Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

In [ ]:
```
NAME = ""
COLLABORATORS = ""
```

Problem description

To a large degree, financial data has traditionally been numeric in format.

But in recent years, non-numeric formats like image, text and audio have been introduced.

Private companies have satellites orbiting the Earth taking photos and offering them to customers. A financial analyst might be able to extract information from these photos that could aid in the prediction of the future price of a stock

- Approximate number of customers visiting each store: count number of cars in parking lot
- Approximate activity in a factory by counting number of supplier trucks arriving and number of delivery trucks leaving
- Approximate demand for a commodity at each location: count cargo ships traveling between ports

In this assignment, we will attempt to recognize ships in satellite photos. This would be a first step toward counting.

As in any other domain: specific knowledge of the problem area will make you a better analyst. For this assignment, we will ignore domain-specific information and just try to use a labeled training set (photo plus a binary indicator for whether a ship is present/absent in the photo), assuming that the labels are perfect.

Goal:

In this notebook, you will need to create a model in `sklearn` to classify satellite photos.

- The features are images: 3 dimensional collection of pixels
    - 2 spatial dimensions
    - 1 dimension with 3 features for different parts of the color spectrum: Red, Green, Blue
- The labels are either 1 (ship is present) or 0 (ship is not present)

Learning objectives

- Learn how to implement a model to solve a Classification task

Imports modules

In [ ]:
```
## Standard imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import sklearn

import os
import math

%matplotlib inline
```

In [ ]:

```
## Load the helper module
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

# Reload all modules imported with %aimport
%reload_ext autoreload
%autoreload 1

# Import nn_helper module
import helper
%aimport helper

helper = helper.Helper()
```

API for students

We have defined some utility routines in a file `helper.py`. There is a class named `Helper` in it.

This will simplify problem solving

More importantly: it adds structure to your submission so that it may be easily graded

`helper = helper.Helper()`

- getData: Get a collection of labeled images, used as follows

  ```
  data, labels = helper.getData()
  ```

- showData: Visualize labelled images, used as follows

  ```
  helper.showData(data, labels)
  ```

- model_interpretation: Visualize the model parameters

  ```
  helper.model_interpretation(Classifier)
  ```

Get the data

The first step in our Recipe is Get the Data.

We have provided a utility method `getData` to simplify this for you

In [ ]:
```
# Get the data
data, labels = helper.getData()
n_samples, width, height, channel = data.shape

print("Data shape: ", data.shape)
print("Labels shape: ", labels.shape)
print("Label values: ", np.unique(labels))
```

Your expected outputs should be following
```
Date shape: (4000, 80, 80, 3)
Labels shape: (4000,)
Label values: [0 1]
```

We will shuffle the examples before doing anything else.

This is usually a good idea

- Many datasets are naturally arranged in a *non-random* order, e.g., examples with the sample label grouped together
- You want to make sure that, when you split the examples into training and test examples, each split has a similar distribution of examples

In [ ]:
```
# Shuffle the data first
data, labels = sklearn.utils.shuffle(data, labels, random_state=42)
```

Have a look at the data

We will not go through all steps in the Recipe, nor in depth.

But here's a peek

```
In [ ]: # Visualize the data samples
        helper.showData(data[:25], labels[:25])
```

Eliminate the color dimension

As a simplification, we will convert the image from color (RGB, with 3 "color" dimensions referred to as Red, Green and Blue) to gray scale.

```
In [ ]: print("Original shape of data: ", data.shape)

        w = (.299, .587, .114)
        data_bw = np.sum(data *w, axis=3)

        print("New shape of data: ", data_bw.shape)
```

```
In [ ]: # Visualize the data samples
        helper.showData(data_bw[:25], labels[:25], cmap="gray")
```

Have look at the data: Examine the image/label pairs

Rather than viewing the examples in random order, let's group them by label.

Perhaps we will learn something about the characteristics of images that contain ships.

We have loaded and shuffled our dataset, now we will take a look at image/label pairs.

Feel free to explore the data using your own ideas and techniques.

In [ ]:
```
# Inspect some data (images)
num_each_label = 10

for lab in np.unique(labels):
    # Fetch images with different labels
    X_lab, y_lab = data_bw[ labels == lab ], labels[ labels == lab]
    # Display images
    fig = helper.showData( X_lab[:num_each_label], [ str(label) for label in y_l
ab[:num_each_label] ], cmap="gray")
    _= fig.suptitle("Label: "+  str(lab), fontsize=14)
    _= fig.show()
    print("\n\n")
```

It appears that a photo is labeled as having a ship present only if the ship is in the **center** of the photo.

Perhaps this prevents us from double-counting.

In any event: we have learned something about the examples that may help us in building models

- Perhaps there is some feature engineering that we can perform to better enable classification

Create a test set

To train and evaluate a model, we need to split the original dataset into a training subset (in-sample) and a test subset (out of sample).

Question:

Split the data

- Set X_train, X_test, y_train and y_tests to match the description in the comment
- 90% will be used for training the model
- 10% will be used as validation (out of sample) examples

Hint:

- Use `train_test_split()` from `sklearn` to perform this split
    - Set the `random_state` parameter of `train_test_split()` to be 42

We will help you by

- Assigning the feature vectors to X and the labels to y
- Flattening the two dimensional spatial dimensions of the features to a single dimension

In [ ]:
```
from sklearn.model_selection import train_test_split

y = labels
X = data_bw

X_train = None
X_test = None
y_train = None
y_test = None

### Flatten X
X = X.reshape(X.shape[0], -1)

# Split data into train and test
# Create variables X_train, X_test, y_train, y_test
#   X_train: training examples
#   y_train: labels of the training examples
#   X_test:  test examples
#   y_test:  labels of test examples

# YOUR CODE HERE
raise NotImplementedError()

print("X_train shape: ", X_train.shape)
print("X_test shape: ", X_test.shape)
print("y_train shape: ", y_train.shape)
print("y_test shape: ", y_test.shape)
```

Your expected outputs should be following

```
X_train shape:  (3600, 6400)
X_test shape:  (400, 6400)
y_train shape:  (3600,)
y_test shape:  (400,)
```

```
In [ ]:
```

Prepare the data and Classifier

Questions:

You will transform the data and create a Classifier.

