Application of Machine Learning to Radar Automatic Target Recognition

Marc Pitarys
Data Analytics Student
11 AUG 2018

Presentation Overview

› Technical Background
› Data Discussion
› Machine Learning Approaches
› Observations/Results
› Summary

Project Description

- Objective: Use machine learning to automatically identify targets in Synthetic Aperture Radar (SAR) Imagery
- Approach:
  - Understand the technical domain
  - Study the problem
  - Find, Format, and Transform Data
  - Choose a machine learning approach
  - Program software and run the model
  - Analyze results/Fine tune the model/Evaluate Accuracy

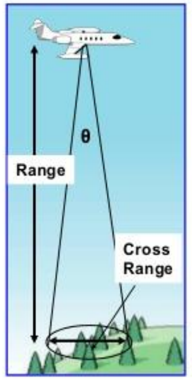
Automatically Identify Targets in Radar Imagery

This Data Science Project Relied Upon Knowledege in Three Areas

- Radar
  - Functionality
  - Signal Processing
  - Target Data (Image Formats)
- Programming
  - Python
  - Programming Libraries
  - Tools
- Mathematics
  - Linear Algebra
  - Statistics
  - Machine Learning

SAR Background

Why Synthetic Aperture Radar (SAR)?



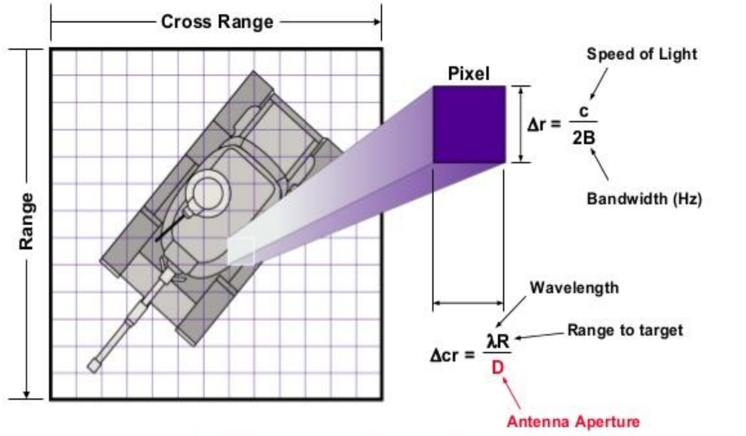
- Radar provides excellent range information
  - Can resolve in range down to inches
  - Not weather/cloud limited as visible and infrared sensors
- Good image resolution requires commensurate cross-range resolution
- Problem: The radar beam is far too wide and not matched to range resolution for good imaging of targets

Radar Parameters

Range = 100 km   Beamwidth = 0.2°   Bandwidth ≈ 500 MHz

Cross Range Resolution = R θ = 350m   Range Resolution = c/2 B ≈ 0.3 m

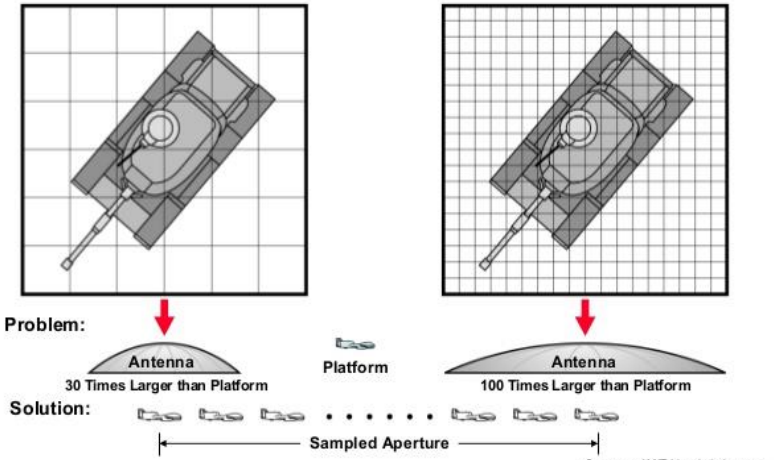
Radar Image



Imaging requires a large antenna

Courtesy of MIT Lincoln Laboratory
Used with Permission

Synthetic Aperture Radar (SAR)



Courtesy of MIT Lincoln Laboratory
Used with Permission

Cross-Range Resolution with SAR

View SAR as a Phased Array Antenna

- **Passive Array Resolution**

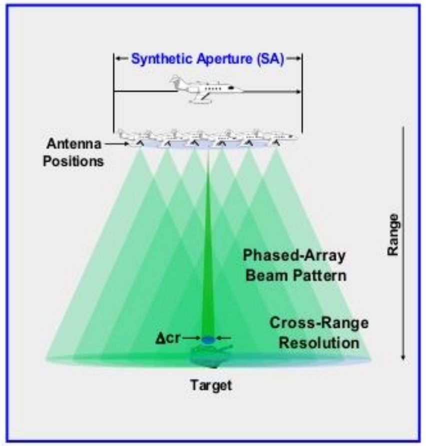
$$\Delta cr = \frac{\lambda R}{D}$$

← Range; Antenna Aperture

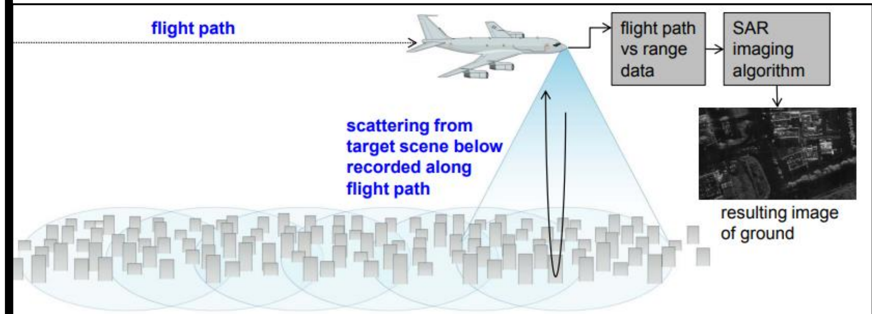
- **SAR Resolution**

$$\Delta cr = \frac{\lambda R}{2 \times SA}$$

Separated radar positions provide twice the phase shift



SAR Example



- Small antenna on aircraft illuminates large swaths of ground
- Range profiles recorded along flight path
- SAR algorithm processes data into image of ground [2]
  - *thereby synthesizing an aperture the length of the aircraft flight path*
  - *narrow beamwidth, high resolution and gain*

