Python Libraries for ML @ GCE Raipur

Instructor: Santosh Chapaneri

Mar 2021

Scikit-learn Library for ML

Logistic Regression

In [1]:
```
import numpy as np
import pandas as pd
import sklearn
import sklearn.linear_model as lm
import sklearn.model_selection as cv
import matplotlib.pyplot as plt
%matplotlib inline
```

In [3]:
```
titanic = pd.read_csv('titanic.csv')
titanic.head()
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

- We will keep only a few fields for this example.
- We also convert the sex field to a binary variable, so that it can be handled correctly by NumPy and scikit-learn.
- Finally, we remove the rows containing NaN values.

In [4]:
```
data = titanic[['Sex', 'Age', 'Pclass', 'Survived']].copy()
data['Sex'] = data['Sex'] == 'female'
data = data.dropna()
data.head()
```

Out[4]:

	Sex	Age	Pclass	Survived
0	False	22.0	3	0
1	True	38.0	1	1
2	True	26.0	3	1
3	True	35.0	1	1
4	False	35.0	3	0

- Now, we convert this DataFrame to a NumPy array, so that we can pass it to scikit-learn.

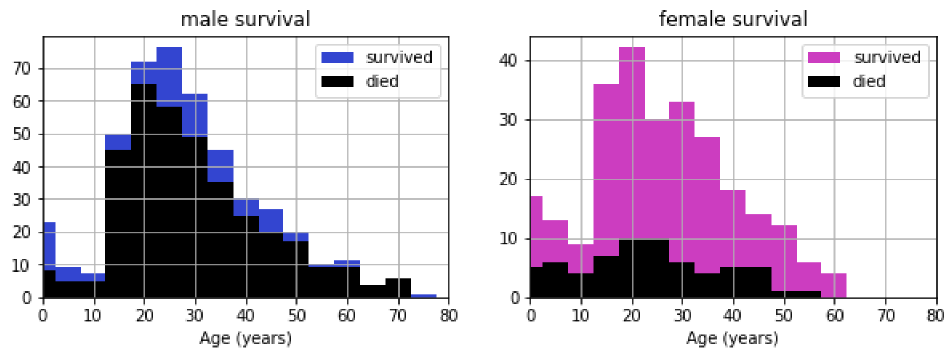
In [5]:
```
data_np = data.astype(np.int32).values
X = data_np[:,:-1] # Features
y = data_np[:,-1] # Target (survived or not)
print(X[:5,:])
print(y[:5])
```

```
[[ 0 22  3]
 [ 1 38  1]
 [ 1 26  3]
 [ 1 35  1]
 [ 0 35  3]]
[0 1 1 1 0]
```

- Let's have a look at the survival of male and female passengers, as a function of their age:

In [6]:
```
# Need few boolean vectors
female = (X[:,0] == 1)
survived = (y == 1)

# This vector contains the age of the passengers.
age = X[:,1]

# Compute few histograms
mybins = np.arange(0, 81, 5)
S = {'male': np.histogram(age[survived & ~female],
                          bins=mybins)[0],
     'female': np.histogram(age[survived & female],
                            bins=mybins)[0]}
D = {'male': np.histogram(age[~survived & ~female],
                          bins=mybins)[0],
     'female': np.histogram(age[~survived & female],
                            bins=mybins)[0]}
```

In [7]:
```
# Now plot the data
bins = mybins[:-1]
plt.figure(figsize=(10,3));
for i, sex, color in zip((0, 1),
                         ('male', 'female'),
                         ('#3345d0', '#cc3dc0')):
    plt.subplot(121 + i);
    plt.bar(bins, S[sex], bottom=D[sex], color=color,
            width=5, label='survived');
    plt.bar(bins, D[sex], color='k', width=5, label='died');
    plt.xlim(0, 80);
    plt.grid(None);
    plt.title(sex + " survival");
    plt.xlabel("Age (years)");
    plt.legend();
```



- Let's train a LogisticRegression classifier. We first need to create a train and a test dataset.

In [8]:
```
# Split X and y into train and test datasets
(X_train, X_test, y_train, y_test) = cv.train_test_split(X, y, test_size=.25)
```

In [9]:
```
# Instantiate the classifier / 'Create the model'
logreg = lm.LogisticRegression()
```

- Train the model (fit) and get the predicted values (predict) on the test set.

In [10]:
```
logreg.fit(X_train, y_train)
```

Out[10]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [11]:
```
y_predicted = logreg.predict(X_test)
```

In [12]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_predicted)
```

Out[12]:
```
0.8156424581005587
```

In [13]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_predicted))
```

```
              precision    recall  f1-score   support

           0       0.84      0.85      0.85       108
           1       0.77      0.76      0.77        71

    accuracy                           0.82       179
   macro avg       0.81      0.81      0.81       179
weighted avg       0.82      0.82      0.82       179
```

In [14]:
```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_predicted))
```

