SciComp With Py

Classifying Real World Data with Supervised Learning: Introduction to Iris Sklearn Dataset

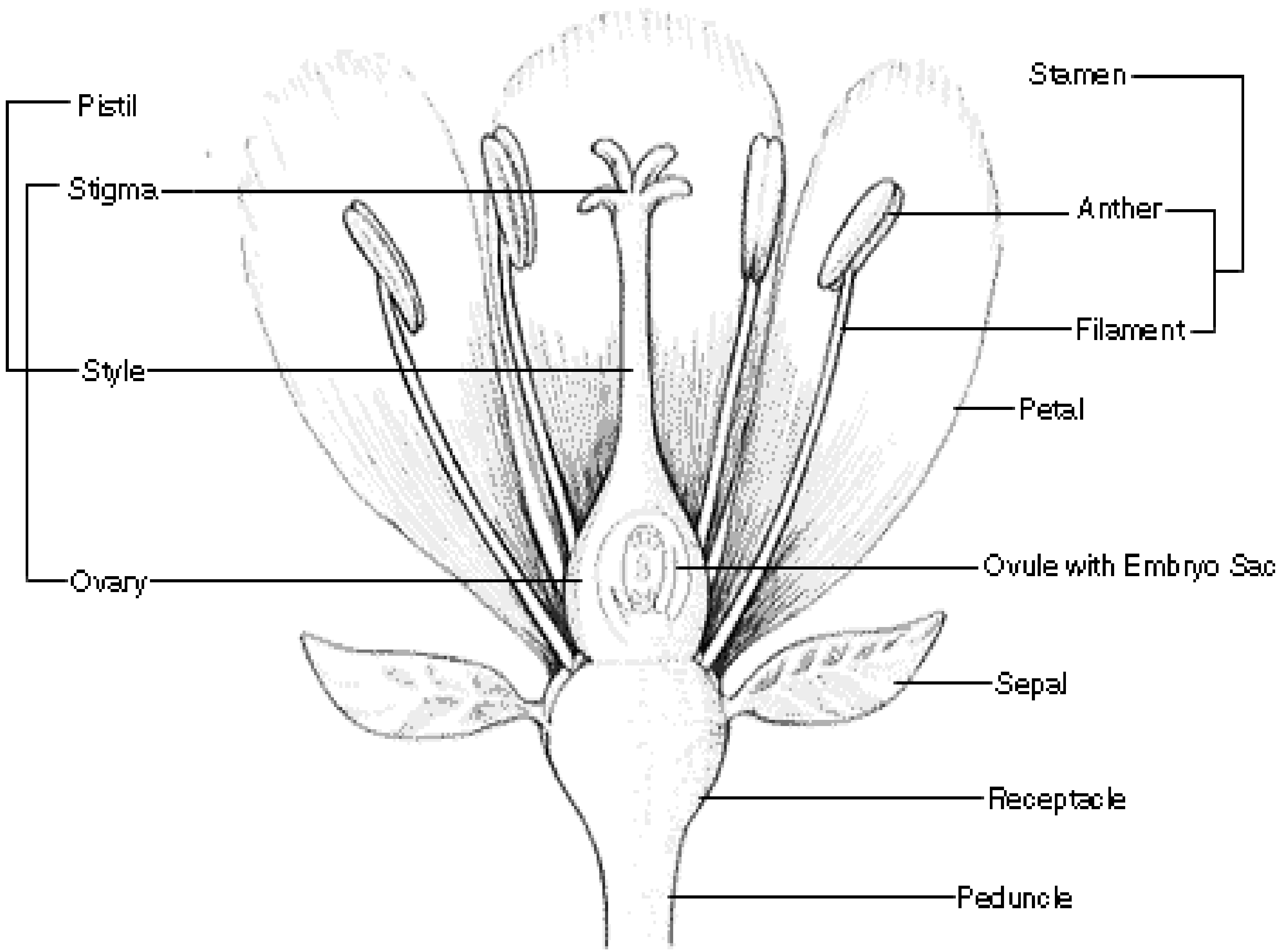
Vladimir Kulyukin
Department of Computer Science
Utah State University

Sklearn Datasets

- We will start out study of supervised learning with Iris Dataset
- Iris Dataset is one of the so-called toy datasets
- Do not be taken aback by the term “toy”: in the sklearn jargon it simply means that the installation of this dataset does not require the installation of any external resources
- All sklearn datasets have the same structure – so, once you understand how one is organized, you can work with all of them or create your own

Iris Dataset from SKLEARN.DATASETS

Flower Morphology



Source https://www.amnh.org/learn/biodiversity_counts/ident_help/Parts_Plants/parts_of_flower.htm