500 m × 830 m   * Lincoln Multi-Mission ISR Testbed

Programming Knowledge

Programming Language

Array Manipulation

Machine Learning

Data Frames

Visualization

imageio - Python library for reading and writing image data

PIL: Python Image Library

Image Processing

Program Development

Data Source

Moving and Stationary Target Acquisition and Recognition (MSTAR) Database

- Publicly Released collection of radar imagery of ten military vehicles
- Standard Common Dataset for Researchers
- Downloadable

MSTAR PUBLIC TARGETS

The following data set was collected in September of 1995 at the Redstone Arsenal, Huntsville, AL by the Sandia National Laboratory (SNL) SAR sensor platform. The collection was jointly sponsored by DARPA and Air Force Research Laboratory as part of the Moving and Stationary Target Acquisition and Recognition (MSTAR) program. SNL used an X-band SAR sensor in one foot resolution spotlight mode. Strip map mode was used to collect the clutter data.

Targets (# of)	Target Description	Amount
T-72 (3)	T-72 Tank	3 replicate targets: each collected at 15 & 17 degree dep. angles and full aspect coverage
BMP2 (3)	Infantry Fighting Vehicle	3 replicate targets: each collected at 15 & 17 degree dep. angles and full aspect coverage
BTR-70 (1)	Armored Personnel Carrier	1 target: collected at 15 & 17 degree dep. angles and full aspect coverage
Slicy (1)	Multiple simple geometric shaped static target	CAD Model November '96 Imagery: TBD in Jan '97

The "Slicy" target is a precisely designed and machined engineering test target containing standard radar reflector primitive shapes such as flat plates, dihedrals, trihedrals, and top hats. The purpose of this target is to allow Image Understanding developers the ability to validate the functionality of their algorithm with a simple known target.

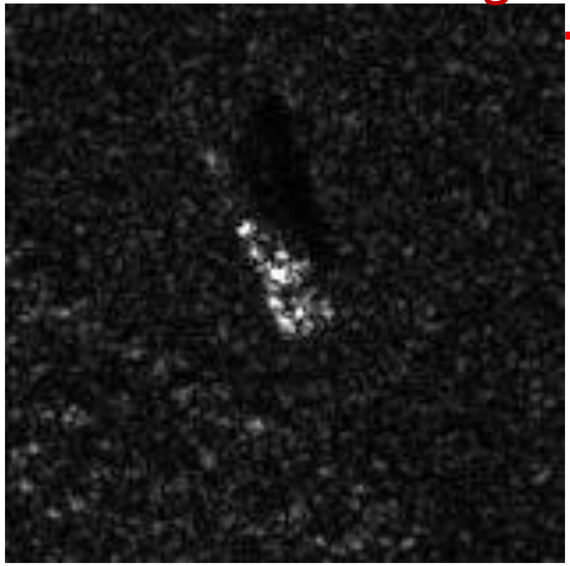
MSTAR/IU MIXED TARGETS

The following data set was collected as part of the MSTAR Data Collection #1, Scene 1 and as part of the MSTAR Data Collection #2, Scenes #1, #2, and #3. Sandia National Laboratory used an X-band STARLOS sensor at 1 foot resolution in Spotlight mode to collect the data at 15, 17, 30, and 45 degree depression angles. The image chips and JPEG files include 2S1, BDRM-2, BTR-60, D7, T62, ZIL-131, ZSU-23/4, and SLICY.