```
[[92 16]
 [17 54]]
```

Run Logistic Regression on the IRIS Dataset

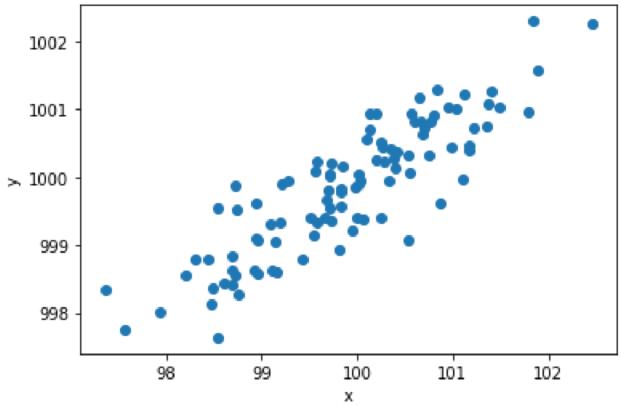
In [15]:
```
import sklearn.datasets as ds
iris = ds.load_iris()

X, y = iris.data, iris.target
(X_train, X_test, y_train, y_test) = cv.train_test_split(X, y, test_size=.25)

logreg = lm.LogisticRegression()
logreg.fit(X_train, y_train)
y_predicted = logreg.predict(X_test)

print(accuracy_score(y_test, y_predicted))
print(classification_report(y_test, y_predicted))
print(confusion_matrix(y_test, y_predicted))
```

```
1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       1.00      1.00      1.00        14
           2       1.00      1.00      1.00         9

    accuracy                           1.00        38
   macro avg       1.00      1.00      1.00        38
weighted avg       1.00      1.00      1.00        38

[[15  0  0]
 [ 0 14  0]
 [ 0  0  9]]

/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Linear Regression

In [16]:
```
import numpy as np
import pandas as pd
import sklearn
import sklearn.linear_model as lm
import sklearn.model_selection as cv
import matplotlib.pyplot as plt
import scipy.stats
%matplotlib inline
```

In [17]:
```
rng = np.random.RandomState(123)
mean = [100, 1000]
cov = [[1, 0.9], [0.9, 1]]
sample = rng.multivariate_normal(mean, cov, size=100)
x, y = sample[:, 0], sample[:, 1]

plt.scatter(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



In [18]:
```
# x.shape => (100,)
# newaxis: increase the dimension of existing array by one more dimension
X = x[:, np.newaxis]
# X.shape => (100,1)
X.shape
```

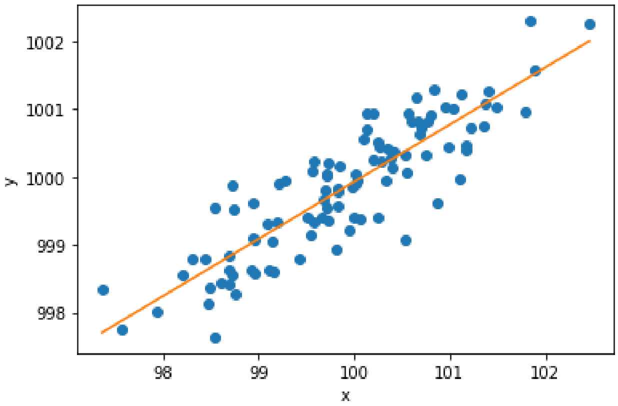
Out[18]: (100, 1)

In [19]:
```
# adding a column vector of "ones
# hstack: stack arrays in sequence horizontally (column wise)
Xb = np.hstack((np.ones((X.shape[0], 1)), X))
# Xb.shape => (100, 2); first column for bias, second column for X
w = np.zeros(Xb.shape[1])

# Closed-form solution
z = np.linalg.inv(np.dot(Xb.T, Xb))

w = np.dot(z, np.dot(Xb.T, y))

b, w1 = w[0], w[1]

print('slope: %.2f' % w1)
print('y-intercept: %.2f' % b)
```

```
slope: 0.84
y-intercept: 915.59
```

- Show the line fit:

In [20]:
```
extremes = np.array([np.min(x), np.max(x)])
predict = extremes*w1 + b

plt.plot(x, y, marker='o', linestyle='')
plt.plot(extremes, predict)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



In [21]:
```
y_predicted = x*w1 + b

mse = np.mean((y - y_predicted)**2)
mse
```

Out[21]: 0.21920128791623675

Support Vector Machines (SVM)

In [22]:
```
import numpy as np
import pandas as pd
import sklearn.datasets as ds
import sklearn.model_selection as cv
import sklearn.svm as svm
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline
```

- We generate 2D points and assign a binary label according to a linear operation on the coordinates.

