Support Vector Machines Project - Iris Data set

October 27, 2016

# 1 Support Vector Machines Project - Iris Data Set

In [17]:
```
# The Iris Setosa
from IPython.display import Image
url = 'http://upload.wikimedia.org/wikipedia/commons/5/56/Kosaciec_szczecinkowaty_Iris_setosa.
Image(url,width=300, height=300)
```

Out[17]:

In [18]:
```
# 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)
```

Out[18]:

```
In [19]: # The Iris Virginica
         from IPython.display import Image
         url = 'http://upload.wikimedia.org/wikipedia/commons/9/9f/Iris_virginica.jpg'
         Image(url,width=300, height=300)
```

Out[19]:

The iris dataset 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
```

## 1.1 Get the data

```
In [24]: import seaborn as sns
         iris = sns.load_dataset('iris')
```

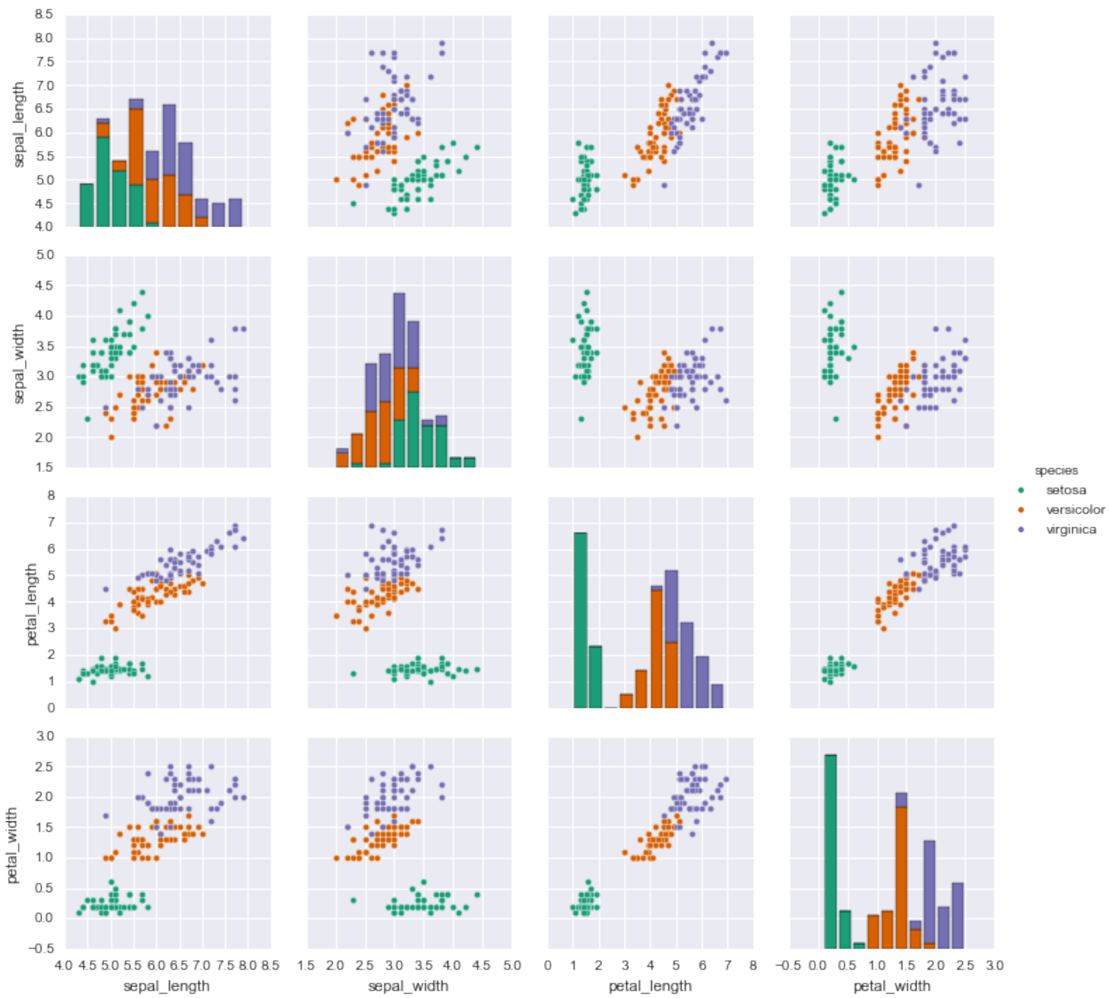
## 1.2 Exploratory Data Analysis

**Import some libraries you think you'll need.**

