Image Processing using Machine learning in Python

Dr. P. Mirunalini & Dr. J. Bhuvana

Dept of CSE, SSN College of Engineering

November 12, 2018

Table of contents

Machine Learning

- **Machine Learning:** A branch of artificial intelligence, concerned with the design and development of algorithms that allow computers to evolve behaviors based on empirical data.

Objects: Given image
Classes : Daisy, Fancy, Sunflower and Crocus

Machine Learning Applications

Machine learning is preferred approach to:

- Speech recognition, Natural language processing
- Computer vision
- Medical outcomes analysis
- Robot control
- Computational biology

Machine Learning framework

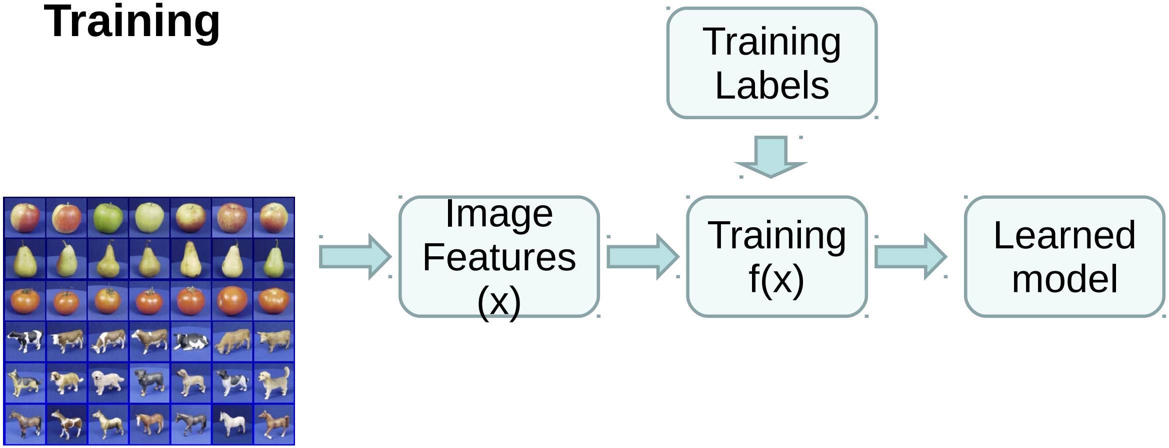
- Given a training set of labeled examples $(x1, y1), , (xN, yN)$
- Extract the features from the training set
- Apply a prediction function to the feature representation of the image to get the desired output:

$$y = f(x) \tag{1}$$

- Estimate the prediction function f by minimizing the prediction error on the training set
- Apply f to a never before seen test example x and output the predicted value $y = f(x)$

Machine Learning framework

Training



Testing

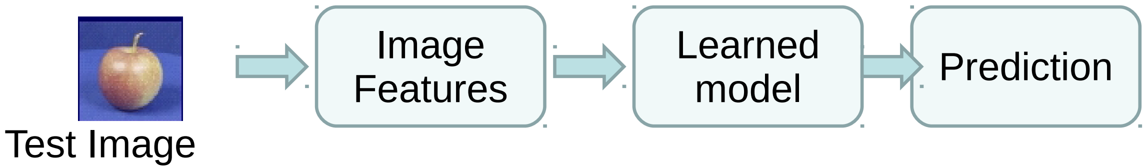
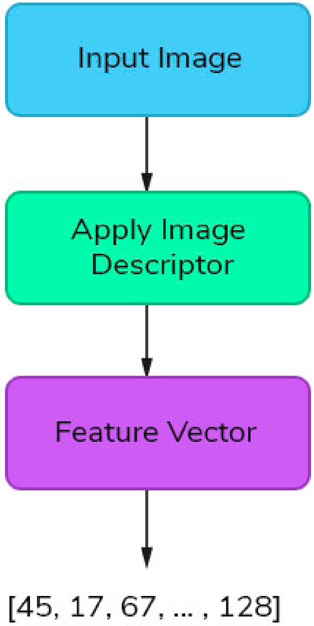


Image Classification- Sample

- Image Classification: The ability of a machine learning model to classify or label an image into its respective class with the help of learned features from hundreds of images.