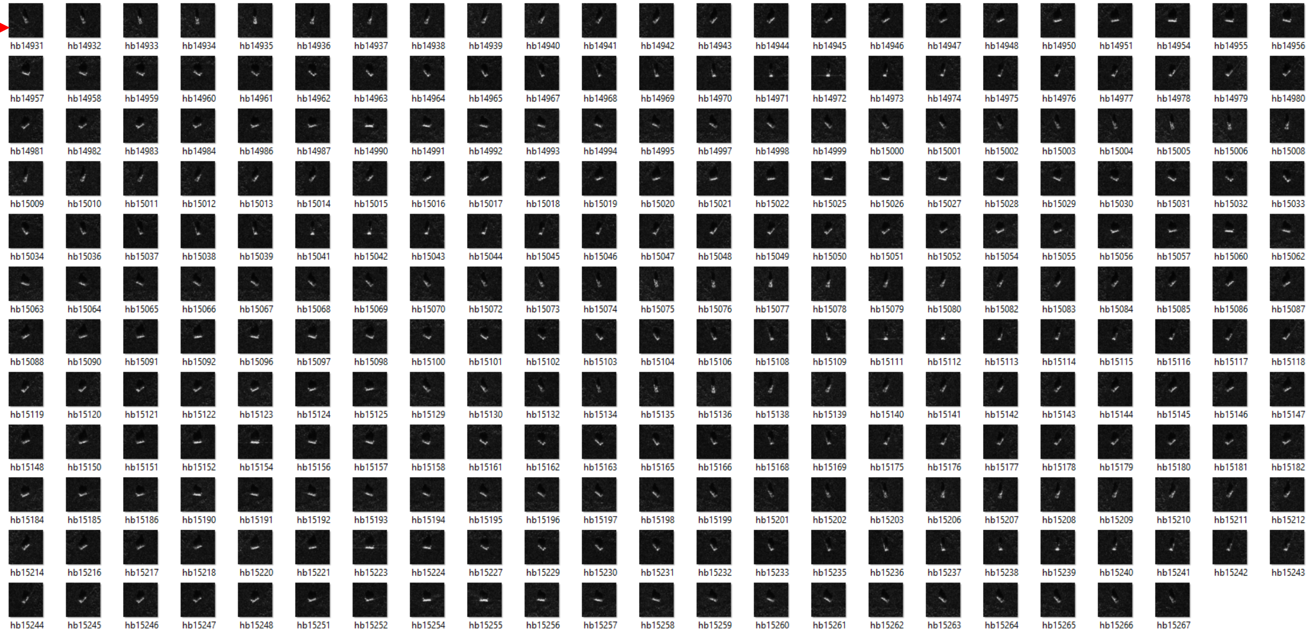
Imagery Test Data Example

The 2S1 Gvozdika (Russian: 2c1 "Carnation") is a Soviet self-propelled artillery vehicle mounting a 122 mm howitzer. It's fully amphibious and when afloat it's propelled by its tracks. A variety of track widths are available to allow the 2S1 to operate in snow or swamp conditions. It is NBC protected and has infra-red night-vision capability.

Processed Radar Image

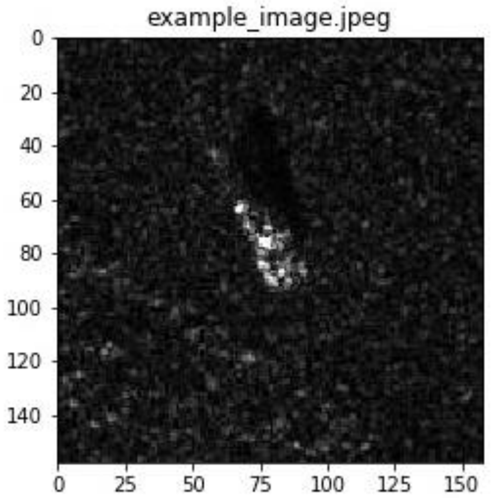


2S1 Testing Data



Radar File Meta-Data

Detailed Ground Truth and sensor information



158 pixels x 0.3m = 47.4m

PhoenixHeaderLength= 01597
PhoenixSigSize= 00201309
PhoenixSigNum= 0001
PhoenixHeaderCallingSequence=
HeaderVersionNumber= 8CM
native_header_length= 0
Filename= hb14931.0000
ParentScene= hb14931
NumberOfColumns= 158
NumberOfRows= 158
TargetType= 2s1_gun
TargetSerNum= b01
TargetAz= 338.224854
TargetRoll= 358.749634
TargetPitch= 359.614655
TargetYaw= 357.094086
DesiredDepression= 15
DesiredGroundPlaneSquint= -90
DesiredSlantPlaneSquint= -90
DesiredRange= 5000
DesiredAimpointElevation= 39
MeasuredDepression= 15.042969
MeasuredGroundPlaneSquint= -91.617958
MeasuredSlantPlaneSquint= -91.562500
MeasuredRange= 4979
MeasuredAimpointElevation= 37.877998
MeasuredAircraftHeading= -178.375000
MeasuredAircraftAltitude= 1330.281006

RadarMode= mode 5 - spot light
SensorCalibrationFactor= 42.995998
RadarPosition= bottom
Range3dBWidth= 0.307800
CrossRange3dBWidth= 0.315300
SceneCenterReferenceLine= 180
X_Velocity= 42.444336
DataCollectors= Sandia National Lab
CollectionName= MSTAR Collection 2 Scene 1
SensorName= Twin Otter
Classification= UNCLASSIFIED
MultiplicativeNoise= -10 dB
AdditiveNoise= -32 to -34 dB
CenterFrequency= 9.599000 GHz
CrossRangeWeighting= -35dB_Taylor
RangeWeighting= -35dB_Taylor
DynamicRange= 64 dB
Bandwidth= 0.591 GHz
RangeResolution= 0.304700
CrossRangeResolution= 0.304700
RangePixelSpacing= 0.202148
CrossRangePixelSpacing= 0.203125
AverageImageCalFactor= 1.253507
Polarization= HH
TargetSeasonalCover= only growing vegitation
TargetWaterContent= dry

Sandia SAR on Twin Otter

Meta Data from the Radar File

.3m pixel

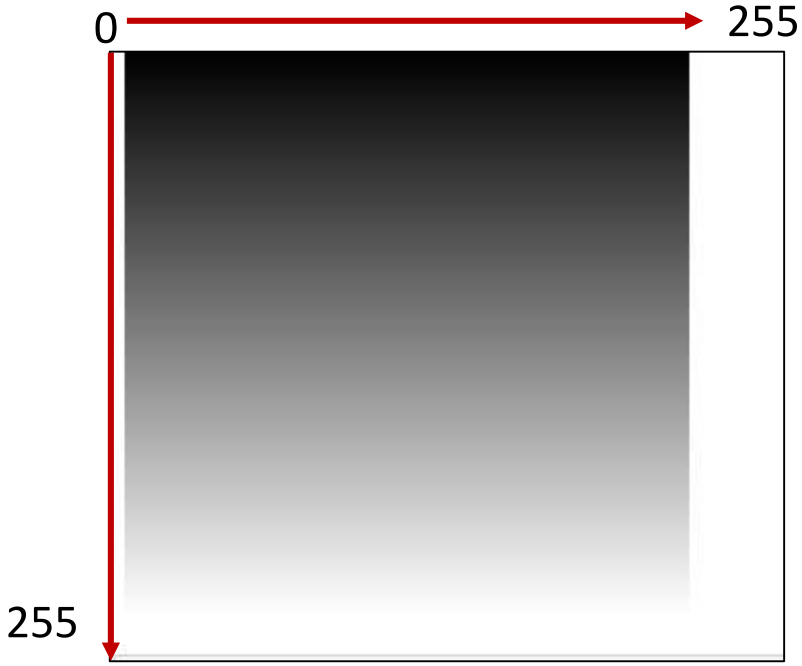
Images are arrays: Use NumPy

```python
temp=np.zeros((1,256), dtype=np.uint8)
for i in range(1,256):
    grays = np.full((1,256), i,dtype =np.uint8)
    st = np.concatenate((temp,grays))
    temp = st
```

```python
st
```

```
array([[  0,   0,   0, ...,   0,   0,   0],
       [  1,   1,   1, ...,   1,   1,   1],
       [  2,   2,   2, ...,   2,   2,   2],
       ...,
       [253, 253, 253, ..., 253, 253, 253],
       [254, 254, 254, ..., 254, 254, 254],
       [255, 255, 255, ..., 255, 255, 255]], dtype=uint8)
```

```python
Image.fromarray(st)
```




```python
st.shape
```