In [23]:
```
X = np.random.randn(200, 2)
y = (X[:, 0] + X[:, 1]) > 1 # Imagine an AND function

X[:10], y[:10]
```

Out[23]:
```
(array([[ 0.10099719, -0.29817764],
        [-0.25311934,  0.27369246],
        [ 1.44501533, -0.42040294],
        [ 1.21945629, -0.11666772],
        [-1.09791891,  2.50445651],
        [ 0.07068561,  0.75997489],
        [-1.991155  ,  1.06416644],
        [-1.69995175, -1.35561656],
        [ 1.3107086 , -2.1062814 ],
        [-0.12962302, -0.01251449]]),
 array([False, False,  True,  True,  True, False, False, False, False,
        False]))
```

- Fit a linear Support Vector Classifier (SVC)

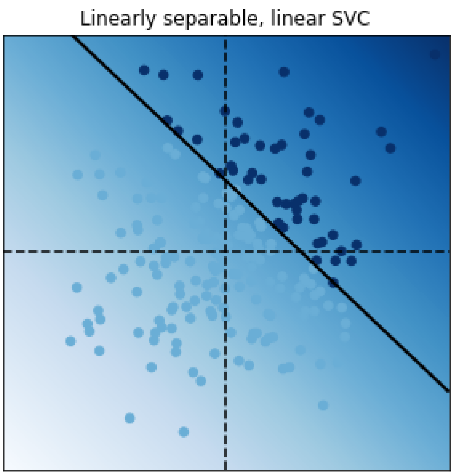
In [24]:
```
est = svm.LinearSVC()
est.fit(X, y)
```

Out[24]:
```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
```

In [25]:
```
# Generate a grid in the square [-3,3 ]^2
xx, yy = np.meshgrid(np.linspace(-3, 3, 500),
                     np.linspace(-3, 3, 500))

# This function takes a SVM estimator as input
def plot_decision_function(est, X, y):
    # We evaluate the decision function on the grid.
    Z = est.decision_function(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    cmap = plt.cm.Blues

    # Display the decision function on the grid
    plt.figure(figsize=(5,5));
    plt.imshow(Z,
               extent=(xx.min(), xx.max(), yy.min(), yy.max()),
               aspect='auto', origin='lower', cmap=cmap)

    # Display the boundaries
    plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='k')

    # Display the points with their true labels
    plt.scatter(X[:, 0], X[:, 1], s=30, c=.5+.5*y, lw=1,
               cmap=cmap, vmin=0, vmax=1);
    plt.axhline(0, color='k', ls='--')
    plt.axvline(0, color='k', ls='--')
    plt.xticks(())
    plt.yticks(())
    plt.axis([-3, 3, -3, 3])
```

In [26]:
```
plot_decision_function(est, X, y)
plt.title("Linearly separable, linear SVC")
```

Out[26]:
```
Text(0.5, 1.0, 'Linearly separable, linear SVC')
```

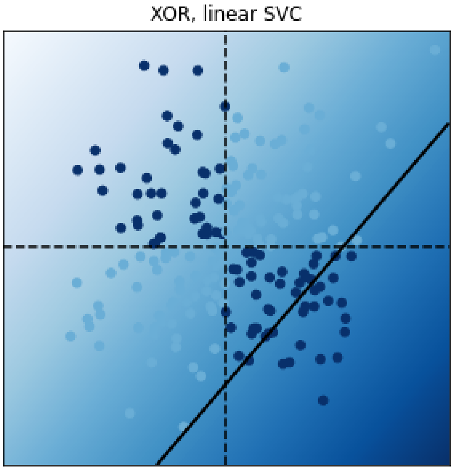


The linear SVC tried to separate the points with a line and it did a good job.

- We now modify the labels with a XOR function.
- A point's label is 1 if the coordinates have different signs. This classification is not linearly separable. Therefore, a linear SVC fails completely.

In [27]:
```
y = np.logical_xor(X[:, 0] > 0, X[:, 1] > 0)

est2 = svm.LinearSVC()
est2.fit(X, y)

plot_decision_function(est2, X, y)
plt.title("XOR, linear SVC")
```

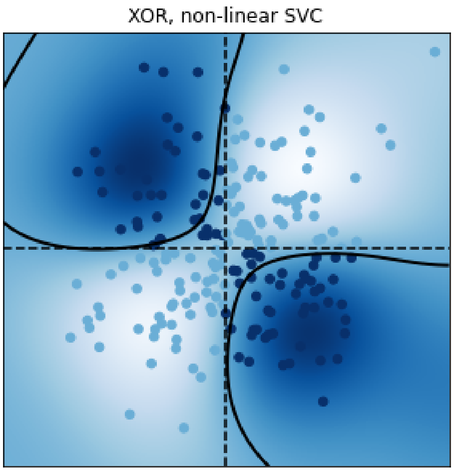
Out[27]: Text(0.5, 1.0, 'XOR, linear SVC')



- It is possible to use non-linear SVCs by using non-linear kernels.
- Kernels specify a non-linear transformation of the points into a higher-dimensional space. Transformed points in this space are assumed to be more linearly separable, although they are not necessarily in the original space.
- By default, the SVC classifier in scikit-learn uses the Radial Basis Function (RBF) kernel.