Feature Descriptors

- Features are information that are extracted from an image.
- These are real-valued numbers (integers, float or binary).
- General type of image features
  - Global Feature Descriptors
  - Local Feature Descriptors:

Global Feature Descriptors

- Feature descriptors that quantifies image globally.
- Takes entire image for processing
- Image is represented by one multiple dimensional feature vector
- Examples of Global feature Descriptors
  - Color - Color Channel Statistics (Mean, Standard Deviation) and Color Histogram
  - Shape - Hu Moments, Zernike Moments
  - Texture - Haralick Texture, Local Binary Patterns (LBP)

Local Feature Descriptors

- Quantifies local regions of an image.
- Interest points are determined in the entire image and image patches/regions considered for analysis.
- The interest points are invariant to view point and illumination changes
- Examples of Local feature Descriptors
    - SIFT (Scale Invariant Feature Transform)
    - SURF (Speeded Up Robust Features)
    - ORB (Oriented Fast and Rotated BRIEF)
    - BRIEF (Binary Robust Independed Elementary Features)

Image Classification:Iris

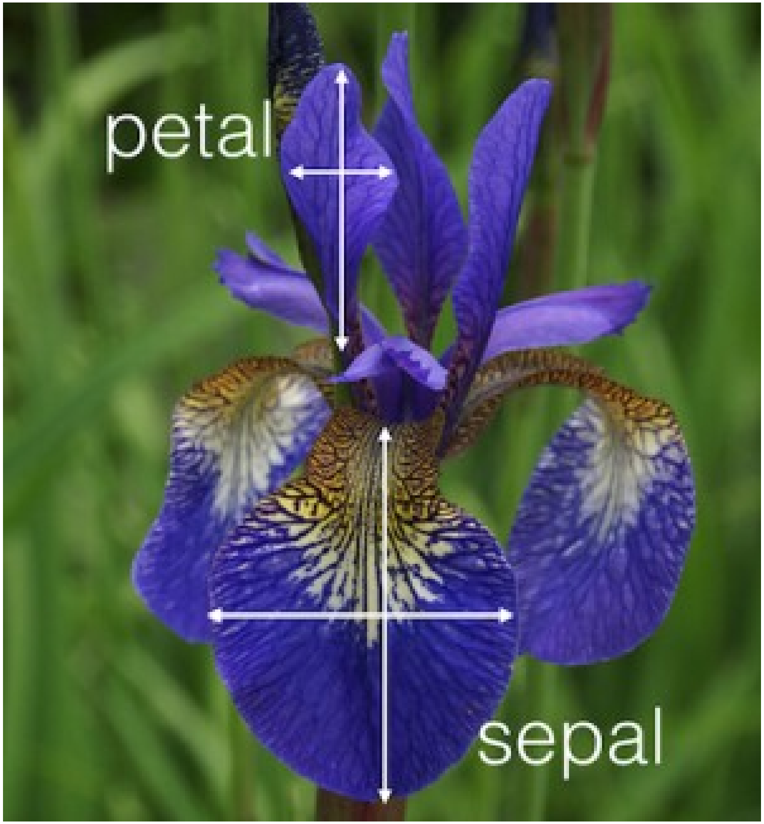


Image Classification:Iris

The iris dataset contains the following data:

- Training: 50 samples of 3 different species of iris (150 samples total)
- Features: sepal length, sepal width, petal length, petal width
- Labels: "0": setosa

