Excercise_7_solutions_part_2

May 30, 2019

1 Support Vector Machines Project

1.0.1 The Data

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

In [2]:
```
from IPython.display import Image
# The Iris Setosa
# url = 'http://upload.wikimedia.org/wikipedia/commons/5/56/Kosaciec_szczecinkowaty_Ir
# Image(url,width=300, height=300)
```

In [3]:
```
# The Iris Versicolor
# from IPython.display import Image
# url = 'http://upload.wikimedia.org/wikipedia/commons/4/41/Iris_versicolor_3.jpg'
# Image(url,width=300, height=300)
```

In [4]:
```
# from IPython.display import Image
# url = 'http://upload.wikimedia.org/wikipedia/commons/9/9f/Iris_virginica.jpg'
# Image(url,width=300, height=300)
```

1.0.2 Get the data

In [5]:
```
iris = sns.load_dataset('iris')
iris.head()
```

Out[5]:

	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

Let's visualize the data.

1.0.3 Exploratory Data Analysis

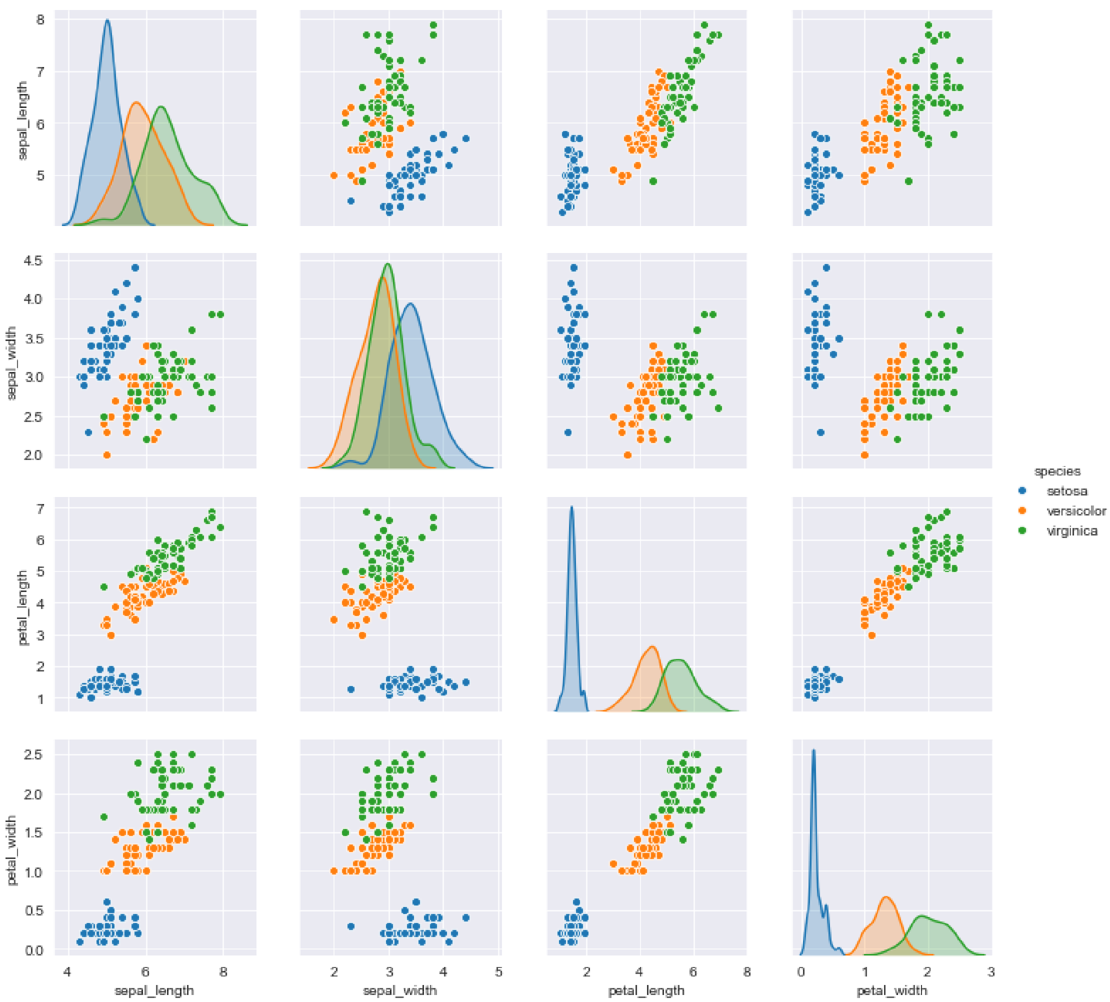
Time to put your data viz skills to the test! Try to recreate the following plots, make sure to import the libraries you'll need!