(x, y)

```
(256, 256)
```

```python
type(example_image)
```

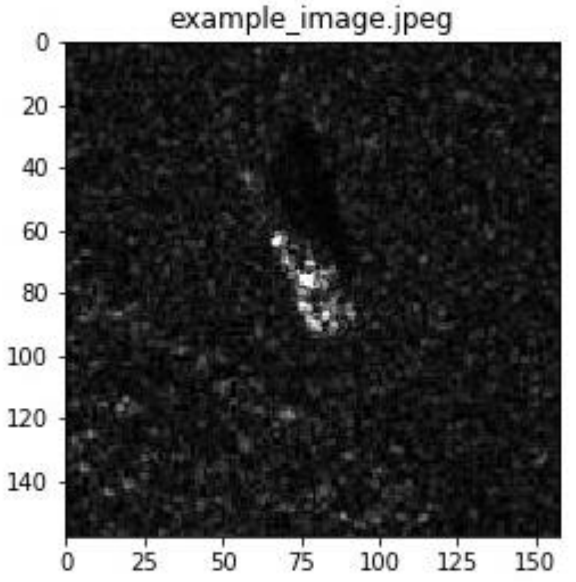
```
imageio.core.util.Image
```

```python
example_image
```

```
Image([[13, 44, 33, ..., 24, 10,  9],
       [ 9, 22, 19, ..., 16,  5,  5],
       [41, 25, 17, ..., 18, 11,  8],
       ...,
       [ 7,  9, 21, ..., 14,  8, 10],
       [30, 15, 14, ..., 15, 11, 13],
       [37, 17,  9, ..., 23, 25, 29]], dtype=uint8)
```

```python
example_image.shape
```

```
(158, 158)
```