Iris Dataset Details

- The dataset consists of 150 data items
- Data items are iris plants of three species: setosa, versicolor, and virgnica
- Each plant is characterized by four numerical attributes (**features**): sepal length (cm), sepal width (cm), petal length (cm), petal width (cm): many thanks to all those meticulous botanists who did this classification manually!
- Each plant is labeled by its species (in sklearn lingo, the class/species label is called a **target**)
- We want to develop an algorithm that predicts the species of an iris plant from its measurements

Iris Flower Species

Iris Setosa

Iris Versicolor

Iris Virginica

Sources
https://en.wikipedia.org/wiki/Iris_setosa
https://en.wikipedia.org/wiki/Iris_versicolor
https://en.wikipedia.org/wiki/Iris_virginica

Loading Iris Dataset

```
from sklearn.datasets import load_iris
iris_data = load_iris()
## load_iris returns an object with several standard fields:
## iris_data.feature_names – array of strings
## iris_data.data – array of data items (feature vectors)
## data.target_names – array of strings (names of species)
## data.target – array of target numbers
```

Py source is in investigate_iris_dataset.py

Printing Feature Names

```
def print_feature_names(data):
    feature_names = data.feature_names
    print type(feature_names)
    print len(feature_names)
    print 'feature names:'
    print feature_names
```

```
>>> print_feature_names(iris_data)
<type 'list'>
4
feature names:
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

Printing Data Items (AKA Feature Vectors)

```
def print_data_items(data):
    data_items = data.data
    print type(data_items)
    print data_items.shape
    print 'data items:'
    print data_items
```

```
>>> print_data_items(iris_data)
<type 'numpy.ndarray'>
(150, 4)
data items:
[[ 5.1  3.5  1.4  0.2]
 [ 4.9  3.   1.4  0.2]
 ...]
```

Each data item (aka feature vector) is an array of 4 floats: 0^{th} float is sepal length, 1^{st} float is sepal width, 2^{nd} float is petal length, 3^{rd} float is petal width; all values are in cm

Aligning Feature Names with Feature Vector Values

```
>>> iris_data = load_iris()
>>> iris_data.feature_names
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
>>> iris_data.data[:10]
array([[ 5.1,  3.5,  1.4,  0.2],
       [ 4.9,  3. ,  1.4,  0.2],
       [ 4.7,  3.2,  1.3,  0.2],
       [ 4.6,  3.1,  1.5,  0.2],
       [ 5. ,  3.6,  1.4,  0.2],
       [ 5.4,  3.9,  1.7,  0.4],
       [ 4.6,  3.4,  1.4,  0.3],
       [ 5. ,  3.4,  1.5,  0.2],
       [ 4.4,  2.9,  1.4,  0.2],
       [ 4.9,  3.1,  1.5,  0.1]])
```

Let's align feature vector values with feature names:

- 1st feature vector value is sepal length;
- 2nd feature vector value is sepal width;
- 3rd feature vector value is petal length;
- 4th feature value vector is petal width.

Printing Target Names

(3,) means that
Target names is
1D array with
3 elements

```
def print_target_names(data):
    target_names = data.target_names
    print type(target_names)
    print target_names.shape
    print target_names
```

```
>>> print_target_names(iris_data)
<type 'numpy.ndarray'>
(3,)
['setosa' 'versicolor' 'virginica']
```

Target names are the names
of three species: setosa,
versicolor, virginica

Printing Data Target

```
def print_data_target(data):
    target = data.target
    print type(target)
    print target.shape
    print target
```

0 – setosa
1 – versicolor
2 – virginica

```
>>> print_data_target(iris_data)
<type 'numpy.ndarray'>
(150,)
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

How To Get Feature Vectors for a Particular Target

this is a boolean index that retrieves
all feature vectors whose target is 0

```
iris_data.data[iris_data.target == 0]
```

iris_data.data is a numpy array that contains 4-element
numpy arrays (i.e., feature vectors) of flowers

Problem

Write a function **print_data_items_classified_as(data, target_name)**. This function retrieves and displays all flower items of a given species in the iris dataset. For example, **print_data_items_classified_as(iris_data, 'virginica')** retrieves and displays all iris virginica flowers, i.e., their feature vectors.