The requirements are as follows:

- Transform the features (i.e., the pixel grids) into standardized values (mean 0, unit standard deviation)
    - Set a variable `scaler` to be your scaler
- Create an `sklearn` Classifier
    - Set variable `clf` to be be your Classifier object
    - We recommend trying Logistic Regression first
        - `sklearn`'s implementation of Logistic Regression has many parameter choices
        - We recommend starting with the single parameter `solver="liblinear"`
        - You may want to use the `sklearn` manual to learn about the other parameters

Hints:

- Look up `StandardScaler` in `sklearn`; this is a transformation to create standardized values
- You will use transformed examples both for training and test examples
    - So be sure that you can perform the transformation on both sets of examples
- Using `Pipeline` in `sklearn`, whose last element is a model, is a very convenient way to
    - Implement transformations and perform model fitting/prediction
    - In a way that ensures that all examples, both training and test, are treated consistently
    - Enables Cross Validation without cheating

In [ ]:

```
import time
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import Pipeline


## Data Scaler
#  Create a StandardScaler object
#    scaler: sklearn standard scaler
scaler = None

# YOUR CODE HERE
raise NotImplementedError()


## Classification Model
#  Create a classifier
#    clf: sklearn classifier
#    name: string, name of your classifier
#    model_pipeline: sklearn Pipeline, if you use pipeline, please use this vari
able
clf = None
name = None

# YOUR CODE HERE
raise NotImplementedError()
```

In [ ]:

```
```

Train model

Question:

- Use your Classifier or model pipeline to train your dataset and compute the in-sample accuracy
    - Set a variable `score_in_sample` to store the in-sample accuracy

Hint:

- The `sklearn` function `accuracy_score` may be helpful

In [ ]:
```
from sklearn.metrics import accuracy_score
# Set variable
# score_in_sample: a scalar number, score for your in-sample examples
score_in_sample = None

# YOUR CODE HERE
raise NotImplementedError()

print("Model: {m:s} in sample score={s:3.2f}\n".format(m=name, s=score_in_sampl
e))
```

In [ ]:
```

```

Train the model using Cross Validation

Since we only have one test set, we want to use 5-fold cross validation check model performance.

Question:

- Use 5-fold Cross Validation
    - Set `cross_val_scores` as your scores of k-fold results
    - Set k as the number of folds
    - Report the average score

Hint:

- `cross_val_score` in `sklearn` will be useful

In [ ]:
```
# Set variable
#  scores: an array of scores (length 5), one for each fold that is out-of-sampl
e during cross-validation
#  k: number of folds
cross_val_scores = None
k = 5

t0 = time.time()

# YOUR CODE HERE
raise NotImplementedError()

print("Model: {m:s} avg cross validation score={s:3.2f}\n".format(m=name, s=cros
s_val_scores.mean()) )
```

In [ ]:
```
```

How many parameters in the model ?

Question:

- Calculate the number of parameters in your model. Report only the number of *non-intercept* parameters.
    - Set `num_parameters` to store the number of parameters

Hint:

- The model object may have a method to help you ! Remember that Jupyter can help you find the methods that an object implements.

In [ ]:
```
# Set num_parameters equal to the number of non-intercept parameters in the mode
l
num_parameters = None

# YOUR CODE HERE
raise NotImplementedError()

print("\nShape of intercept: {i}; shape of coefficients: {c}".format(i=clf.inter
cept_.shape,
                                                                      c=num_paramet
ers) )
```

In [ ]:
```

```

Evaluate the model

Question:

We have trained our model. We now need to evaluate the model using the test dataset created in an earlier cell.

Please store the model accuracy on the test set in a variable named `score_out_of_sample`.

Hint:

- If you have transformed examples for training, you must perform the same transformation for test examples !

- Remember: you *fit* the transformations only on the training examples, not on the test examples !

```
In [ ]: # Set variable to store the model accuracy on the test set
        score_out_of_sample = None

        # YOUR CODE HERE
        raise NotImplementedError()

        print("Model: {m:s} out-of-sample score={s:3.2f}\n".format(m=name, s=score_out_o
        f_sample))
```

```
In [ ]:
```

Visualize the parameters

Remember: there is a one-to-one association between parameters and input features (pixels).

So we can arrange the parameters into the same two dimensional grid structure as images.

This might tell us what "pattern" of features the model is trying to match.

```
In [ ]: helper.model_interpretation(clf)
```

Further Exploration (Optional)

Now you can build your own model using what you have learned from the course. Some ideas to try:

- Was it a good idea to drop the "color" dimension by converting the 3 color channels to a single one ?
- Can you interpret the coefficients of the model ? Is there a discernible "pattern" being matched ?
- Feature engineering !
    - Come up with some ideas for features that may be predictive, e.g, patterns of pixels
    - Test them
- Use Error Analysis to guide your feature engineering
- Add a *regularization penalty* to your loss function
    - How does this affect
        - The in-sample fit ?
        - The visualization of the parameters
    - **Hint**: The `sklearn` `LogisticRegression` model
        - has several choices for the `penalty` parameter
        - has a variable value for the regularization strength parameter `C`

Observe the effect of each change on the Loss and Accuracy.

In [ ]:
```
print("Done")
```