Python: Machine Learning Examples

kNN for Sample Data

In [10]:

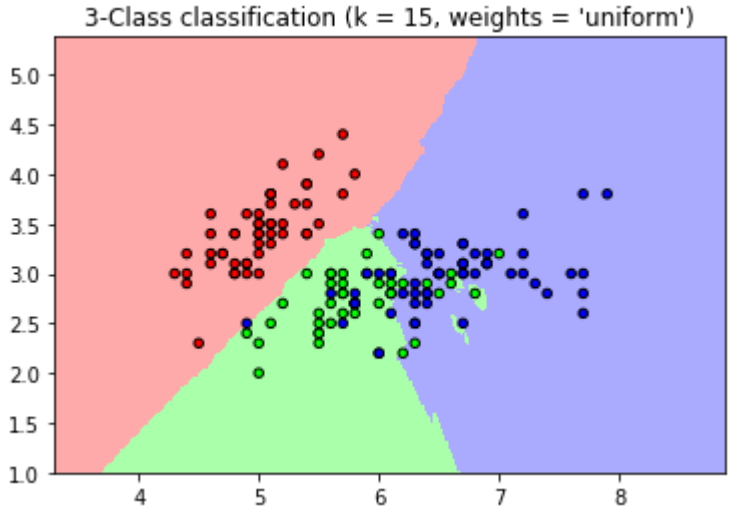
```
from sklearn.neighbors import KNeighborsClassifier

FRAUD = 1
NOT_FRAUD = -1

labels = [FRAUD, FRAUD, NOT_FRAUD, NOT_FRAUD]
obvs = [ [3], [4], [10], [20] ]
# obvs = [ [3, 0], [1, 1] ]


alg = KNeighborsClassifier(1)
alg.fit(obvs, labels)

unknown = [ [7], [8], [15], [2], [30] ]

results = alg.predict(unknown)

print("Predictions for", unknown)
print(results)

for u, r in zip(unknown, results):
    if r == 1:
        print(f'{u[0]} \tFRAUD')
    else:
        print(f'{u[0]} \tNOT FRAUD')
```

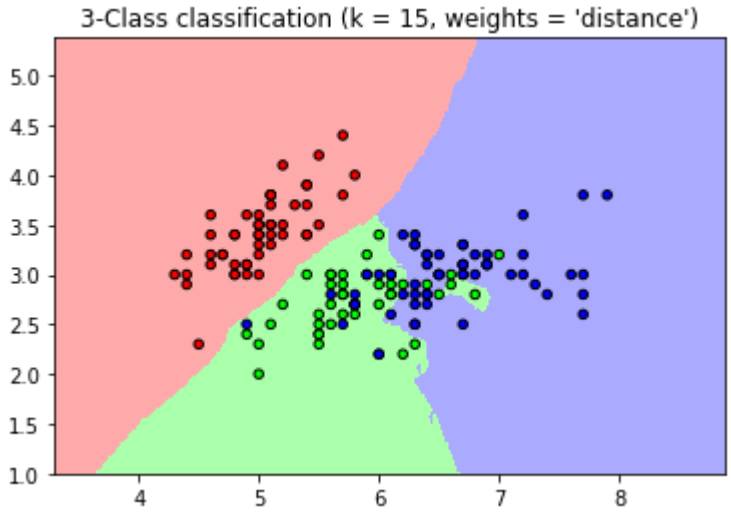
```
Predictions for [[7], [8], [15], [2], [30]]
[ 1 -1 -1  1 -1]
7       FRAUD
8       NOT FRAUD
15      NOT FRAUD
2       FRAUD
30      NOT FRAUD
```

kNN for Iris Flowers

In [16]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets

iris = datasets.load_iris()
X = iris.data[:, :2]
y = iris.target

n_neighbors = 15

h = .02  # step size in the mesh

cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])

for weights in ['uniform', 'distance']:
    clf = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)
    clf.fit(X, y)