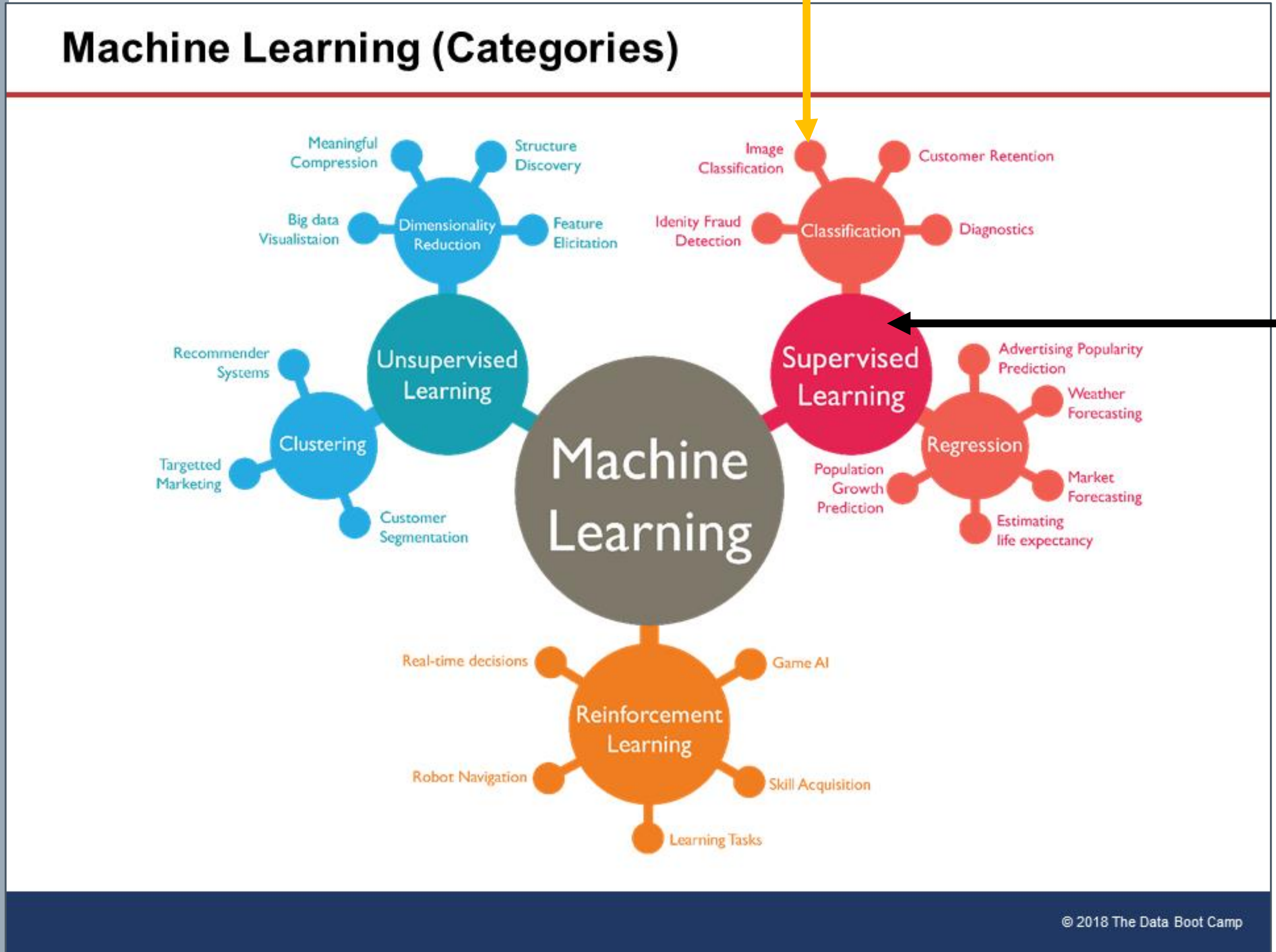


Pixel Intensity is an integer on a Grey scale of 0 to 255

- 0 : Black
- 255: White
- Radar Returns from flat surfaces: black
- Radar Returns from uneven surfaces: white

Machine Learning Category

Use Supervised Learning to Solve the Radar Image Classification Problem



Supervised Learning Machine Learning

- Train the machine with labelled data tagged with the correct answer
- Test the machine by operating on labeled data to produce a correct output

Train and Test Set Sizes

In [9]:
```
train_source_df.tail()
```
Out[9]:

	Class	Directory	Height	Width
2741	ZSU_23_4	MSTAR-10/train/ZSU_23_4/	158	158
2742	ZSU_23_4	MSTAR-10/train/ZSU_23_4/	158	158
2743	ZSU_23_4	MSTAR-10/train/ZSU_23_4/	158	158
2744	ZSU_23_4	MSTAR-10/train/ZSU_23_4/	158	158
2745	ZSU_23_4	MSTAR-10/train/ZSU_23_4/	158	158

In [10]:
```
train_source_df.shape
```
Out[10]: (2746, 4)

2746 Training Files each 158 x 158 pixels

```
Getting Data
Subdirectory: MSTAR-10/train/2S1/
Class: 2S1 Number of Files: 299

Subdirectory: MSTAR-10/train/BMP2/
Class: BMP2 Number of Files: 233

Subdirectory: MSTAR-10/train/BRDM_2/
Class: BRDM_2 Number of Files: 298

Subdirectory: MSTAR-10/train/BTR60/
Class: BTR60 Number of Files: 256

Subdirectory: MSTAR-10/train/BTR70/
Class: BTR70 Number of Files: 233

Subdirectory: MSTAR-10/train/D7/
Class: D7 Number of Files: 299

Subdirectory: MSTAR-10/train/T62/
Class: T62 Number of Files: 298

Subdirectory: MSTAR-10/train/T72/
Class: T72 Number of Files: 232

Subdirectory: MSTAR-10/train/ZIL131/
Class: ZIL131 Number of Files: 299

Subdirectory: MSTAR-10/train/ZSU_23_4/
Class: ZSU_23_4 Number of Files: 299
```

In [5]:
```
test_source_df.head()
```
Out[5]:

	Class	Directory	Height	Width
0	2S1	MSTAR-10/test/2S1/	158	158
1	2S1	MSTAR-10/test/2S1/	158	158
2	2S1	MSTAR-10/test/2S1/	158	158
3	2S1	MSTAR-10/test/2S1/	158	158
4	2S1	MSTAR-10/test/2S1/	158	158

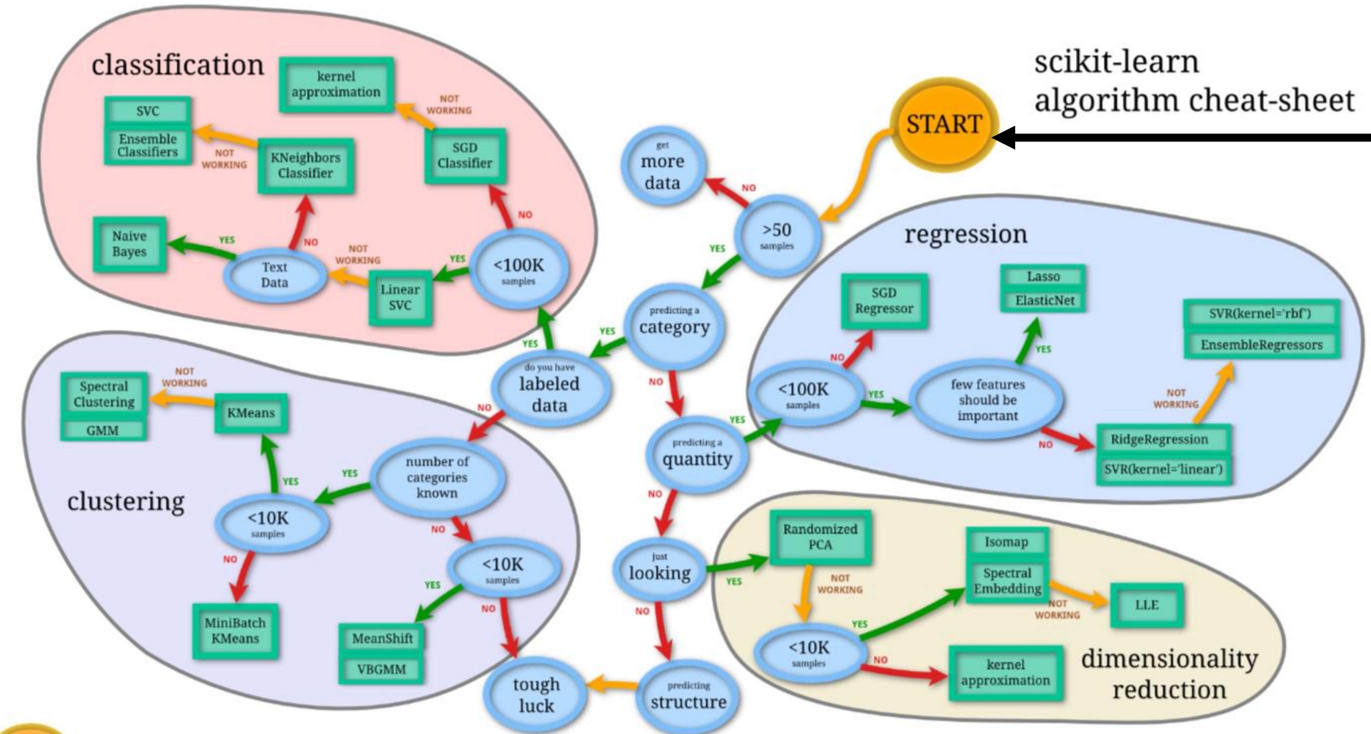
In [7]:
```
test_source_df.shape
```
Out[7]: (2425, 4)