Create a pairplot of the data set. Which flower species seems to be the most separable?

```
In [6]: sns.set_style('darkgrid')
        sns.pairplot(iris, hue = 'species')
```

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

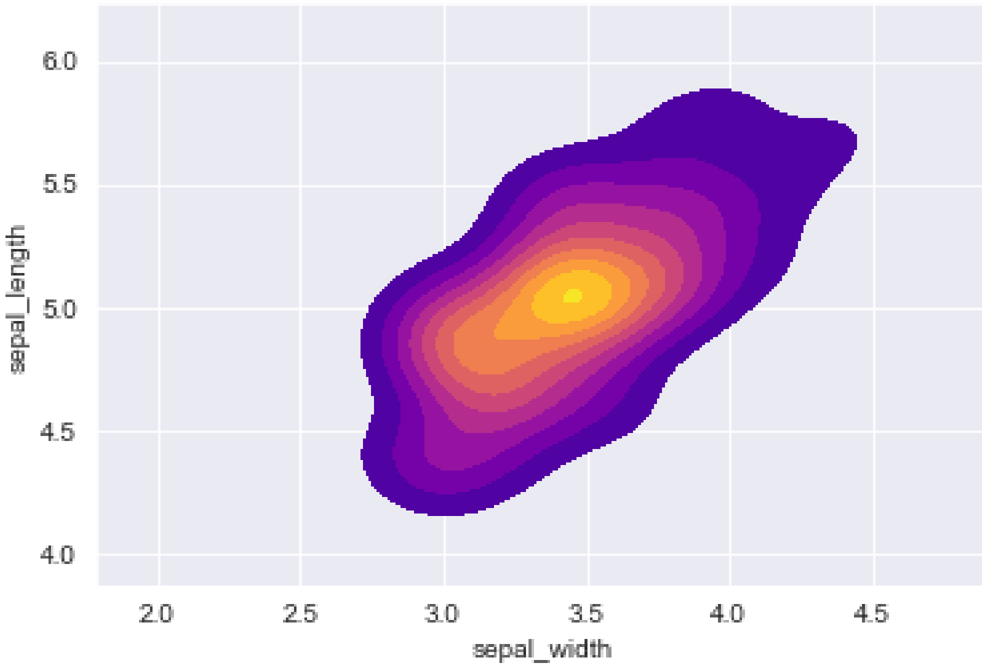
```
Out[6]: <seaborn.axisgrid.PairGrid at 0x23158c54da0>
```



Create a kde plot of sepal_length versus sepal width for setosa species of flower.

```
In [7]: sns.kdeplot(iris[iris['species'] == 'setosa']['sepal_width'],
                    iris[iris['species'] == 'setosa']['sepal_length'],
                    shade=True, cmap='plasma', shade_lowest=False)
```

```
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2315aa61470>
```



1.0.4 Train Test Split

Split your data into a training set and a testing set.

```
In [8]: from sklearn.model_selection import train_test_split
```

```
In [9]: iris.head()
```

Out[9]:

	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 [10]: X = iris.drop('species', axis=1)
         y = iris['species']
         X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)
         type(X)
```

```
Out[10]: pandas.core.frame.DataFrame
```

1.0.5 Train a Model

Now its time to train a Support Vector Machine Classifier. #### Call the SVC() model from sklearn and fit the model to the training data.

```
In [11]: from sklearn.svm import SVC
```

```
In [12]: svm_classifier = SVC()
         svm_classifier.fit(X_train,y_train)
```

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

```
Out[12]: 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)
```

1.0.6 Model Evaluation

Now get predictions from the model and create a confusion matrix and a classification report.