Print All Items Of A Given Specifies

```
def print_data_items_classified_as(data, target_name):
    target_names = data.target_names
    data_items = data.data
    target = data.target
    try:
        t = target_names.tolist().index(target_name).
        target_items = data_items[target==t]
        print target_items.shape
        print 'All data items classified as', target_name
        print target_items
    except Exception as e:
        print e
```

get all target items classified as t, where t is 0, 1, or 2: 0 for setosa, 1 for versicolor, 2 for virginica

Py source is in investigate_iris_dataset.py

Problem

Write a function **get_all_feature_values_for_target(data, target_number, feature_number)**. This function retrieves all iris flowers classified as **target_number** and then retrieves and displays an array of feature values of **feature_number** for each flower. For example, **get_all_feature_values_for_target(iris_data, 1, 3)** retrieves the values of petal width (feature_number is 3) of all iris versicolor flowers (target_number is 1)

Getting All Feature Values for a Target

tn (target number) is 0, 1, or 2: 0 for setosa, 1 for versicolor, 2 for virginica;

```
def get_all_feature_values_for_target(data, tn, fn):
    data_items = data.data
    target = data.target
    feature_vals = data_items[target==tn,fn]
    print feature_vals
```

fn (feature number) is 0, 1, 2, 3: 0 for sepal length, 1 for sepal width, 2 for petal length, 3 for petal width

Py source is in investigate_iris_dataset.py

Problem

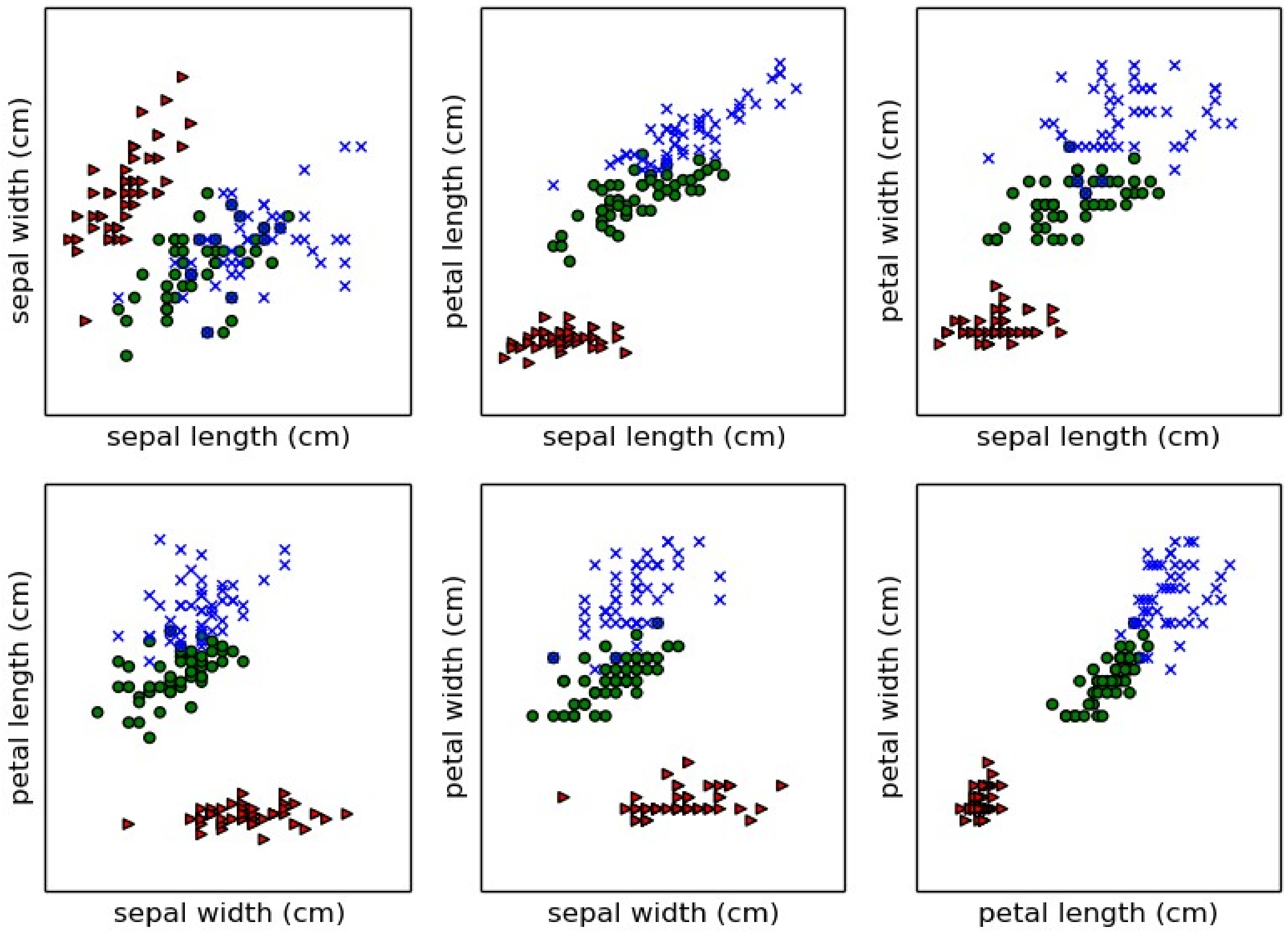
Write a function **save_feature_vs_feature_plot_as_figure(png_image_path)** that creates a figure with six feature vs feature plots (1. sepal length vs sepal width; 2. sepal length vs petal length; 3. sepal length vs petal width; 4. sepal width vs petal length; 5. sepal width vs petal width; 6. petal length vs petal width) and then saves the figure with six plots in a png file specified by **png_image_path** .