2425 Test Files each 158 x 158 pixels

```
Getting Data
Subdirectory: MSTAR-10/test/2S1/
Class: 2S1 Number of Files: 274

Subdirectory: MSTAR-10/test/BMP2/
Class: BMP2 Number of Files: 195

Subdirectory: MSTAR-10/test/BRDM_2/
Class: BRDM_2 Number of Files: 274

Subdirectory: MSTAR-10/test/BTR60/
Class: BTR60 Number of Files: 195

Subdirectory: MSTAR-10/test/BTR70/
Class: BTR70 Number of Files: 196

Subdirectory: MSTAR-10/test/D7/
Class: D7 Number of Files: 274

Subdirectory: MSTAR-10/test/T62/
Class: T62 Number of Files: 273

Subdirectory: MSTAR-10/test/T72/
Class: T72 Number of Files: 196

Subdirectory: MSTAR-10/test/ZIL131/
Class: ZIL131 Number of Files: 274

Subdirectory: MSTAR-10/test/ZSU_23_4/
Class: ZSU_23_4 Number of Files: 274
```

Selection of the Supervised Learning Algorithm: Kneighbors

Use Kneighbors to Solve the
Radar Image Classification Problem



Input Data
- 2746 Training Samples:
- 2425 Test Samples:
- Data labelled with the correct class

Output:
- Category: One of 10 classes
- Integers that Range from 0 to 9

Back

KNeighborsClassifier

- How it works: Find a predefined number of training samples closest in distance to the new point, and predict the label from these points. It remembers all of its training data
- Adjustments:
  - Number of training samples
  - Distance metric measure
- *Successful in handwriting and satellite image scenes according to scikit-learn.org*
- Features: An integer on a grey scale of 0 to 255 for each pixel in an image equivalent to the energy of the radar return
- Classes: Ten military vehicles

Ten Classes for the Identifcation

["2S1", "BMP2", "BRDM_2", "BTR60", "BTR70", "D7", "T62", "T72", "ZIL131", "ZSU_23_4"]]

2S1

BMP2

BRDM_2

BTR60

BTR70

D7

T62

T72

ZIL 131

ZSU_23_4

Format Data for Machine Learning

Training Files

Testing Files

Read Files
Read Size Images
Crop Images
Store in a NumPy Arrays

Reshape the Arrays

```
np_X_test.shape #Start with this array for the test input
```

```
(2425, 96, 96)
```

```
np_X_test = np.reshape(np_X_test, [np_X_test.shape[0], np_X_test.shape[1] * np_X_test.shape[2]])  #Reshape test data
```

```
np_X_test.shape
```

```
(2425, 9216)
```

```
np_X_train
```

```
array([[[21, 15, 18, ..., 45, 29, 25],
        [12, 23, 23, ..., 45, 41, 23],
        [ 9, 19, 21, ..., 25, 26, 17],
        ...,
        [20, 34, 33, ..., 11, 13, 13],
        [14, 13, 18, ..., 13, 18, 29],
        [15,  6, 18, ..., 10, 16, 28]],

       [[ 9, 14, 27, ..., 19,  9, 15],
        [20, 23, 20, ..., 14,  7, 27],
        [20, 33, 23, ..., 20, 14, 23],
        ...,
        [41, 28,  9, ..., 22, 22,  7],
        [25, 16, 21, ...,  9, 12, 16],
        [26, 26, 33, ..., 11, 15, 15]],

       [[22, 25, 31, ..., 11, 22, 21],
        [14, 26, 34, ..., 11, 29, 27],
        [31, 30, 17, ..., 11, 12, 14],
        ...,
        [32, 20, 13, ..., 11,  8, 28],
        [29, 17, 25, ..., 15, 24, 34],
        [25, 25, 29, ..., 24, 23, 22]],

       ...,
```

```
np_y_train
```

```
array([0, 0, 0, ..., 9, 9, 9])
```

```
np_y_train.shape
```

```
(2425,)
```

```
np_X_train.shape
```

```
(2425, 96, 96)
```

```
np_X_test.shape
```

```
(2425, 96, 96)
```

```
np_y_test
```

```
array([0, 0, 0, ..., 9, 9, 9])
```

```
np_y_test.shape
```

```
(2425,)
```

```
np_y_test[:, np.newaxis]  #Slice up y into a array of 1-element arrays
```

```
array([[0],
       [0],
       [0],
       ...,
       [9],
       [9],
       [9]])
```

```
y_test = np_y_test[:, np.newaxis]
```

```
y_test
```

```
array([[0],
       [0],
       [0],
       ...,
       [9],
       [9],
       [9]])
```

```
y_test.shape
```

```
(2425, 1)
```