```
In [25]: import pandas as pd
         import matplotlib.pyplot as plt
         %matplotlib inline
```

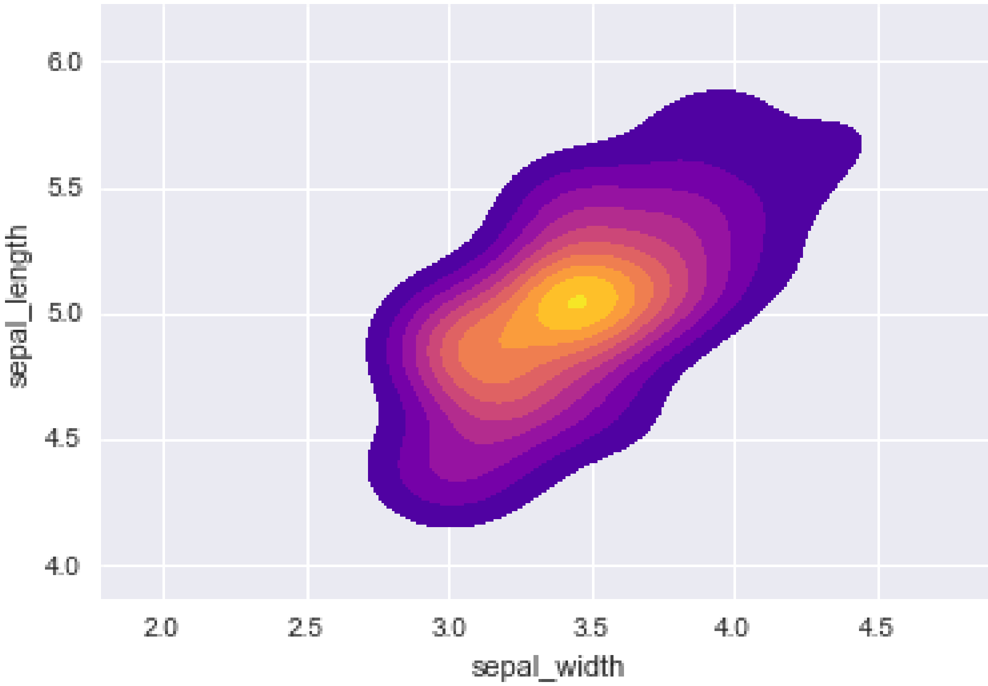
** Create a pairplot of the data set. Which flower species seems to be the most separable?**

```
In [37]: sns.pairplot(iris,hue='species',palette='Dark2')

Out[37]: <seaborn.axisgrid.PairGrid at 0x12afb9cc0>
```




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

```
In [44]: setosa = iris[iris['species']=='setosa']
         sns.kdeplot( setosa['sepal_width'], setosa['sepal_length'],
                      cmap="plasma", shade=True, shade_lowest=False)

Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x12f102080>
```




## 2 Train Test Split

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

```
In [45]: from sklearn.cross_validation import train_test_split
```

```
In [47]: X = iris.drop('species',axis=1)
         y = iris['species']
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
```

## 3 Train a Model

**Call the SVC() model from sklearn and fit the model to the training data.**

```
In [48]: from sklearn.svm import SVC
```

```
In [49]: svc_model = SVC()
```

```
In [50]: svc_model.fit(X_train,y_train)
```

```
Out[50]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)
```

### 3.1 Model Evaluation

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

```
In [51]: predictions = svc_model.predict(X_test)

In [52]: from sklearn.metrics import classification_report,confusion_matrix

In [53]: print(confusion_matrix(y_test,predictions))
```

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

```
In [54]: print(classification_report(y_test,predictions))
```

```
precision    recall  f1-score   support

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

avg / total       0.98      0.98      0.98        45
```

### 3.2 Gridsearch Practice

** Import GridsearchCV from SciKit Learn.**

```
In [55]: from sklearn.grid_search import GridSearchCV

In [57]: 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 [58]: grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=2)
         grid.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 16 candidates, totalling 48 fits
[CV] gamma=1, C=0.1 ...
[CV] ... gamma=1, C=0.1 -   0.0s
[CV] gamma=1, C=0.1 ...
[CV] ... gamma=1, C=0.1 -   0.0s
[CV] gamma=1, C=0.1 ...
[CV] ... gamma=1, C=0.1 -   0.0s
[CV] gamma=0.1, C=0.1 ...
[CV] ... gamma=0.1, C=0.1 -   0.0s
[CV] gamma=0.1, C=0.1 ...
[CV] ... gamma=0.1, C=0.1 -   0.0s
[CV] gamma=0.1, C=0.1 ...
[CV] ... gamma=0.1, C=0.1 -   0.0s
[CV] gamma=0.01, C=0.1 ...
[CV] ... gamma=0.01, C=0.1 -   0.0s
[CV] gamma=0.01, C=0.1 ...
[CV] ... gamma=0.01, C=0.1 -   0.0s
[CV] gamma=0.01, C=0.1 ...
[CV] ... gamma=0.01, C=0.1 -   0.0s
[CV] gamma=0.001, C=0.1 ...
```