In [28]:
```
y = np.logical_xor(X[:, 0] > 0, X[:, 1] > 0)

est3 = svm.SVC()
est3.fit(X, y)

plot_decision_function(est3, X, y)
plt.title("XOR, non-linear SVC")
```

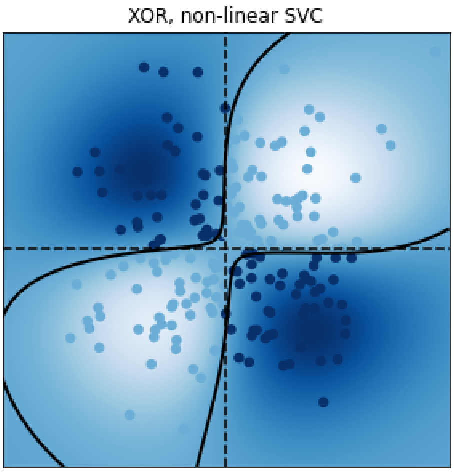
Out[28]: Text(0.5, 1.0, 'XOR, non-linear SVC')



In [29]:
```
from sklearn.model_selection import GridSearchCV

est4 = GridSearchCV(svm.SVC(),
                      {'C': np.logspace(-3., 3., 10),
                       'gamma': np.logspace(-3., 3., 10)}, cv=5);
est4.fit(X, y)
print("Score: {0:.3f}".format(
      cv.cross_val_score(est4, X, y).mean()))

plot_decision_function(est4.best_estimator_, X, y)
plt.title("XOR, non-linear SVC")
```

```
Score: 0.940
```

Out[29]:
```
Text(0.5, 1.0, 'XOR, non-linear SVC')
```



Applying SVM on IRIS dataset

In [30]:
```
import pandas as pd
import numpy as np
from sklearn import svm, datasets
import matplotlib.pyplot as plt
%matplotlib inline

iris = datasets.load_iris()
X = iris.data
y = iris.target
```

- Let's do **hyper-parameter tuning**
- Use **5-fold cross validation to perform grid search to calculate optimal hyper-parameters**

In [31]:
```
from sklearn.model_selection import GridSearchCV
import sklearn.model_selection as cv
from sklearn.metrics import classification_report
from sklearn.utils import shuffle

# Split the dataset
X_train, X_test, y_train, y_test = cv.train_test_split(X, y, test_size=0.25)
```

In [32]:
```
# Set the parameters by cross-validation
parameters = [{'kernel': ['rbf'],
               'gamma': [1e-4, 1e-3, 0.01, 0.1, 0.2, 0.5],
                'C': [1, 10, 100, 1000]},
              {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]

print("# Tuning hyper-parameters")
print()

clf = GridSearchCV(svm.SVC(decision_function_shape='ovr'), parameters, cv=5)
clf.fit(X_train, y_train)
```

```
# Tuning hyper-parameters
```