Randomize and Scale Feature Data to be on the same scale

Reshaped Arrays

```
data = np.hstack([np_X_test, y_test])          #Test Data
```

```
data
```

```
array([[21, 15, 18, ..., 16, 28,  0],
       [ 9, 14, 27, ..., 15, 15,  0],
       [22, 25, 31, ..., 23, 22,  0],
       ...,
       [ 8,  7,  4, ..., 35, 23,  9],
       [ 8,  8,  8, ...,  6, 14,  9],
       [ 9, 11, 10, ..., 11, 11,  9]])
```

Horizontally Stack Arrays For Shuffling

```
data.shape  #Notice that data has 2425 arrays that now contain 9217 elements
```

```
(2425, 9217)
```

Randomly Shuffle Arrays

```
#Randomly shuffle the data before training the neural network
#Use numpy.random.shuffle
#Multi-dimensional arrays are only shuffled along the first axis
#Modify a sequence in-place by shuffling its contents.

#Shuffle both training and test data
```

```
data[0:5,]  #Before
```

```
array([[21, 15, 18, ..., 16, 28,  0],
       [ 9, 14, 27, ..., 15, 15,  0],
       [22, 25, 31, ..., 23, 22,  0],
       [32, 34, 28, ..., 16, 17,  0],
       [18, 14, 10, ..., 57, 39,  0]])
```

```
np.random.shuffle(data)  #Shuffle
```

```
data[0:5,]  #  After, Look the data array has been shuffled
```

```
array([[15, 37, 44, ..., 20, 13,  5],
       [18, 12, 15, ..., 23, 18,  8],
       [18, 15, 25, ..., 31, 38,  0],
       [14, 16, 21, ..., 15, 25,  4],
       [40, 81, 42, ..., 34, 11,  4]])
```

```
X_test_shuffled = data[:, :-1] # Get everything before the least element in the array.
y_test_shuffled = data[:, -1]  #Get only the last element of the data array.  This is the class output
```

```
X_test_shuffled
```

```
array([[15, 37, 44, ..., 20, 20, 13],
       [18, 12, 15, ..., 13, 23, 18],
       [18, 15, 25, ..., 40, 31, 38],
       ...,
       [23, 15, 15, ..., 11, 19, 20],
       [ 9, 17, 14, ..., 11, 14, 19],
       [38, 30, 29, ..., 23, 19, 24]])
```

```
y_test_shuffled
```

```
array([5, 8, 0, ..., 8, 1, 4])
```

Scale Training and Test Data to fall between 0 and 1.
Divide by 256 or 2^8

```
X_test_shuffled_scaled = X_test_shuffled/255
```

```
X_test_shuffled_scaled
```

```
array([[0.05882353, 0.14509804, 0.17254902, ..., 0.07843137, 0.07843137,
        0.05098039],
       [0.07058824, 0.04705882, 0.05882353, ..., 0.05098039, 0.09019608,
        0.07058824],
       [0.07058824, 0.05882353, 0.09803922, ..., 0.15686275, 0.12156863,
        0.14901961],
       ...,
       [0.09019608, 0.05882353, 0.05882353, ..., 0.04313725, 0.0745098 ,
        0.07843137],
       [0.03529412, 0.06666667, 0.05490196, ..., 0.04313725, 0.05490196,
        0.0745098 ],
       [0.14901961, 0.11764706, 0.11372549, ..., 0.09019608, 0.0745098 ,
        0.09411765]])
```

Feature Scaling Example

```
X_train_shuffled_scaled
```

```
array([[0.00392157, 0.08627451, 0.14117647, ..., 0.16862745, 0.16862745,
        0.11372549],
       [0.03921569, 0.03529412, 0.07058824, ..., 0.03529412, 0.03529412,
        0.0627451 ],
       [0.07058824, 0.12156863, 0.19215686, ..., 0.18431373, 0.23921569,
        0.24313725],
       ...,
       [0.07058824, 0.04705882, 0.09803922, ..., 0.05882353, 0.05490196,
        0.03137255],
       [0.11372549, 0.08627451, 0.03529412, ..., 0.14117647, 0.16862745,
        0.16470588],
       [0.05882353, 0.08235294, 0.03921569, ..., 0.06666667, 0.10196078,
        0.05490196]])
```

```
X_train_shuffled_scaled.shape
```

```
(2746, 9216)
```

```
X_train_shuffled_scaled.mean(axis=0)
```

```
array([0.09042886, 0.08962198, 0.08860374, ..., 0.08802822, 0.08868943,
       0.08819959])
```

```
X_train_centered = X_train_shuffled_scaled - X_train_shuffled_scaled.mean(axis=0)
```

```
#X_train_shuffled_scaled_centered
X_train_centered
```

```
array([[-0.08650729, -0.00334747,  0.05257273, ...,  0.08059923,
         0.07993802,  0.0255259 ],
       [-0.05121317, -0.05432786, -0.01801551, ..., -0.0527341 ,
        -0.05339531, -0.02545449],
       [-0.01984062,  0.03194665,  0.10355312, ...,  0.09628551,
         0.15052626,  0.15493766],
       ...,
       [-0.01984062, -0.04256316,  0.00943547, ..., -0.02920469,
        -0.03378747, -0.05682704],
       [ 0.02329663, -0.00334747, -0.05330963, ...,  0.05314825,
         0.07993802,  0.07650629],
       [-0.03160533, -0.00726904, -0.04938806, ..., -0.02136155,
         0.01327135, -0.03329763]])
```

To center the data (make it have zero mean and unit standard error),
you subtract the mean and then divide the result by the standard deviation.