```
In [13]: from sklearn.metrics import classification_report, confusion_matrix
```

```
In [14]: y_pred = svm_classifier.predict(X_test)
```

```
In [15]: print('Classification report:\n', classification_report(y_test, y_pred))
```

```
Classification report:
               precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       1.00      1.00      1.00        11
   virginica       1.00      1.00      1.00        12

   micro avg       1.00      1.00      1.00        38
   macro avg       1.00      1.00      1.00        38
weighted avg       1.00      1.00      1.00        38
```

```
In [16]: print('Confusion matrix:\n', confusion_matrix(y_test, y_pred))
```

```
Confusion matrix:
 [[15  0  0]
 [ 0 11  0]
 [ 0  0 12]]
```

You should have noticed that your model was pretty good! Let's see if we can tune the parameters to try to get even better (unlikely, and you probably would be satisfied with these results in real like because the data set is quite small, but I just want you to practice using GridSearch.

1.0.7 Gridsearch Practice

Import GridsearchCV from SciKit Learn.

```
In [17]: from sklearn.model_selection import GridSearchCV
```

Create a dictionary called param_grid and fill out some parameters for C and gamma.

```
In [18]: param_grid = {'C' : [0.1, 1, 10, 100], 'gamma' : [1, 0.1, 0.01, 0.001]}
```

Create a GridSearchCV object and fit it to the training data.

```
In [19]: GS = GridSearchCV(SVC(),param_grid=param_grid, verbose=3)
         GS.fit(X_train, y_train)
```

```
C:\Users\Kamil\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=0.9210526315789473, total=   0.0s
[CV] C=0.1, gamma=1 ...
[CV] ... C=0.1, gamma=1, score=0.8947368421052632, total=   0.0s
[CV] C=0.1, gamma=1 ...
[CV] ... C=0.1, gamma=1, score=0.9444444444444444, total=   0.0s
[CV] C=0.1, gamma=0.1 ...
[CV] ... C=0.1, gamma=0.1, score=0.8421052631578947, total=   0.0s
[CV] C=0.1, gamma=0.1 ...
[CV] ... C=0.1, gamma=0.1, score=0.9210526315789473, total=   0.0s
[CV] C=0.1, gamma=0.1 ...
[CV] ... C=0.1, gamma=0.1, score=0.9166666666666666, total=   0.0s
[CV] C=0.1, gamma=0.01 ...
[CV] ... C=0.1, gamma=0.01, score=0.34210526315789475, total=   0.0s
[CV] C=0.1, gamma=0.01 ...
[CV] ... C=0.1, gamma=0.01, score=0.34210526315789475, total=   0.0s
[CV] C=0.1, gamma=0.01 ...
[CV] ... C=0.1, gamma=0.01, score=0.6111111111111112, total=   0.0s
[CV] C=0.1, gamma=0.001 ...
[CV] ... C=0.1, gamma=0.001, score=0.34210526315789475, total=   0.0s
[CV] C=0.1, gamma=0.001 ...
[CV] ... C=0.1, gamma=0.001, score=0.34210526315789475, total=   0.0s
[CV] C=0.1, gamma=0.001 ...
[CV] ... C=0.1, gamma=0.001, score=0.6111111111111112, total=   0.0s
[CV] C=1, gamma=1 ...
[CV] ... C=1, gamma=1, score=0.9736842105263158, total=   0.0s
```