Out[32]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid=[{'C': [1, 10, 100, 1000],
                          'gamma': [0.0001, 0.001, 0.01, 0.1, 0.2, 0.5],
                          'kernel': ['rbf']},
                         {'C': [1, 10, 100, 1000], 'kernel': ['linear']}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [33]:
```
print("Best parameters set found on training set:")
print()
print(clf.best_params_)
print()
print("Grid scores on training set:")
print()
means = clf.cv_results_['mean_test_score']
stds = clf.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, clf.cv_results_['params']):
    print("%0.3f (+/-%0.03f) for %r"
          % (mean, std * 2, params))
print()
```

```
Best parameters set found on training set:

{'C': 1, 'gamma': 0.5, 'kernel': 'rbf'}

Grid scores on training set:

0.357 (+/-0.015) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.687 (+/-0.014) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.919 (+/-0.157) for {'C': 1, 'gamma': 0.01, 'kernel': 'rbf'}
0.955 (+/-0.141) for {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
0.973 (+/-0.072) for {'C': 1, 'gamma': 0.2, 'kernel': 'rbf'}
0.974 (+/-0.070) for {'C': 1, 'gamma': 0.5, 'kernel': 'rbf'}
0.687 (+/-0.014) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.919 (+/-0.157) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.955 (+/-0.141) for {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
0.964 (+/-0.087) for {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
0.955 (+/-0.114) for {'C': 10, 'gamma': 0.2, 'kernel': 'rbf'}
0.955 (+/-0.114) for {'C': 10, 'gamma': 0.5, 'kernel': 'rbf'}
0.919 (+/-0.157) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.955 (+/-0.141) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.955 (+/-0.114) for {'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
0.955 (+/-0.114) for {'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
0.964 (+/-0.087) for {'C': 100, 'gamma': 0.2, 'kernel': 'rbf'}
0.955 (+/-0.114) for {'C': 100, 'gamma': 0.5, 'kernel': 'rbf'}
0.955 (+/-0.141) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.955 (+/-0.114) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.955 (+/-0.114) for {'C': 1000, 'gamma': 0.01, 'kernel': 'rbf'}
0.974 (+/-0.070) for {'C': 1000, 'gamma': 0.1, 'kernel': 'rbf'}
0.956 (+/-0.080) for {'C': 1000, 'gamma': 0.2, 'kernel': 'rbf'}
0.946 (+/-0.105) for {'C': 1000, 'gamma': 0.5, 'kernel': 'rbf'}
0.946 (+/-0.145) for {'C': 1, 'kernel': 'linear'}
0.955 (+/-0.114) for {'C': 10, 'kernel': 'linear'}
0.955 (+/-0.114) for {'C': 100, 'kernel': 'linear'}
0.955 (+/-0.114) for {'C': 1000, 'kernel': 'linear'}
```

In [34]:
```
print("Detailed classification report:")
print()
print("The model is trained on the training set.")
print("The scores are computed on the testing set.")
print()
y_true, y_pred = y_test, clf.predict(X_test)
print(classification_report(y_true, y_pred))
print()
# The support is the number of occurrences of each class in y_true
```

```
Detailed classification report:

The model is trained on the training set.
The scores are computed on the testing set.

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       0.93      0.93      0.93        15
           2       0.90      0.90      0.90        10

    accuracy                           0.95        38
   macro avg       0.94      0.94      0.94        38
weighted avg       0.95      0.95      0.95        38
```

In [35]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_true, y_pred)
```

Out[35]:
```
0.9473684210526315
```

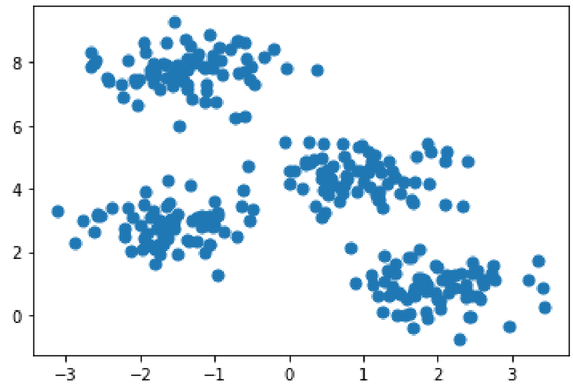
K-Means Clustering

- Here we'll explore K Means Clustering, which is an unsupervised clustering technique.
- K-Means is an algorithm for unsupervised clustering: that is, finding clusters in data based on the data attributes alone (not the labels).
- K-Means is a relatively easy-to-understand algorithm. It searches for cluster centers which are the mean of the points within them, such that every point is closest to the cluster center it is assigned to.

In [36]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from scipy import stats

# use seaborn plotting defaults
import seaborn as sns
```

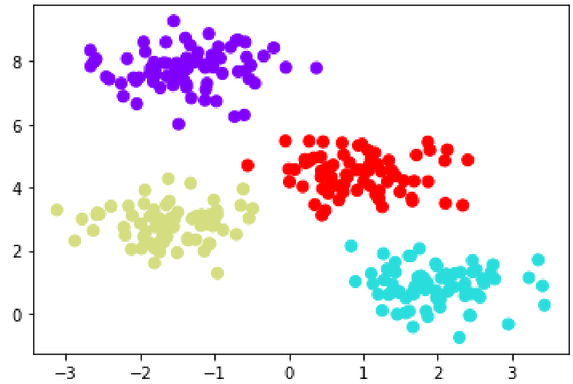
In [37]:
```
from sklearn.datasets.samples_generator import make_blobs

X, y = make_blobs(n_samples=300, centers=4,
                  random_state=0, cluster_std=0.60)

plt.scatter(X[:, 0], X[:, 1], s=50);
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning: The sklear
n.datasets.samples_generator module is  deprecated in version 0.22 and will be removed in version
0.24. The corresponding classes / functions should instead be imported from sklearn.datasets. Anyt
hing that cannot be imported from sklearn.datasets is now part of the private API.
  warnings.warn(message, FutureWarning)
```



- By eye, it is relatively easy to pick out the four clusters.
- If we were to perform an exhaustive search for the different segmentations of the data, however, the search space would be exponential in the number of points.
- Fortunately, there is a well-known Expectation Maximization (EM) procedure which scikit-learn implements, so that KMeans can be solved relatively quickly.

