Machine Learning Tools and Libraries in Python

1. Scikit-learn

Example1: Least Squares Regression:

In [ ]:

```
from sklearn import linear_model
reg = linear_model.LinearRegression()
X = [[0, 0], [1, 1], [2, 2]]
y = [0, 1, 2]

# fit model
reg.fit(X, y)


#show coefficients of the regression line:
print("coeficients: a=", reg.coef_[0], "b=", reg.coef_[1])


# predict on new data:
X_new = [[0.1, 0.2]]
print("X_new:", X_new)

y_predicted_new = reg.predict(X_new)
print("prediction for X_new:", y_predicted_new)
```

```
coeficients: a= 0.5 b= 0.4999999999999999
X_new: [[0.1, 0.2]]
prediction for X_new: [0.15]
```

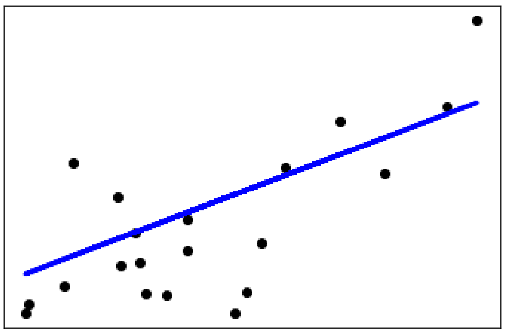
Example2: Least Squares Linear Regression 2

In [ ]:

```
# Code source: Jaques Grobler  https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html#sphx-
# License: BSD 3 clause

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_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

# Use only one feature
diabetes_X = diabetes_X[:, 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_y[:-20]
diabetes_y_test = diabetes_y[-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))
# The coefficient of determination: 1 is perfect prediction
print("Coefficient of determination: %.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.xticks(())
plt.yticks(())

plt.show()
```

```
Coefficients: 
 [938.23786125]
```

```
Mean squared error: 2548.07
Coefficient of determination: 0.47
```



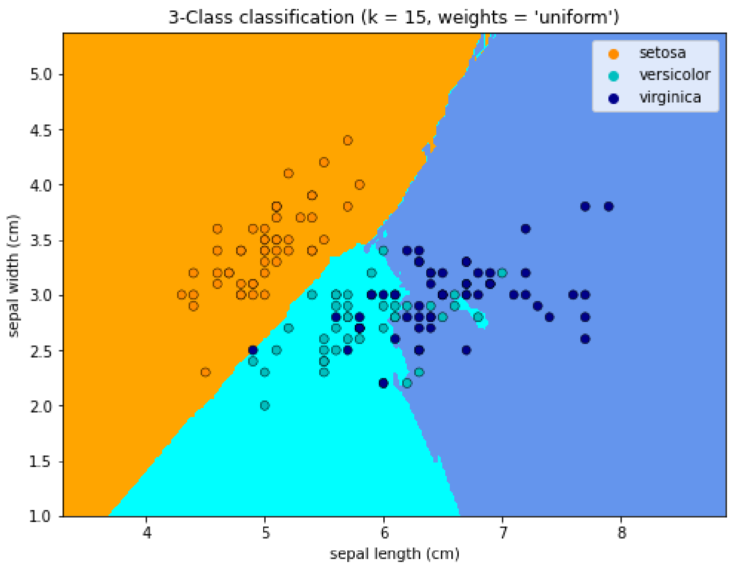
Example 3: Decision Tree Classifier - the *XOR* problem

