Support Vector Machines Project

June 10, 2018

1 Support Vector Machines Project

1.1 The Data

For this series of lectures, we will be using the famous Iris flower data set.

The Iris flower data set or Fisher's Iris data set is a multivariate data set introduced by Sir Ronald Fisher in the 1936 as an example of discriminant analysis.

The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor), so 150 total samples. Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters.

Here's a picture of the three different Iris types:

```
In [17]: # The Iris Setosa
         from IPython.display import Image
         url = 'http://upload.wikimedia.org/wikipedia/commons/5/56/Kosaciec_szczecinkowaty_Iri
         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
```

1.2 Get the data

Use seaborn to get the iris data by using: iris = sns.load_dataset('iris')

```
In [1]: import seaborn as sns
        iris = sns.load_dataset('iris')
```

Let's visualize the data and get you started!

1.3 Exploratory Data Analysis

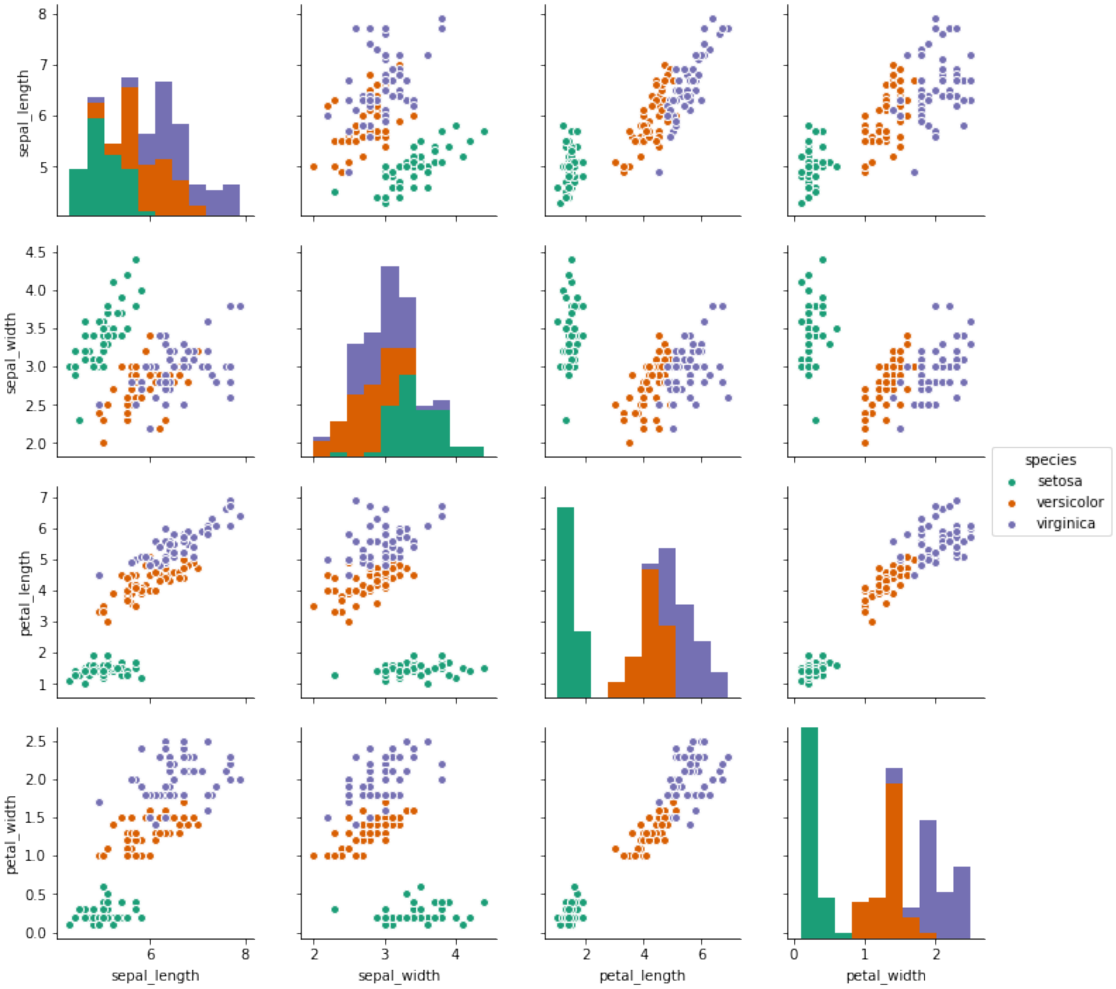
Time to put your data viz skills to the test! Try to recreate the following plots, make sure to import the libraries you'll need!