In [38]:
```python
from sklearn.cluster import KMeans

est = KMeans(4)

est.fit(X)

y_kmeans = est.predict(X)

plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='rainbow');
```



In [39]:
```python
from sklearn.metrics import calinski_harabasz_score

calinski_harabasz_score(X, y_kmeans)
```

Out[39]: 1210.0899142587816

- The algorithm identifies the four clusters of points in a manner very similar to what we would do by eye!

Application of KMeans to Digits dataset

- Here we'll use K-Means to automatically cluster the data in 64 dimensions, and then look at the cluster centers to see what the algorithm has found.

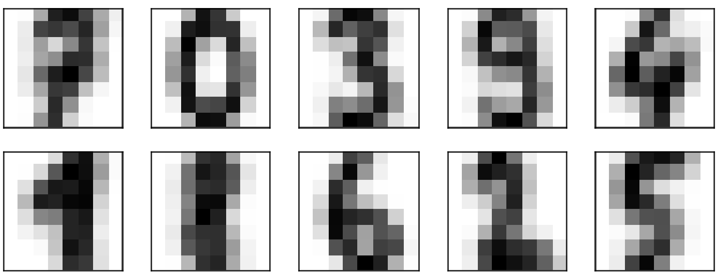
In [40]:
```python
from sklearn.datasets import load_digits
digits = load_digits()
```

In [41]:
```python
est = KMeans(n_clusters=10)

clusters = est.fit_predict(digits.data)

est.cluster_centers_.shape
```

Out[41]: (10, 64)

- We see ten clusters in 64 dimensions. Let's visualize each of these cluster centers to see what they represent:

In [42]:
```python
fig = plt.figure(figsize=(8, 3))
for i in range(10):
    ax = fig.add_subplot(2, 5, 1 + i, xticks=[], yticks=[])
    ax.imshow(est.cluster_centers_[i].reshape((8, 8)), cmap=plt.cm.binary)
```



In [43]:
```
from sklearn.metrics import calinski_harabasz_score

calinski_harabasz_score(digits.data, clusters)
```

Out[43]:
```
169.30606481277894
```

- We see that even without the labels, KMeans is able to find clusters whose means are recognizable digits (sorry to number 8)!

Principal Component Analysis

In [44]:
```
import numpy as np
from sklearn.decomposition import PCA
X = np.array([[1, -1], [0, 1], [-1, 0]])
print(X)
print()
pca = PCA()
pca.fit(X)

print(pca.components_)
# Principal axes representing the directions of maximum variance in the data
print()
print(pca.explained_variance_ratio_)
# Percentage of variance explained by each component
print()
X_new = pca.transform(X)
print(X_new)
```

```
[[ 1 -1]
 [ 0  1]
 [-1  0]]

[[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]

[0.75 0.25]

[[ 1.41421356e+00 -3.33066907e-16]
 [-7.07106781e-01  7.07106781e-01]
 [-7.07106781e-01 -7.07106781e-01]]
```

In [45]:
```
import numpy as np
from sklearn.decomposition import PCA
X = np.array([[-1, -1, 5], [-2, -1, 8], [-3, -2, 6],
              [1, 1, 8], [2, 1, 5], [3, 2, 9]])
print(X)
print()
pca = PCA(n_components=2)
pca.fit(X)

print(pca.components_) # n_components x n_features
# Principal axes representing the directions of maximum variance in the data
print()
print(pca.explained_variance_ratio_)
# Percentage of variance explained by each component
print()
X_new = pca.transform(X)
print(X_new)
```

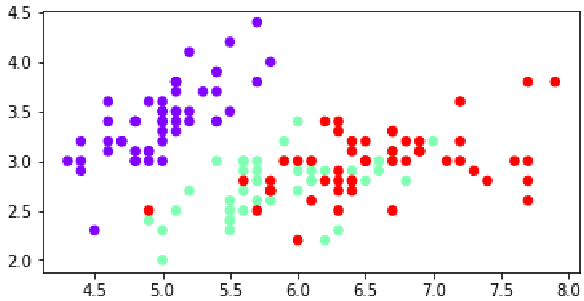
```
[[-1 -1  5]
 [-2 -1  8]
 [-3 -2  6]
 [ 1  1  8]
 [ 2  1  5]
 [ 3  2  9]]

[[ 0.79554567  0.53157493  0.29074936]
 [ 0.31712316  0.04357793 -0.94738264]]

[0.76997386 0.22885582]

[[-1.86016108  1.37616707]
 [-1.78345867 -1.783104  ]
 [-3.69207799 -0.24903982]
 [ 1.66632818 -0.74457865]
 [ 1.58962577  2.41469243]
 [ 4.0797438  -1.01413703]]
```

