creating a Support vector classfier that uses default parameters and using it to predict the target for test data

```
In [94]: from sklearn.svm import SVC

In [95]: svm = SVC()

In [96]: svm.fit(train_x,train_y)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\base.py:196: FutureWarning: The default
  "avoid this warning.", FutureWarning)


Out[96]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
           kernel='rbf', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

In [97]: pred_y = svm.predict(test_x)

In [98]: from sklearn.metrics import classification_report,confusion_matrix

In [99]: print('Default Values Model:\n')
         print('Confusion Matrix:\n')
         print(confusion_matrix(test_y,pred_y))
         print('\n')
         print('classification report: \n')
         print(classification_report(test_y,pred_y))

Default Values Model:

Confusion Matrix:

[[16  0  0]
 [ 0 13  1]
 [ 0  0 15]]


classification report:

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        16
  versicolor       1.00      0.93      0.96        14
   virginica       0.94      1.00      0.97        15

   micro avg       0.98      0.98      0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

```
In [100]: sns.countplot(test_y)
```

```
Out[100]: <matplotlib.axes._subplots.AxesSubplot at 0x18da6b80ba8>
```



The model performed pretty good with default parameters where we had an accuracy of 0.98
Performing Grid search to find best parameters for the SVM model, to check if there would be any important.

```
In [101]: from sklearn.model_selection import GridSearchCV
```

```
In [102]: param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001]}
```

```
In [103]: grid_svm = GridSearchCV(SVC(),param_grid,verbose = 3)
```

```
In [104]: grid_svm.fit(train_x,train_y)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:2053: FutureWarni
  warnings.warn(CV_WARNING, FutureWarning)
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:    0.0s
```

```
Fitting 3 folds for each of 16 candidates, totalling 48 fits
[CV] C=0.1, gamma=1 ...
[CV] ... C=0.1, gamma=1, score=0.9166666666666666, total=   0.0s
[CV] C=0.1, gamma=1 ...
[CV] ... C=0.1, gamma=1, score=0.9142857142857143, total=   0.0s
[CV] C=0.1, gamma=1 ...
[CV] ... C=0.1, gamma=1, score=0.9411764705882353, total=   0.0s
[CV] C=0.1, gamma=0.1 ...
[CV] ... C=0.1, gamma=0.1, score=0.8333333333333334, total=   0.0s
[CV] C=0.1, gamma=0.1 ...
[CV] ... C=0.1, gamma=0.1, score=0.8571428571428571, total=   0.0s
[CV] C=0.1, gamma=0.1 ...
[CV] ... C=0.1, gamma=0.1, score=0.9411764705882353, total=   0.0s
[CV] C=0.1, gamma=0.01 ...
[CV] ... C=0.1, gamma=0.01, score=0.3333333333333333, total=   0.0s
[CV] C=0.1, gamma=0.01 ...
[CV] ... C=0.1, gamma=0.01, score=0.34285714285714286, total=   0.0s
[CV] C=0.1, gamma=0.01 ...
[CV] ... C=0.1, gamma=0.01, score=0.6470588235294118, total=   0.0s
[CV] C=0.1, gamma=0.001 ...
[CV] ... C=0.1, gamma=0.001, score=0.3333333333333333, total=   0.0s
[CV] C=0.1, gamma=0.001 ...
[CV] ... C=0.1, gamma=0.001, score=0.34285714285714286, total=   0.0s
[CV] C=0.1, gamma=0.001 ...
[CV] ... C=0.1, gamma=0.001, score=0.6470588235294118, total=   0.0s
[CV] C=1, gamma=1 ...
[CV] ... C=1, gamma=1, score=0.9444444444444444, total=   0.0s
[CV] C=1, gamma=1 ...
[CV] ... C=1, gamma=1, score=0.9428571428571428, total=   0.0s
[CV] C=1, gamma=1 ...
[CV] ... C=1, gamma=1, score=0.9705882352941176, total=   0.0s
[CV] C=1, gamma=0.1 ...
[CV] ... C=1, gamma=0.1, score=0.9444444444444444, total=   0.0s
[CV] C=1, gamma=0.1 ...
[CV] ... C=1, gamma=0.1, score=0.9714285714285714, total=   0.0s
[CV] C=1, gamma=0.1 ...
[CV] ... C=1, gamma=0.1, score=0.9411764705882353, total=   0.0s
[CV] C=1, gamma=0.01 ...
[CV] ... C=1, gamma=0.01, score=0.8333333333333334, total=   0.0s
[CV] C=1, gamma=0.01 ...
[CV] ... C=1, gamma=0.01, score=0.8857142857142857, total=   0.0s
[CV] C=1, gamma=0.01 ...
[CV] ... C=1, gamma=0.01, score=0.9705882352941176, total=   0.0s
[CV] C=1, gamma=0.001 ...
[CV] ... C=1, gamma=0.001, score=0.3333333333333333, total=   0.0s
[CV] C=1, gamma=0.001 ...
[CV] ... C=1, gamma=0.001, score=0.34285714285714286, total=   0.0s
[CV] C=1, gamma=0.001 ...
```