    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold, edgecolor='k', s=20)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title(f"3-Class classification (k = {n_neighbors}, weights = '{weights}
')")
plt.show()
```





In [ ]:

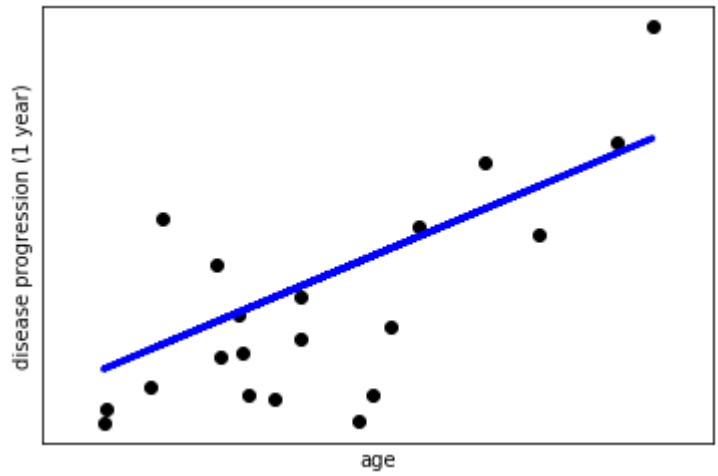
```
print(iris['DESCR'])
```

Linear Regression for Diabetes Prognosis

In [12]:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes = datasets.load_diabetes()

# Use only one feature
x = "age"
y = "disease progression (1 year)"

diabetes_X = diabetes.data[:, np.newaxis, 2]

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# The coefficients
print('Coefficients: \n', regr.coef_)

# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))

# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(diabetes_y_test, diabetes_y_pred))

# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test,  color='black')
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)

plt.xlabel(x)
plt.ylabel(y)
plt.xticks(tuple())
plt.yticks(tuple())

plt.show()
```

```
Coefficients:
 [938.23786125]
Mean squared error: 2548.07
Variance score: 0.47
```



In [13]:

```
print(diabetes['DESCR'])
```

```
.. _diabetes_dataset:

Diabetes dataset
----------------

Ten baseline variables, age, sex, body mass index, average blood
pressure, and six blood serum measurements were obtained for each of
n =
442 diabetes patients, as well as the response of interest, a
quantitative measure of disease progression one year after baseline.

**Data Set Characteristics:**

  :Number of Instances: 442

  :Number of Attributes: First 10 columns are numeric predictive val
ues

  :Target: Column 11 is a quantitative measure of disease progressio
n one year after baseline

  :Attribute Information:
      - Age
      - Sex
      - Body mass index
      - Average blood pressure
      - S1
      - S2
      - S3
      - S4
      - S5
      - S6

Note: Each of these 10 feature variables have been mean centered and
scaled by the standard deviation times `n_samples` (i.e. the sum of
squares of each column totals 1).

Source URL:
http://www4.stat.ncsu.edu/~boos/var.select/diabetes.html

For more information see:
Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani
(2004) "Least Angle Regression," Annals of Statistics (with discussi
on), 407-499.
(http://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)
```

Handwriting Recognition with Logistic Regression

In [ ]:

```
from sklearn.datasets import fetch_mldata
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression


mnist = fetch_mldata('MNIST original')

train_img, test_img, train_lbl, test_lbl = train_test_split(
    mnist.data, mnist.target, test_size=1/7.0, random_state=0
)



scaler = StandardScaler()
scaler.fit(train_img)


train_img = scaler.transform(train_img)
test_img = scaler.transform(test_img)




pca = PCA(.95) # retain 95% of varience, 330 dim
pca.fit(train_img)

train_img = pca.transform(train_img)
test_img = pca.transform(test_img)




logisticRegr = LogisticRegression(solver = 'lbfgs')
logisticRegr.fit(train_img, train_lbl)


# Predict for One Observation (image)
logisticRegr.predict(test_img[0].reshape(1,-1))

# Predict for One Observation (image)
logisticRegr.predict(test_img[0:10])#
logisticRegr.score(test_img, test_lbl)









### IMAGES
approximation = pca.inverse_transform(train_img)
plt.figure(figsize=(8,4))

# Original Image
plt.subplot(1, 2, 1)
plt.imshow(
```

```
    mnist.data[1].reshape(28,28),
    cmap = plt.cm.gray, interpolation='nearest',
    clim=(0, 255)
)

plt.xlabel('784 components', fontsize = 14)
plt.title('Original Image', fontsize = 20)

# 154 principal components
plt.subplot(1, 2, 2);
plt.imshow(
    approximation[1].reshape(28, 28),
    cmap = plt.cm.gray, interpolation='nearest',
    clim=(0, 255)
)

plt.xlabel('154 components', fontsize = 14)
plt.title('95% of Explained Variance', fontsize = 20);

plt.show()
```