- PCA on IRIS dataset

In [46]:
```
import sklearn.datasets as ds
import matplotlib.pyplot as plt
%matplotlib inline

iris = ds.load_iris()
X = iris.data
y = iris.target
print(X.shape)

plt.figure(figsize=(6,3));
plt.scatter(X[:,0], X[:,1], c=y, s=30, cmap=plt.cm.rainbow)
```

```
(150, 4)
```

Out[46]: `<matplotlib.collections.PathCollection at 0x7fbed2dde290>`



In [47]:
```
from sklearn.decomposition import PCA
pca = PCA()

X_pca = pca.fit_transform(X)

print(pca.components_)
print()
print(pca.explained_variance_ratio_)
```

```
[[ 0.36138659 -0.08452251  0.85667061  0.3582892 ]
 [ 0.65658877  0.73016143 -0.17337266 -0.07548102]
 [-0.58202985  0.59791083  0.07623608  0.54583143]
 [-0.31548719  0.3197231   0.47983899 -0.75365743]]

[0.92461872 0.05306648 0.01710261 0.00521218]
```

- Now display the same dataset, but in a new coordinate system (or equivalently, a linearly transformed version of the initial dataset).

In [48]:
```
print(X_pca.shape)
```

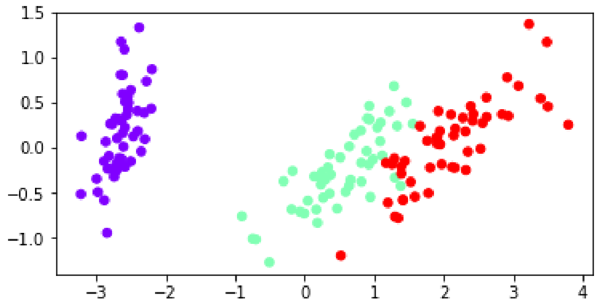
```
(150, 4)
```

In [49]:
```
X_pca[:,0]
# observe that dimensions 2 and 3 are almost zero values
# (change 0th column to 1st or 2nd or 3rd)
```

Out[49]:
```
array([-2.68412563, -2.71414169, -2.88899057, -2.74534286, -2.72871654,
       -2.28085963, -2.82053775, -2.62614497, -2.88638273, -2.6727558 ,
       -2.50694709, -2.61275523, -2.78610927, -3.22380374, -2.64475039,
       -2.38603903, -2.62352788, -2.64829671, -2.19982032, -2.5879864 ,
       -2.31025622, -2.54370523, -3.21593942, -2.30273318, -2.35575405,
       -2.50666891, -2.46882007, -2.56231991, -2.63953472, -2.63198939,
       -2.58739848, -2.4099325 , -2.64886233, -2.59873675, -2.63692688,
       -2.86624165, -2.62523805, -2.80068412, -2.98050204, -2.59000631,
       -2.77010243, -2.84936871, -2.99740655, -2.40561449, -2.20948924,
       -2.71445143, -2.53814826, -2.83946217, -2.54308575, -2.70335978,
        1.28482569,  0.93248853,  1.46430232,  0.18331772,  1.08810326,
        0.64166908,  1.09506066, -0.74912267,  1.04413183, -0.0087454 ,
       -0.50784088,  0.51169856,  0.26497651,  0.98493451, -0.17392537,
        0.92786078,  0.66028376,  0.23610499,  0.94473373,  0.04522698,
        1.11628318,  0.35788842,  1.29818388,  0.92172892,  0.71485333,
        0.90017437,  1.33202444,  1.55780216,  0.81329065, -0.30558378,
       -0.06812649, -0.18962247,  0.13642871,  1.38002644,  0.58800644,
        0.80685831,  1.22069088,  0.81509524,  0.24595768,  0.16641322,
        0.46480029,  0.8908152 ,  0.23054802, -0.70453176,  0.35698149,
        0.33193448,  0.37621565,  0.64257601, -0.90646986,  0.29900084,
        2.53119273,  1.41523588,  2.61667602,  1.97153105,  2.35000592,
        3.39703874,  0.52123224,  2.93258707,  2.32122882,  2.91675097,
        1.66177415,  1.80340195,  2.1655918 ,  1.34616358,  1.58592822,
        1.90445637,  1.94968906,  3.48705536,  3.79564542,  1.30079171,
        2.42781791,  1.19900111,  3.49992004,  1.38876613,  2.2754305 ,
        2.61409047,  1.25850816,  1.29113206,  2.12360872,  2.38800302,
        2.84167278,  3.23067366,  2.15943764,  1.44416124,  1.78129481,
        3.07649993,  2.14424331,  1.90509815,  1.16932634,  2.10761114,
        2.31415471,  1.9222678 ,  1.41523588,  2.56301338,  2.41874618,
        1.94410979,  1.52716661,  1.76434572,  1.90094161,  1.39018886])
```