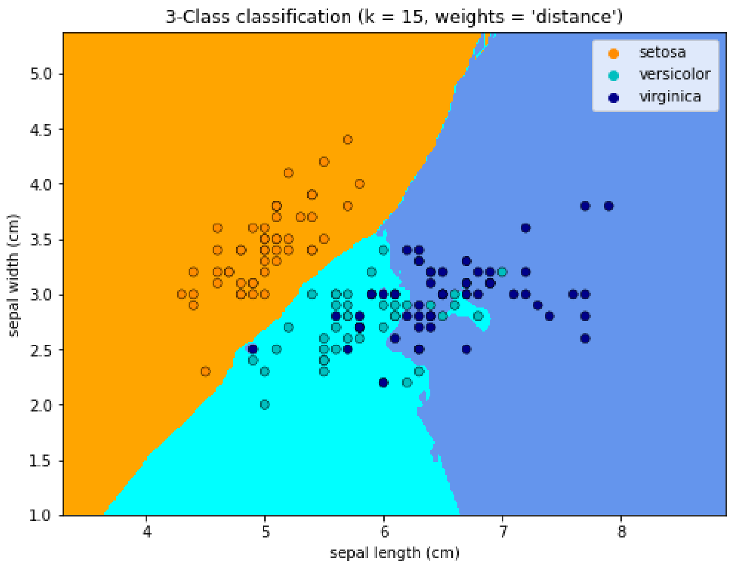
In [ ]:
```python
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(random_state=0)

# XOR problem
X = [[0,0], [0,1], [1,1], [1, 0]]
Y = [0, 1, 0, 1]

clf.fit(X, Y)

clf.predict([[1,0]])
```

Out[ ]:
```
array([1])
```

Example4: Nearest Neighbors Classifier:

In [ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets

n_neighbors = 15

# import some data to play with
iris = datasets.load_iris()

# we only take the first two features. We could avoid this ugly
# slicing by using a two-dim dataset
X = iris.data[:, :2]
y = iris.target

h = 0.02  # step size in the mesh

# Create color maps
cmap_light = ListedColormap(["orange", "cyan", "cornflowerblue"])
cmap_bold = ["darkorange", "c", "darkblue"]

for weights in ["uniform", "distance"]:
    # we create an instance of Neighbours Classifier and fit the data.
    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(figsize=(8, 6))
    plt.contourf(xx, yy, Z, cmap=cmap_light)

    # Plot also the training points
    sns.scatterplot(
        x=X[:, 0],
        y=X[:, 1],
        hue=iris.target_names[y],
        palette=cmap_bold,
        alpha=1.0,
        edgecolor="black",
    )
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title(
        "3-Class classification (k = %i, weights = '%s')" % (n_neighbors, weights)
    )
    plt.xlabel(iris.feature_names[0])
    plt.ylabel(iris.feature_names[1])

plt.show()
```





Scikit-learn resources:

- Official scikit-learn website: https://scikit-learn.org/stable/

2. Tensorflow

Example 1: (tensorflow quickstart) https://www.tensorflow.org/tutorials/quickstart/beginner

In [ ]:
```
import tensorflow as tf
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

In [ ]:
```
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

In [ ]:
```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
```

In [ ]:
```
predictions = model(x_train[:1]).numpy()
predictions
```

Out[ ]:
```
array([[ 0.14117278, -0.24113557,  0.19131026, -0.31744638, -0.33681893,
         0.33033165, -0.43933952,  1.0717434 , -0.24013911,  0.1962855 ]],
      dtype=float32)
```

In [ ]:
```
tf.nn.softmax(predictions).numpy()
```

Out[ ]:
```
array([[0.09970777, 0.06802908, 0.10483431, 0.06303086, 0.06182154,
        0.12047021, 0.05579763, 0.2528545 , 0.0680969 , 0.10535719]],
      dtype=float32)
```

In [ ]:
```
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

In [ ]:
```
model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])
```

In [ ]:
```
model.fit(x_train, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 7s 3ms/step - loss: 0.2927 - accuracy: 0.9138
Epoch 2/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1414 - accuracy: 0.9578
Epoch 3/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1055 - accuracy: 0.9680
Epoch 4/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0873 - accuracy: 0.9729
Epoch 5/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0726 - accuracy: 0.9772
```

Out[ ]:
```
<keras.callbacks.History at 0x7fa4f3e05290>
```

In [ ]:
```
model.evaluate(x_test,  y_test, verbose=2)
```

```
313/313 - 1s - loss: 0.0765 - accuracy: 0.9762 - 539ms/epoch - 2ms/step
```