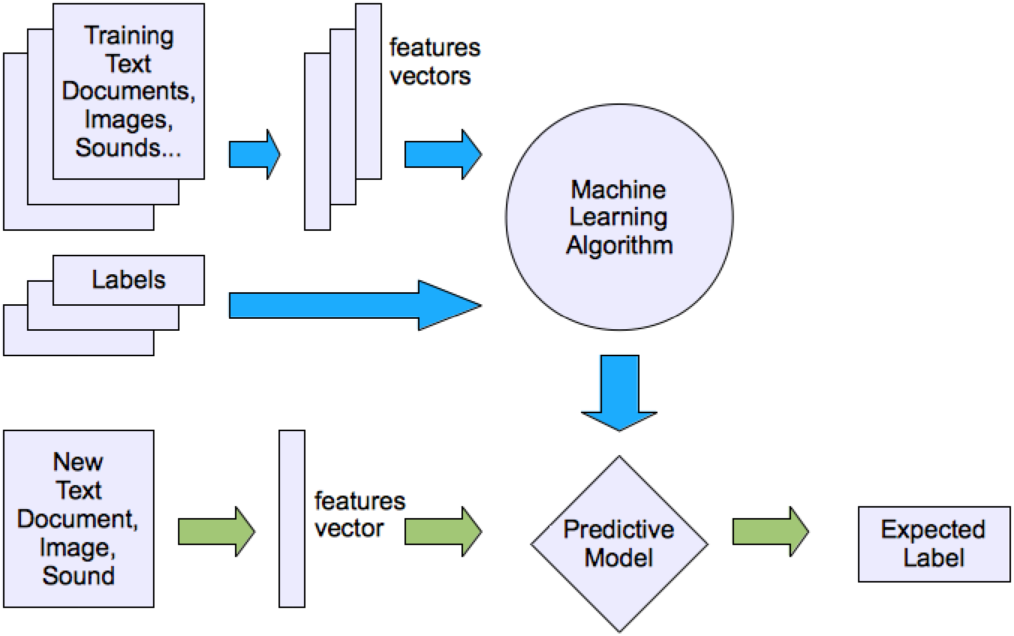
  "1": versicolor

  "2": virginica
- Feature vectors: [[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]....]

Types of Training

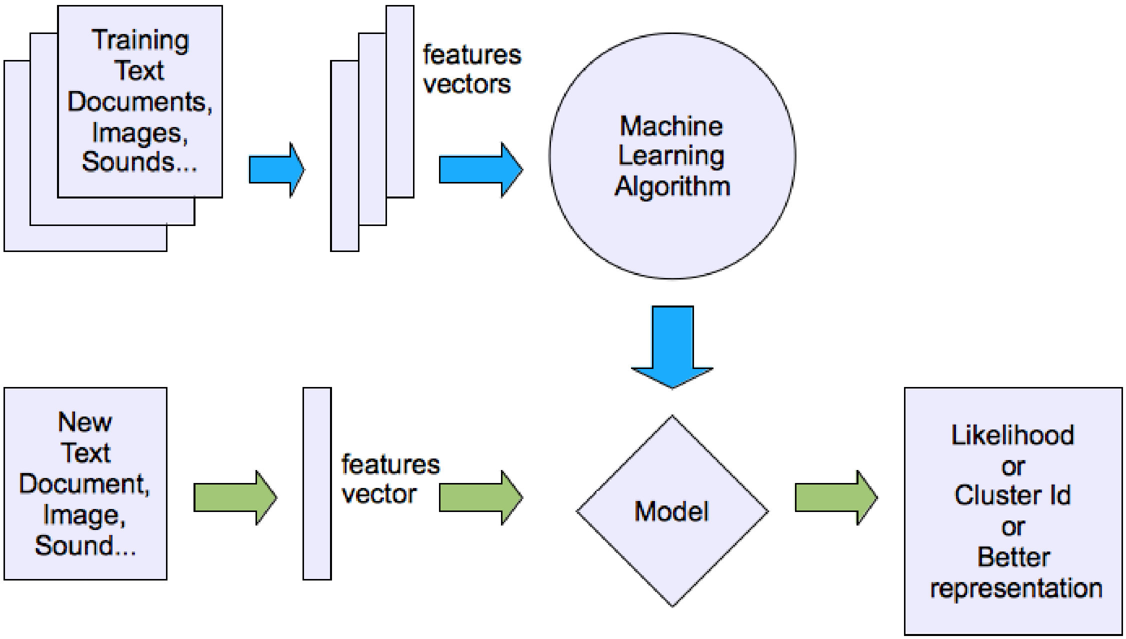
- Supervised learning: uses a series of labelled examples with direct feedback
- Unsupervised/clustering learning: no feedback
- Semisupervised
- Reinforcement learning: indirect feedback, after many examples