$x' = (x - \mu)/\sigma$

Note we have apriori knowledge that the different attributes are all on the same scale. Accordingly, do not divide each element by the variance

Center the testing data with the mean computed from the training data

Reduce Dimensionality: Principal Component Analysis (PCA)

```
pca = PCA()
pca.fit(X_train_centered)  #First don't reduce any of the dimensions
cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.95) + 1
```

```
cumsum
```

```
array([0.15276593, 0.23165207, 0.27965282, ..., 0.99999311, 1.        ,
       1.        ])
```

Cumulative variances by array element

```
pca.explained_variance_ratio_
```

Variances of each principal component

```
array([1.52765926e-01, 7.88861425e-02, 4.80007543e-02, ...,
       7.74407398e-05, 7.71373640e-05, 7.69276289e-05])
```

```
d
```

Number of dimensions that account for 95% of the variance

```
1284
```

Reduced dimensions from 9216 (96x96) to 1284

```
pca = PCA(n_components = d)
x_pca_fit = pca.fit(X_train_centered)

X_Reduced_train_transform = x_pca_fit.transform(X_train_centered)
X_Reduced_test_transform = x_pca_fit.transform(X_centered_test )
```

Fit the training data to the reduce dimensions

Project data to the reduce dimensions

Results

```
from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors=10, weights="distance",
algorithm="auto").fit(X_Reduced_train_transform,y_train_shuffled)
```

```
classifier.score(X_Reduced_test_transform, y_test_shuffled)
```

```
0.9512017479970867
```

95%

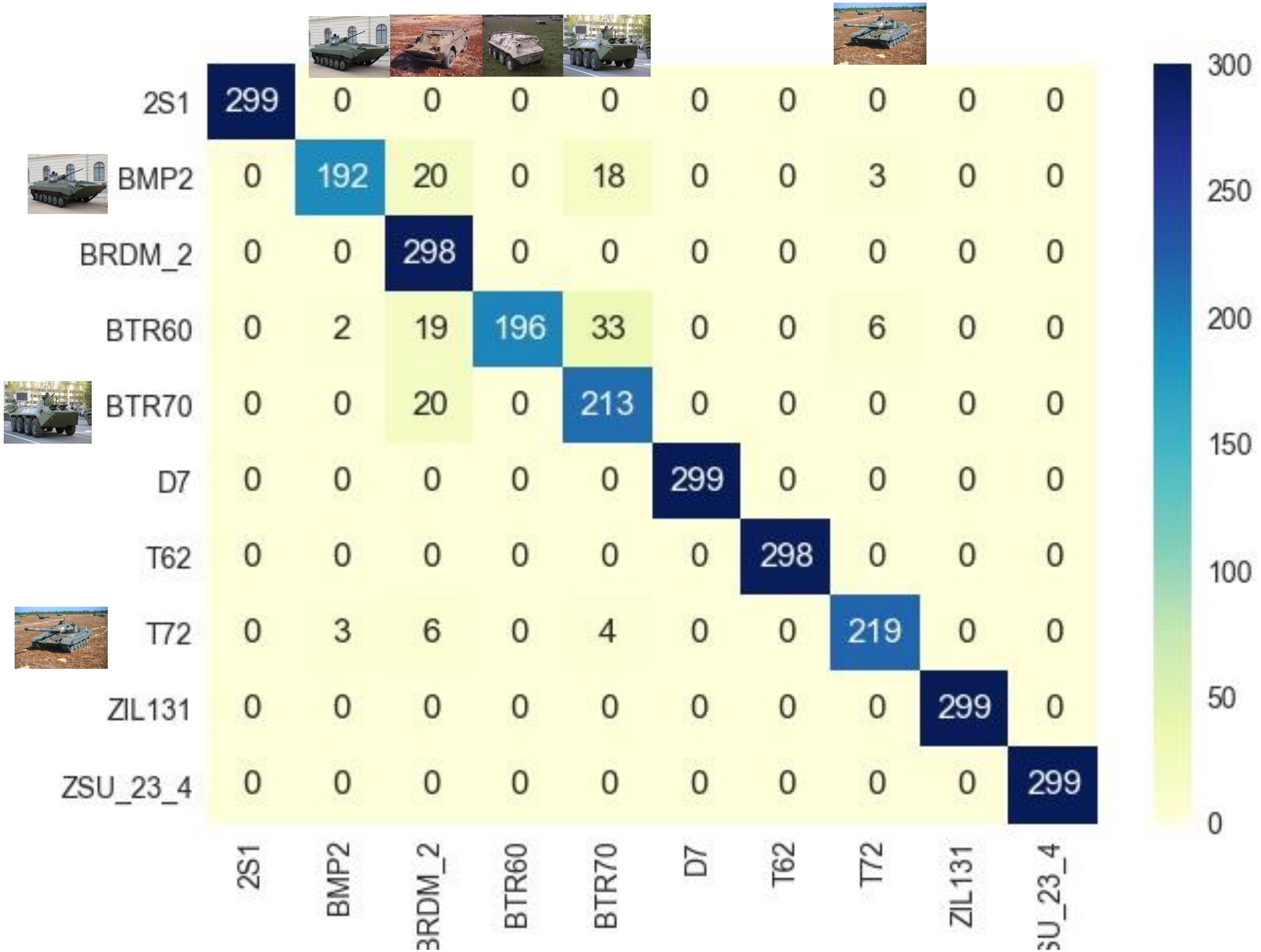
```
y_predict = classifier.predict(X_Reduced_test_transform)
y_predict
```

```
array([2, 2, 4, ..., 3, 5, 1])
```

```
y_test_shuffled
```

```
array([2, 2, 4, ..., 3, 5, 1])
```

Confusion Matrix



Summary

- SAR Automatic Target Recognition using Machine Learning is Feasible
- In the case presented, KNN is the appropriate classifier
- 95% correct identification using the test data