In [50]:
```
plt.figure(figsize=(6,3))
plt.scatter(X_pca[:,0], X_pca[:,1], c=y, s=30, cmap=plt.cm.rainbow)
```

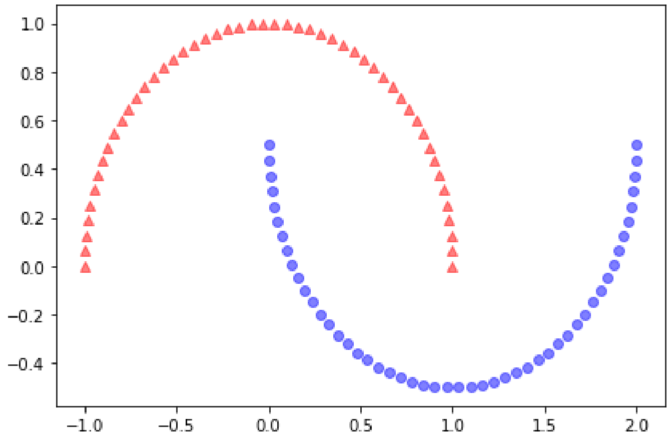
Out[50]:
```
<matplotlib.collections.PathCollection at 0x7fbecfd2e350>
```



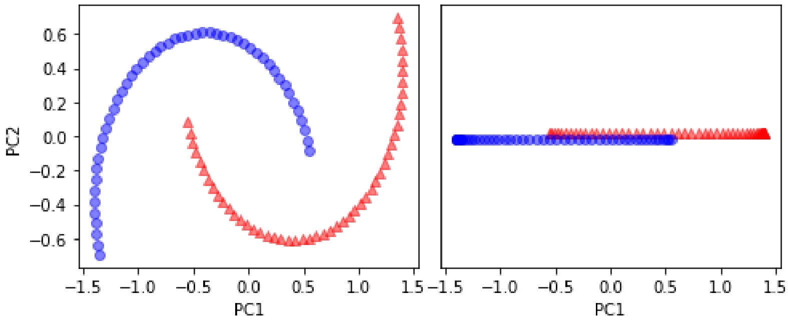
- Points belonging to the same classes are now grouped together, even though the PCA estimator dit not use the labels.
- The PCA was able to find a projection maximizing the variance, which corresponds here to a projection where the classes are well separated.

Kernel PCA

In [51]:
```
from sklearn.decomposition import KernelPCA

X, y = ds.make_moons(n_samples=100)
# Synthetic Data: two interleaving half circles

plt.scatter(X[y == 0, 0], X[y == 0, 1], color='red', marker='^', alpha=0.5)
plt.scatter(X[y == 1, 0], X[y == 1, 1], color='blue', marker='o', alpha=0.5)

plt.tight_layout()
plt.show()
```



In [52]:
```
# First, let's try with PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(7, 3))

ax[0].scatter(X_pca[y == 0, 0], X_pca[y == 0, 1],
              color='red', marker='^', alpha=0.5)
ax[0].scatter(X_pca[y == 1, 0], X_pca[y == 1, 1],
              color='blue', marker='o', alpha=0.5)

ax[1].scatter(X_pca[y == 0, 0], np.zeros((50, 1)) + 0.02,
              color='red', marker='^', alpha=0.5)
ax[1].scatter(X_pca[y == 1, 0], np.zeros((50, 1)) - 0.02,
              color='blue', marker='o', alpha=0.5)

ax[0].set_xlabel('PC1')
ax[0].set_ylabel('PC2')
ax[1].set_ylim([-1, 1])
ax[1].set_yticks([])
ax[1].set_xlabel('PC1')

plt.tight_layout()
plt.show()
```



In [53]:
```
kpca = KernelPCA(n_components=2, kernel='rbf', gamma=15)
X_kpca = kpca.fit_transform(X)

plt.scatter(X_kpca[y == 0, 0], X_kpca[y == 0, 1],
            color='red', marker='^', alpha=0.5)
plt.scatter(X_kpca[y == 1, 0], X_kpca[y == 1, 1],
            color='blue', marker='o', alpha=0.5)

plt.xlabel('PC1')
plt.ylabel('PC2')
plt.tight_layout()
plt.show()
```