```
[CV] C=1, gamma=1 ...
[CV] ... C=1, gamma=1, score=0.8947368421052632, total=   0.0s
[CV] C=1, gamma=1 ...
[CV] ... C=1, gamma=1, score=0.9722222222222222, total=   0.0s
[CV] C=1, gamma=0.1 ...
[CV] ... C=1, gamma=0.1, score=0.9736842105263158, total=   0.0s
[CV] C=1, gamma=0.1 ...
[CV] ... C=1, gamma=0.1, score=0.8947368421052632, total=   0.0s
[CV] C=1, gamma=0.1 ...
[CV] ... C=1, gamma=0.1, score=0.9722222222222222, total=   0.0s
[CV] C=1, gamma=0.01 ...
[CV] ... C=1, gamma=0.01, score=0.9736842105263158, total=   0.0s
[CV] C=1, gamma=0.01 ...
[CV] ... C=1, gamma=0.01, score=0.9210526315789473, total=   0.0s
[CV] C=1, gamma=0.01 ...
[CV] ... C=1, gamma=0.01, score=0.9166666666666666, total=   0.0s
[CV] C=1, gamma=0.001 ...
[CV] ... C=1, gamma=0.001, score=0.34210526315789475, total=   0.0s
[CV] C=1, gamma=0.001 ...
[CV] ... C=1, gamma=0.001, score=0.34210526315789475, total=   0.0s
[CV] C=1, gamma=0.001 ...
[CV] ... C=1, gamma=0.001, score=0.6111111111111112, total=   0.0s
[CV] C=10, gamma=1 ...
[CV] ... C=10, gamma=1, score=0.9473684210526315, total=   0.0s
[CV] C=10, gamma=1 ...
[CV] ... C=10, gamma=1, score=0.8947368421052632, total=   0.0s
[CV] C=10, gamma=1 ...
[CV] ... C=10, gamma=1, score=0.9722222222222222, total=   0.0s
[CV] C=10, gamma=0.1 ...
[CV] ... C=10, gamma=0.1, score=0.9473684210526315, total=   0.0s
[CV] C=10, gamma=0.1 ...
[CV] ... C=10, gamma=0.1, score=0.8947368421052632, 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=0.01 ...
[CV] ... C=10, gamma=0.01, score=1.0, total=   0.0s
[CV] C=10, gamma=0.01 ...
[CV] ... C=10, gamma=0.01, score=0.8947368421052632, total=   0.0s
[CV] C=10, gamma=0.01 ...
[CV] ... C=10, gamma=0.01, score=0.9722222222222222, total=   0.0s
[CV] C=10, gamma=0.001 ...
[CV] ... C=10, gamma=0.001, score=0.9736842105263158, total=   0.0s
[CV] C=10, gamma=0.001 ...
[CV] ... C=10, gamma=0.001, score=0.9210526315789473, total=   0.0s
[CV] C=10, gamma=0.001 ...
[CV] ... C=10, gamma=0.001, score=0.9166666666666666, total=   0.0s
[CV] C=100, gamma=1 ...
[CV] ... C=100, gamma=1, score=0.9473684210526315, total=   0.0s
```

```
[CV] C=100, gamma=1 ...
[CV] ... C=100, gamma=1, score=0.8947368421052632, total=   0.0s
[CV] C=100, gamma=1 ...
[CV] ... C=100, gamma=1, score=0.9444444444444444, total=   0.0s
[CV] C=100, gamma=0.1 ...
[CV] ... C=100, gamma=0.1, score=0.9473684210526315, total=   0.0s
[CV] C=100, gamma=0.1 ...
[CV] ... C=100, gamma=0.1, score=0.8947368421052632, total=   0.0s
[CV] C=100, gamma=0.1 ...
[CV] ... C=100, gamma=0.1, score=1.0, total=   0.0s
[CV] C=100, gamma=0.01 ...
[CV] ... C=100, gamma=0.01, score=1.0, total=   0.0s
[CV] C=100, gamma=0.01 ...
[CV] ... C=100, gamma=0.01, score=0.8947368421052632, total=   0.0s
[CV] C=100, gamma=0.01 ...
[CV] ... C=100, gamma=0.01, score=1.0, total=   0.0s
[CV] C=100, gamma=0.001 ...
[CV] ... C=100, gamma=0.001, score=1.0, total=   0.0s
[CV] C=100, gamma=0.001 ...
[CV] ... C=100, gamma=0.001, score=0.8947368421052632, total=   0.0s
[CV] C=100, gamma=0.001 ...
[CV] ... C=100, gamma=0.001, score=0.9722222222222222, total=   0.0s


[Parallel(n_jobs=1)]: Done  48 out of  48 | elapsed:    0.1s finished
C:\Users\Kamil\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:841: DeprecationW
  DeprecationWarning)
```

```
Out[19]: 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)
```

Now take that grid model and create some predictions using the test set and create classification reports and confusion matrices for them. Were you able to improve?

```
In [20]: y_pred = GS.predict(X_test)
```

```
In [21]: print(confusion_matrix(y_test, y_pred))
```

```
[[15  0  0]
 [ 0 11  0]
 [ 0  0 12]]
```

```
In [22]: print(classification_report(y_test, y_pred))

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       1.00      1.00      1.00        11
   virginica       1.00      1.00      1.00        12

   micro avg       1.00      1.00      1.00        38
   macro avg       1.00      1.00      1.00        38
weighted avg       1.00      1.00      1.00        38
```

You should have done about the same or exactly the same, this makes sense, there is basically just one point that is too noisy to grab, which makes sense, we don't want to have an overfit model that would be able to grab that.