```
[CV] ... C=1, gamma=0.001, score=0.6470588235294118, total=   0.0s
[CV] C=10, gamma=1 ...
[CV] ... C=10, gamma=1, score=0.9444444444444444, total=   0.0s
[CV] C=10, gamma=1 ...
[CV] ... C=10, gamma=1, score=0.9428571428571428, total=   0.0s
[CV] C=10, gamma=1 ...
[CV] ... C=10, gamma=1, score=0.9411764705882353, total=   0.0s
[CV] C=10, gamma=0.1 ...
[CV] ... C=10, gamma=0.1, score=0.9722222222222222, total=   0.0s
[CV] C=10, gamma=0.1 ...
[CV] ... C=10, gamma=0.1, score=0.9428571428571428, total=   0.0s
[CV] C=10, gamma=0.1 ...
[CV] ... C=10, gamma=0.1, score=0.9705882352941176, total=   0.0s
[CV] C=10, gamma=0.01 ...
[CV] ... C=10, gamma=0.01, score=0.9444444444444444, total=   0.0s
[CV] C=10, gamma=0.01 ...
[CV] ... C=10, gamma=0.01, score=0.9714285714285714, total=   0.0s
[CV] C=10, gamma=0.01 ...
[CV] ... C=10, gamma=0.01, score=0.9411764705882353, total=   0.0s
[CV] C=10, gamma=0.001 ...
[CV] ... C=10, gamma=0.001, score=0.8333333333333334, total=   0.0s
[CV] C=10, gamma=0.001 ...
[CV] ... C=10, gamma=0.001, score=0.8857142857142857, total=   0.0s
[CV] C=10, gamma=0.001 ...
[CV] ... C=10, gamma=0.001, score=0.9705882352941176, total=   0.0s
[CV] C=100, gamma=1 ...
[CV] ... C=100, gamma=1, score=0.8333333333333334, total=   0.0s
[CV] C=100, gamma=1 ...
[CV] ... C=100, gamma=1, score=0.9428571428571428, total=   0.0s
[CV] C=100, gamma=1 ...
[CV] ... C=100, gamma=1, score=0.9411764705882353, total=   0.0s
[CV] C=100, gamma=0.1 ...
[CV] ... C=100, gamma=0.1, score=0.9166666666666666, total=   0.0s
[CV] C=100, gamma=0.1 ...
[CV] ... C=100, gamma=0.1, score=0.9714285714285714, total=   0.0s
[CV] C=100, gamma=0.1 ...
[CV] ... C=100, gamma=0.1, score=0.9705882352941176, total=   0.0s
[CV] C=100, gamma=0.01 ...
[CV] ... C=100, gamma=0.01, score=0.9722222222222222, total=   0.0s
[CV] C=100, gamma=0.01 ...
[CV] ... C=100, gamma=0.01, score=0.9714285714285714, total=   0.0s
[CV] C=100, gamma=0.01 ...
[CV] ... C=100, gamma=0.01, score=0.9705882352941176, total=   0.0s
[CV] C=100, gamma=0.001 ...
[CV] ... C=100, gamma=0.001, score=0.9444444444444444, total=   0.0s
[CV] C=100, gamma=0.001 ...
[CV] ... C=100, gamma=0.001, score=0.9714285714285714, total=   0.0s
[CV] C=100, gamma=0.001 ...
```

```
[CV] ... C=100, gamma=0.001, score=0.9411764705882353, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done  48 out of  48 | elapsed:    0.1s finished
```

```
Out[104]: GridSearchCV(cv='warn', error_score='raise-deprecating',
                 estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
            decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
            kernel='rbf', max_iter=-1, probability=False, random_state=None,
            shrinking=True, tol=0.001, verbose=False),
                 fit_params=None, iid='warn', n_jobs=None,
                 param_grid={'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001]},
                 pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                 scoring=None, verbose=3)
```

```
In [110]: grid_svm.best_params_
```

```
Out[110]: {'C': 100, 'gamma': 0.01}
```

From the grid search its identified that C=100 and gamma = 0.01 would give the best model through which we are going to predict target variable for test data again.

```
In [111]: grid_svm.best_estimator_
```

```
Out[111]: SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
            decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf',
            max_iter=-1, probability=False, random_state=None, shrinking=True,
            tol=0.001, verbose=False)
```

```
In [112]: pred_new = grid_svm.predict(test_x)
```

```
In [113]: print('Default Values Model:\n')
          print('Confusion Matrix:\n')
          print(confusion_matrix(test_y,pred_new))
          print('\n')
          print('classification report: \n')
          print(classification_report(test_y,pred_new))
```

```
Default Values Model:

Confusion Matrix:

[[16  0  0]
 [ 0 14  0]
 [ 0  0 15]]


classification report:
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        16
  versicolor       1.00      1.00      1.00        14
   virginica       1.00      1.00      1.00        15

   micro avg       1.00      1.00      1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

We have achieved an SVM model that is 100% accurate for the given data by performing grid search for best parameters. It would be really useful in datasets with large data than smaller datasets to perform grid search

Now that we have a model, we will try to predict the target again by dropping the sepal_width variable to check if it would really have any effect in predicting the flower species

```
In [114]: df_x = df_x.drop(['sepal_width'],axis=1)

In [115]: train_x,test_x,train_y,test_y = train_test_split(df_x,df_y,test_size=0.3,random_stat

In [116]: svm = SVC(C=100,gamma=0.01)

In [117]: svm.fit(train_x,train_y)

Out[117]: SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
            decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf',
            max_iter=-1, probability=False, random_state=None, shrinking=True,
            tol=0.001, verbose=False)

In [118]: pred_3 = svm.predict(test_x)

In [119]: print('Default Values Model:\n')
          print('Confusion Matrix:\n')
          print(confusion_matrix(test_y,pred_3))
          print('\n')
          print('classification report: \n')
          print(classification_report(test_y,pred_3))

Default Values Model:

Confusion Matrix:

[[16  0  0]
 [ 0 14  0]
 [ 0  0 15]]
```

```
classification report:

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        16
  versicolor       1.00      1.00      1.00        14
   virginica       1.00      1.00      1.00        15

   micro avg       1.00      1.00      1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

After running the model, we can see that we still have an accuracy of 100%, which confirms us that sepal_width can be ommited from the independent variables.

```
In [ ]:
```