Supervised learning



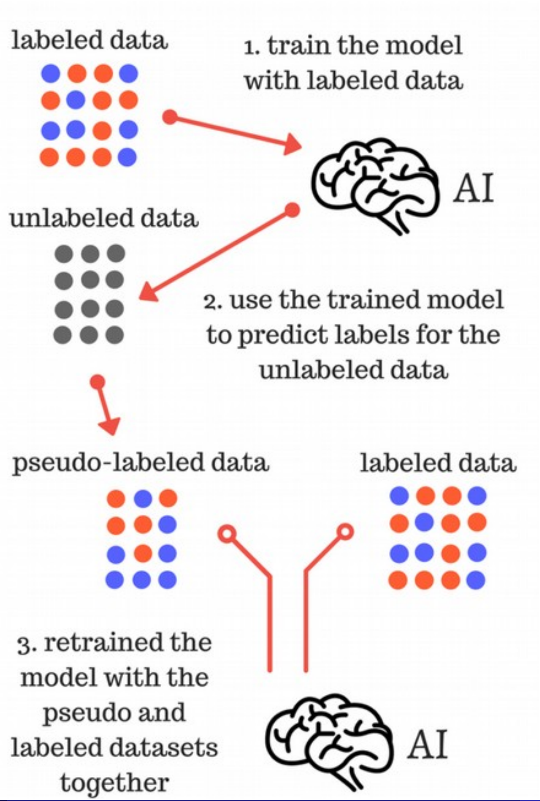
Algorithms: Nearest Neighbor, Naive Bayes, Decision Trees, Linear Regression, Support Vector Machines (SVM), Neural Networks

UnSupervised learning

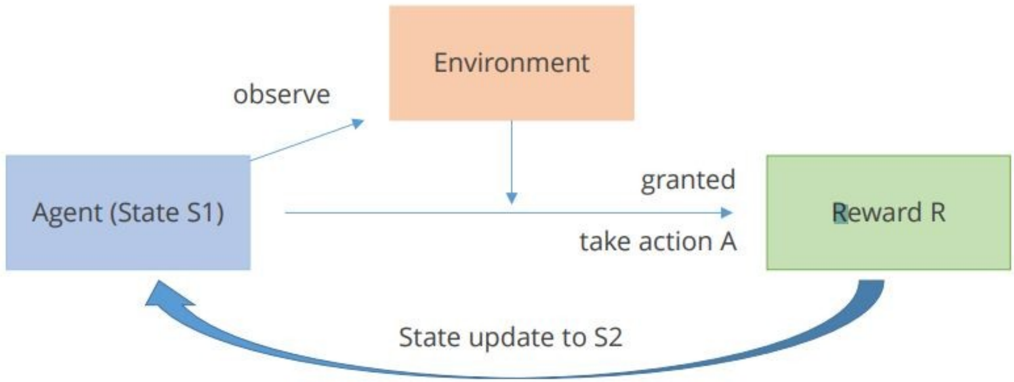


Algorithms: k-means clustering, Association Rules

Semisupervised Learning



Reinforcement Learning



Algorithms: **Q-Learning, Temporal Difference (TD), Deep Adversarial Networks**

Keras

- Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano.
- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.
- Keras is compatible with: Python 2.7-3.5.

Model

- The core data structure of Keras is a model, a way to organize layers.
- The simplest type of model is the Sequential model, a linear stack of layers.
- For more complex architectures, use the Keras functional API, which allows to build arbitrary graphs of layers.