Import some libraries you think you'll need.

```
In [2]: import pandas as pd
        import numpy as np
        %matplotlib inline
```

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

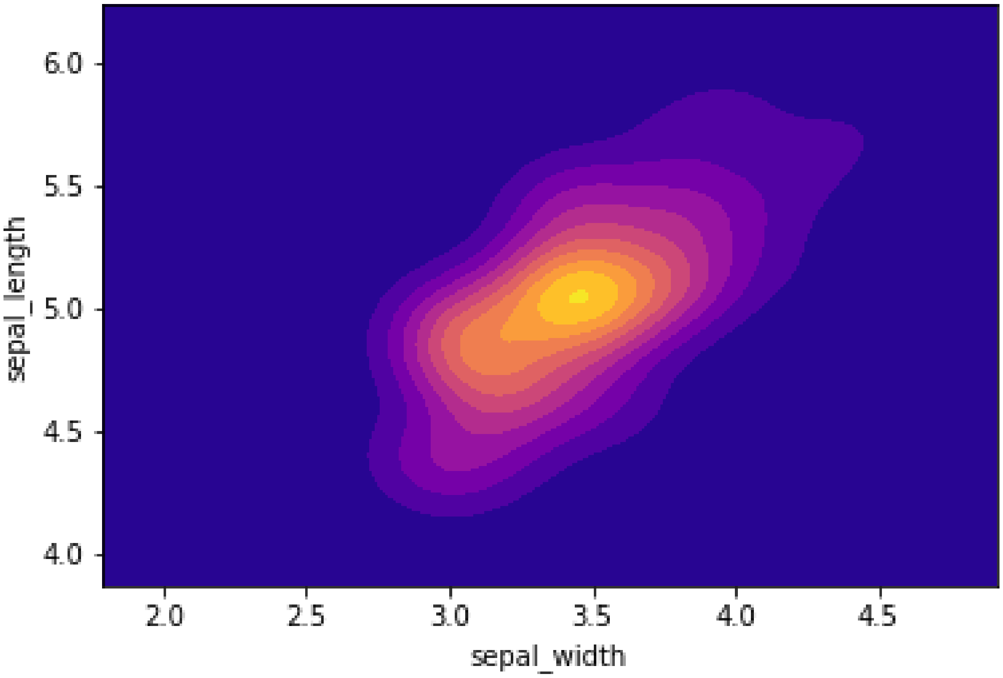
```
In [6]: sns.pairplot(iris, hue='species', palette='Dark2')

        #setosa seems to be the most seperable.

Out[6]: <seaborn.axisgrid.PairGrid at 0x1a15c8f240>
```



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

```
In [9]: setosa = iris[iris['species']=='setosa']
        sns.kdeplot(setosa['sepal_width'],setosa['sepal_length'],cmap='plasma',shade='True',sha
```

```
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x1a16c66a20>
```



2 Train Test Split

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

```
In [10]: from sklearn.cross_validation import train_test_split
```

```
/Users/Momin/anaconda3/lib/python3.6/site-packages/sklearn/cross_validation.py:41: Deprecation
  "This module will be removed in 0.20.", DeprecationWarning)
```

```
In [12]: X = iris.drop('species',axis=1)
         y = iris['species']

         X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3)
```

3 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 [13]: from sklearn.svm import SVC
```

```
In [14]: svc_model = SVC()
```

```
In [15]: svc_model.fit(X_train,y_train)
```

```
Out[15]: 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)
```

3.1 Model Evaluation

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

```
In [16]: predictions = svc_model.predict(X_test)
```

```
In [17]: from sklearn.metrics import classification_report, confusion_matrix
```

```
In [22]: print(confusion_matrix(y_test,predictions))
```

```
[[13  0  0]
 [ 0 19  1]
 [ 0  1 11]]
```

```
In [21]: print(classification_report(y_test, predictions))
```

```
             precision    recall  f1-score   support

     setosa       1.00      1.00      1.00        13
 versicolor       0.95      0.95      0.95        20
  virginica       0.92      0.92      0.92        12

avg / total       0.96      0.96      0.96        45
```

Wow! 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.

3.2 Gridsearch Practice

** Import GridsearchCV from SciKit Learn.**

```
In [23]: from sklearn.grid_search import GridSearchCV
```

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

```
In [24]: 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 [25]: grid = GridSearchCV(SVC(),param_grid,verbose=2)
         grid.fit(X_train,y_train)
```

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

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

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


[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  48 out of  48 | elapsed:    0.2s finished
```

```
Out[25]: GridSearchCV(cv=None, 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={}, iid=True, n_jobs=1,
                param_grid={'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001]},
                pre_dispatch='2*n_jobs', refit=True, scoring=None, verbose=2)
```

** 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 [26]: grid_predictions = grid.predict(X_test)
```

```
In [27]: print(confusion_matrix(y_test,grid_predictions))
```

```
[[13  0  0]
 [ 0 19  1]
 [ 0  1 11]]
```

```
In [29]: print(classification_report(y_test, grid_predictions))
```

```
             precision    recall  f1-score   support

     setosa       1.00      1.00      1.00        13
 versicolor       0.95      0.95      0.95        20
  virginica       0.92      0.92      0.92        12

avg / total       0.96      0.96      0.96        45
```

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