Py source is in plot_iris_dataset_features.py

Sample Call

```
$ python plot_iris_dataset_features.py plot.png
```

Sample Output



Solution

```
data = load_iris()
feature_names = data.feature_names
data_items = data.data
target_names = data.target_names

## we will do a figure of 2 rows and 3 columns, i.e.,
## each row will have 3 plots
fn_pairs = [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)]

# set up 3 different pairs of (color, marker)
color_markers = [
        ('r', '>'),        ## setosa is red triangle
        ('g', 'o'),        ## versicolor is green circle
        ('b', 'x'),        ## virginica is blue x
        ]
```

Py source is in plot_iris_dataset_features.py

Solution

```
def save_feature_vs_feature_plot_as_figure(png_image_path):
    fig, axes = plt.subplots(2, 3)
    for i, (fn0, fn1) in enumerate(fn_pairs):
        ax = axes.flat[i]
        for t in xrange(3): # there are 4 target classes (i.e., species): 0, 1, 2, 3
            c, marker = color_markers[t]          # use a different color/marker for each class t
            x = data_items[target==t, fn0]        # find all feature values for target==t and feature fn0
            y = data_items[target==t, fn1]        # find all feature values for target==t and feature fn1
            ax.scatter(x, y, marker=marker, c=c) # scatter plot x vs y
        ax.set_xlabel(feature_names[fn0])  # label x axis as the name of feature whose number is fn0
        ax.set_ylabel(feature_names[fn1])  # label y axis as the name of feature whose number is fn1
        ## eliminate the ticks on both axes
        ax.set_xticks([])
        ax.set_yticks([])
    fig.tight_layout()
    fig.savefig(png_image_path)
```

Py source is in plot_iris_dataset_features.py

Separating Iris Setosas

- The six visualization plots indicate that it is possible to separate iris setosas on the basis of petal length
- We need to discover what the cutoff value is
- We can do it by computing the maximum value of all setosa petal lengths and the minimum value of all non-setosa petal lengths

Separating Iris Setosas

```
from sklearn.datasets import load_iris
data = load_iris()

# load_iris returns an object with several fields
data_items = data.data
feature_names = data.feature_names
target = data.target
target_names = data.target_names

petal_lengths = data_items[:, 2]
flower_names = target_names[target]
flower_names = target_names[target]

max_setosa = petal_lengths[is_setosa].max()
min_non_setosa = petal_lengths[~is_setosa].min()
print("Maximum of setosa's petal lengths: {0}.".format(max_setosa))
print("Minimum of others's petal lengths: {0}.".format(min_non_setosa))
```

this is a boolean index of setosas

Py source is in iris_model_01.py

Petal Length Model for Iris Setosa

If the petal length of a flower is less than 2, it is a setosa.
Else it is either a virginica or versicolor.

```
python iris_model_01.py
Maximum of setosa's petal lengths: 1.9.
Minimum of others's petal lengths: 3.0.
```

Computing Model's Accuracy

```
## petal length model to separate setosas from non-setosas.
def petal_length_model_for_setosa(petal_length_vals):
    classification_results = []
    for pl in petal_length_vals:
        if pl < 2:
            classification_results.append('setosa')
        else:
            classification_results.append('non-setosa')
    return classification_results

## here is how one can compute the accuracy of our model to
## separate setosas from versicolors and virginicas.
def compute_petal_length_model_for_setosa_accuracy(petal_length_vals):
    model_counts = petal_length_model_for_setosa(petal_length_vals).count('setosa')
    data_counts = flower_names.tolist().count('setosa')
    return model_counts/float(data_counts)
```

Py source is in iris_model_01.py

Computing Model's Accuracy

```
print("Model's accuracy: {0}.".format(compute_petal_length_model_for_setosa_accuracy(petal_lengths)))
```

Output

```
Model's accuracy: 1.0.
```

Py source is in iris_model_01.py

References

- http://scikit-learn.org/stable/datasets/index.html
- W. Richert & L. Coelho. “Building ML Systems with Python”, Ch. 2, Pack, 2013.