Keras Sequential Model

- Create a Sequential model by passing a list of layer instances
- Specify the input shape.
- Compilation
- Training
- Prediction / Evaluation

Keras Sequential Model

Create a Sequential model by passing a list of layer instances to the constructor:

from keras.models import Sequential
from keras.layers import Dense, Activation

model = **Sequential**([Dense(32, input_shape=(784,)),
Activation('relu'),
Dense(10),
Activation('softmax'),])

Or simply add layers via the **.add()** method:
model = Sequential()
model.**add**(Dense(32, input_dim=784))
model.**add**(Activation('relu'))

Layers in Keras

- Core layers
- Convolutional layers
- Locally connected layers
- Pooling layers
- Recurrent layers
- Embedding layers
- Merge layers
- Normalization layers

Core Layers in Keras

- Dense
- Activation
- Dropout
- Flatten
- Reshape
- Permute
- RepeatVector

Layer - Dense

It is regular densly connected Neural Network layer

keras.layers.core.Dense(units, **activation**=None,
use_bias=True, **kernel_initializer**='glorot_uniform',
bias_initializer='zeros',
kernel_regularizer=None,**bias_regularizer**=None,
activity_regularizer=None, kernel_constraint=None,
bias_constraint=None)

model = Sequential()
model.add(Dense(32, input_shape=(16,)))

Activation

Applies an activation function to an output.

Activations can either be used through an Activation layer, or through the activation argument supported by all forward layers:

from keras.layers import Activation, Dense
model.add(Dense(64))
model.add(Activation('tanh'))

Or
model.add(Dense(64, activation='tanh'))

Activation

- elu(x, alpha=1.0)
- selu(x)
- softplus(x)
- relu(x, alpha=0.0, max_value=None)
- tanh(x)
- sigmoid(x)
- softmax(x, axis=1)

Initializers

Initializations define the way to set the initial random weights of Keras layers.

```
model.add(Dense(64,
 kernel_initializer='random_uniform',
 bias_initializer='zeros'))
```

Regularizers

Regularizers allow to apply penalties on layer parameters or layer activity during optimization.

The penalties are applied on a per-layer basis.
These layers expose 3 keyword arguments:

- kernel_regularizer
- bias_regularizer
- activity_regularizer

Compilation

Before training a model, configure the learning process, which is done via the compile method.

compile(optimizer, loss, metrics=None)

Three arguments:

- An optimizer : a str id of an existing optimizer (rmsprop or adagrad), or an instance of the Optimizer class.
- A loss function : This is the objective that the model will try to minimize. It can be the str id of an existing loss function (categorical_ crossentropy or mse), or it can be an objective function.
- A list of metrics : list of metrics to be evaluated by the model during training and testing. Ex: metrics=['accuracy']

Compilation

```
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Instance an optimizer:

```
sgd = optimizers.SGD(lr=0.01, decay=1e-6,
                     momentum=0.9, nesterov=True)
model.compile(loss='mean_squared_error',
              optimizer=sgd)
```

Loss Function

A loss function is a objective function, or optimization score function.
Loss functions:

- mean_squared_error(y_true, y_pred)
- categorical_crossentropy(y_true, y_pred)
- binary_crossentropy(y_true, y_pred)
- kullback_leibler_divergence(y_true, y_pred)
- cosine_proximity(y_true, y_pred)

Fit

Trains the model for a fixed number of epochs.

```
fit(x, y, batch_size=32, epochs=10, verbose=1,
callbacks=None, validation_split=0.0,
validation_data=None, shuffle=True,
class_weight=None, sample_weight=None,
initial_epoch=0)
```

Fit

- x: input data, as a Numpy array or list of Numpy arrays.
- y: labels, as a Numpy array.
- batch_size: integer. Number of samples per gradient update.
- epochs: integer, the number of epochs to train the model.
- verbose: 0 for no logging to stdout, 1 for progress bar logging, 2 for one log line per epoch.
- Callback: to get a view on internal states and statistics of the model during training.

Evaluate

Returns the loss value & metrics values for the model in test mode.

evaluate(object, x, y, batch_size = NULL, verbose = 1, sample_weight = NULL, steps = NULL)

- object :Model object to evaluate
- x : Numpy array of test data
- y : array of labls
- batch_size : Number of samples per gradient update.
- verbose : Verbosity mode (0 = silent, 1 = verbose, 2 = one log line per epoch).
- sample_weight : Optional array of the same length as x
- steps : Total number of steps (batches of samples)

Datagenerator

```
train_datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.1,
    height_shift_range=0.1,
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')
```

References

- https://keras.io/

Thank You