```
[CV] ... gamma=0.001, C=0.1 -   0.0s
[CV] gamma=0.001, C=0.1 ...
[CV] ... gamma=0.001, C=0.1 -   0.0s
[CV] gamma=0.001, C=0.1 ...
[CV] ... gamma=0.001, C=0.1 -   0.0s
[CV] gamma=1, C=1 ...
[CV] ... gamma=1, C=1 -   0.0s
[CV] gamma=1, C=1 ...
[CV] ... gamma=1, C=1 -   0.0s
[CV] gamma=1, C=1 ...
[CV] ... gamma=1, C=1 -   0.0s
[CV] gamma=0.1, C=1 ...
[CV] ... gamma=0.1, C=1 -   0.0s
[CV] gamma=0.1, C=1 ...
[CV] ... gamma=0.1, C=1 -   0.0s
[CV] gamma=0.1, C=1 ...
[CV] ... gamma=0.1, C=1 -   0.0s
[CV] gamma=0.01, C=1 ...
[CV] ... gamma=0.01, C=1 -   0.0s
[CV] gamma=0.01, C=1 ...
[CV] ... gamma=0.01, C=1 -   0.0s
[CV] gamma=0.01, C=1 ...
[CV] ... gamma=0.01, C=1 -   0.0s
[CV] gamma=0.001, C=1 ...
[CV] ... gamma=0.001, C=1 -   0.0s
[CV] gamma=0.001, C=1 ...
[CV] ... gamma=0.001, C=1 -   0.0s
[CV] gamma=0.001, C=1 ...
[CV] ... gamma=0.001, C=1 -   0.0s
[CV] gamma=1, C=10 ...
[CV] ... gamma=1, C=10 -   0.0s
[CV] gamma=1, C=10 ...
[CV] ... gamma=1, C=10 -   0.0s
[CV] gamma=1, C=10 ...
[CV] ... gamma=1, C=10 -   0.0s
[CV] gamma=0.1, C=10 ...
[CV] ... gamma=0.1, C=10 -   0.0s
[CV] gamma=0.1, C=10 ...
[CV] ... gamma=0.1, C=10 -   0.0s
[CV] gamma=0.1, C=10 ...
[CV] ... gamma=0.1, C=10 -   0.0s
[CV] gamma=0.01, C=10 ...
[CV] ... gamma=0.01, C=10 -   0.0s
[CV] gamma=0.01, C=10 ...
[CV] ... gamma=0.01, C=10 -   0.0s
[CV] gamma=0.01, C=10 ...
[CV] ... gamma=0.01, C=10 -   0.0s
[CV] gamma=0.001, C=10 ...
[CV] ... gamma=0.001, C=10 -   0.0s
[CV] gamma=0.001, C=10 ...
[CV] ... gamma=0.001, C=10 -   0.0s
[CV] gamma=0.001, C=10 ...
[CV] ... gamma=0.001, C=10 -   0.0s
[CV] gamma=1, C=100 ...
```

```
[CV] ... gamma=1, C=100 -   0.0s
[CV] gamma=1, C=100 ...
[CV] ... gamma=1, C=100 -   0.0s
[CV] gamma=1, C=100 ...
[CV] ... gamma=1, C=100 -   0.0s
[CV] gamma=0.1, C=100 ...
[CV] ... gamma=0.1, C=100 -   0.0s
[CV] gamma=0.1, C=100 ...
[CV] ... gamma=0.1, C=100 -   0.0s
[CV] gamma=0.1, C=100 ...
[CV] ... gamma=0.1, C=100 -   0.0s
[CV] gamma=0.01, C=100 ...
[CV] ... gamma=0.01, C=100 -   0.0s
[CV] gamma=0.01, C=100 ...
[CV] ... gamma=0.01, C=100 -   0.0s
[CV] gamma=0.01, C=100 ...
[CV] ... gamma=0.01, C=100 -   0.0s
[CV] gamma=0.001, C=100 ...
[CV] ... gamma=0.001, C=100 -   0.0s
[CV] gamma=0.001, C=100 ...
[CV] ... gamma=0.001, C=100 -   0.0s
[CV] gamma=0.001, C=100 ...
[CV] ... gamma=0.001, C=100 -   0.0s

[Parallel(n_jobs=1)]: Done  40 tasks       | elapsed:    0.2s
[Parallel(n_jobs=1)]: Done  48 out of  48 | elapsed:    0.2s finished
```

```
Out[58]: GridSearchCV(cv=None, error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'gamma': [1, 0.1, 0.01, 0.001], 'C': [0.1, 1, 10, 100]},
       pre_dispatch='2*n_jobs', refit=True, scoring=None, verbose=2)
```

```
In [59]: grid_predictions = grid.predict(X_test)
```

```
In [60]: print(confusion_matrix(y_test,grid_predictions))
```

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

```
In [61]: print(classification_report(y_test,grid_predictions))
```

```
precision    recall  f1-score   support

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

avg / total       0.98      0.98      0.98        45
```