Out[ ]:
```
[0.07650239765644073, 0.9761999845504761]
```

Example 2: Fashion MNIST https://www.tensorflow.org/tutorials/keras/classification

Tensorflow resources:

- https://www.tensorflow.org/tutorials

3. Keras

Keras resources:

- https://keras.io/examples/

4. PyTorch

In [ ]:
```
# source: https://pytorch.org/tutorials/beginner/examples_nn/polynomial_nn.html#sphx-glr-beginner-examples-nn

import torch
import math


# Create Tensors to hold input and outputs.
x = torch.linspace(-math.pi, math.pi, 2000)
y = torch.sin(x)

# For this example, the output y is a linear function of (x, x^2, x^3), so
# we can consider it as a linear layer neural network. Let's prepare the
# tensor (x, x^2, x^3).
p = torch.tensor([1, 2, 3])
xx = x.unsqueeze(-1).pow(p)

# In the above code, x.unsqueeze(-1) has shape (2000, 1), and p has shape
# (3,), for this case, broadcasting semantics will apply to obtain a tensor
# of shape (2000, 3)

# Use the nn package to define our model as a sequence of layers. nn.Sequential
# is a Module which contains other Modules, and applies them in sequence to
# produce its output. The Linear Module computes output from input using a
# linear function, and holds internal Tensors for its weight and bias.
# The Flatten layer flatens the output of the linear layer to a 1D tensor,
```

```python
# to match the shape of `y`.
model = torch.nn.Sequential(
    torch.nn.Linear(3, 1),
    torch.nn.Flatten(0, 1)
)

# The nn package also contains definitions of popular loss functions; in this
# case we will use Mean Squared Error (MSE) as our loss function.
loss_fn = torch.nn.MSELoss(reduction='sum')

learning_rate = 1e-6
for t in range(2000):

    # Forward pass: compute predicted y by passing x to the model. Module objects
    # override the __call__ operator so you can call them like functions. When
    # doing so you pass a Tensor of input data to the Module and it produces
    # a Tensor of output data.
    y_pred = model(xx)

    # Compute and print loss. We pass Tensors containing the predicted and true
    # values of y, and the loss function returns a Tensor containing the
    # loss.
    loss = loss_fn(y_pred, y)
    if t % 100 == 99:
        print(t, loss.item())

    # Zero the gradients before running the backward pass.
    model.zero_grad()

    # Backward pass: compute gradient of the loss with respect to all the learnable
    # parameters of the model. Internally, the parameters of each Module are stored
    # in Tensors with requires_grad=True, so this call will compute gradients for
    # all learnable parameters in the model.
    loss.backward()

    # Update the weights using gradient descent. Each parameter is a Tensor, so
    # we can access its gradients like we did before.
    with torch.no_grad():
        for param in model.parameters():
            param -= learning_rate * param.grad

# You can access the first layer of `model` like accessing the first item of a list
linear_layer = model[0]

# For linear layer, its parameters are stored as `weight` and `bias`.
print(f'Result: y = {linear_layer.bias.item()} + {linear_layer.weight[:, 0].item()} x + {linear_layer.weight[
```

```
99 1144.9034423828125
199 761.5885009765625
299 507.6730041503906
399 339.45391845703125
499 227.994384765625
599 154.13287353515625
699 105.1794662475586
799 72.72947692871094
899 51.21546173095703
999 36.949501037597656
1099 27.487873077392578
1199 21.211456298828125
1299 17.047086715698242
1399 14.283363342285156
1499 12.448812484741211
1599 11.230703353881836
1699 10.4216947555542
1799 9.884227752685547
1899 9.527064323425293
1999 9.28962230682373
Result: y = 0.007204164285212755 + 0.8366740942001343 x + -0.0012428383342921734 x^2 + -0.09047607332468033 x
^3
```

Example 2: MNIST classification wth PyTorch: https://nextjournal.com/gkoehler/pytorch-mnist

Resources:

- Examples with PyTorch: https://github.com/pytorch/examples/