CS3VI18 - Task 1

Lucille L. Blumire

February 18, 2019

The objective of this project is to classify as accurately as possible the ground level topology of an area based on an aerial picture, as well as LIDAR response times and a Near-Infrared image.

This is to be done through means of supervised learning. A file providing actual ground level topology has been provided alongside the image data. Samples will be taken from this for use with a supervised learning technique, and can be used to assess the accuracy of the final result of estimating the most likely ground level topology based on the image inputs.

1 Initial Setup

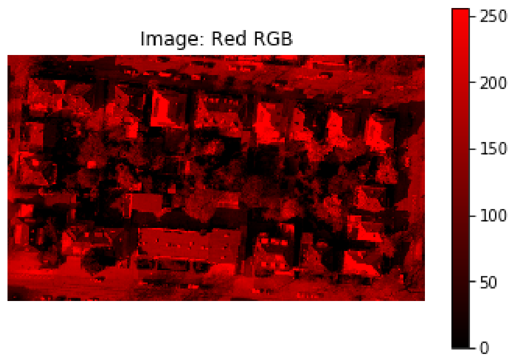
The initial setup serves mostly simply to establish the functions of the code, as well as initializing certain configuration options. This culminates in the inputs images being graphically represented.

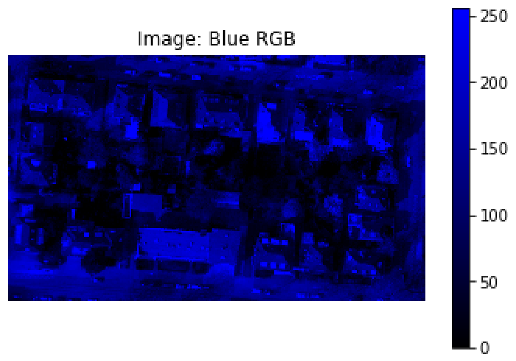
```
In [1]: # Import resources
        import scipy.io as sio
        import scipy.stats as sst
        import matplotlib.pyplot as plt
        import matplotlib.cm as pcm
        import sklearn.metrics as skm
        import numpy as np
        import pandas as pd
        from matplotlib.colors import LinearSegmentedColormap
        from functools import reduce
        from random import shuffle
        from sklearn.mixture import GaussianMixture

        # Configure color maps
        red_cmap = LinearSegmentedColormap('red_channel', segmentdata={
            'red': [(0.0, 0.0, 0.0), (1.0, 1.0, 1.0)],
            'green': [(0.0, 0.0, 0.0), (1.0, 0.0, 0.0)],
            'blue': [(0.0, 0.0, 0.0), (1.0, 0.0, 0.0)]
        })
        green_cmap = LinearSegmentedColormap('green_channel', segmentdata={
            'red': [(0.0, 0.0, 0.0), (1.0, 0.0, 0.0)],
            'green': [(0.0, 0.0, 0.0), (1.0, 1.0, 1.0)],
            'blue': [(0.0, 0.0, 0.0), (1.0, 0.0, 0.0)]
        })
        blue_cmap = LinearSegmentedColormap('blue_channel', segmentdata={
            'red': [(0.0, 0.0, 0.0), (1.0, 0.0, 0.0)],
            'green': [(0.0, 0.0, 0.0), (1.0, 0.0, 0.0)],
            'blue': [(0.0, 0.0, 0.0), (1.0, 1.0, 1.0)]
        })
        true_cmap = pcm.get_cmap("plasma", 4)
```

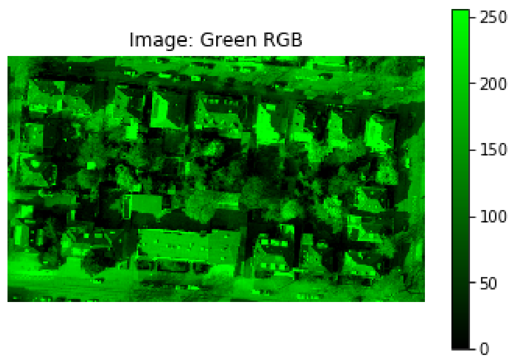
```
In [2]: # Import data
        images = {
            n: (e, plt.imread("res/" + n + ".bmp"), c)
            for (n, e, c) in [
                ("r", "Red RGB", red_cmap),
                ("b", "Blue RGB", blue_cmap),
                ("g", "Green RGB", green_cmap),
                ("fe", "LIDAR First Echo", "viridis"),
                ("le", "LIDAR Last Echo", "viridis"),
                ("nir", "Near Infra-Red", "inferno"),
            ]
        }
        ground_truth = sio.loadmat("res/ground_truth.mat")["labelled_ground_truth"]
```

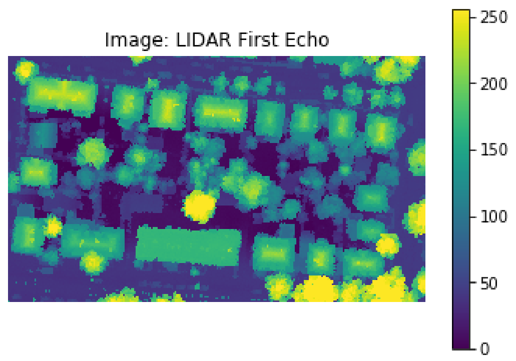
```
In [3]: # Create DataFrame of samples

        data = {}
        data["X"] = []
        data["Y"] = []

        for imagename, image in images.items():
            data[image[0]] = []
            for y in range(ground_truth.shape[0]):
                for x in range(ground_truth.shape[1]):
                    data_magnitude = ground_truth.shape[0] * ground_truth.shape[1]
                    if (len(data["X"]) < data_magnitude):
                        data["X"].append(x)
                        data["Y"].append(y)
                    data[image[0]].append(image[1][y, x])

        data = pd.DataFrame(data)
```

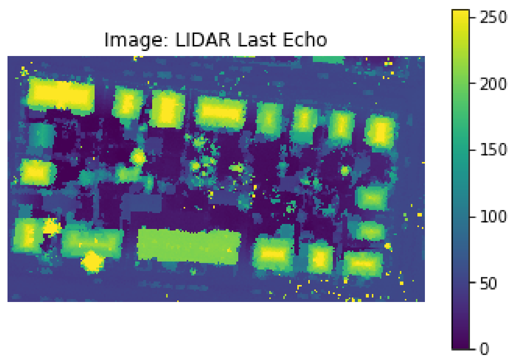
```
In [4]: # Output data for visual inspection
        for (key, (name, value, colors)) in images.items():
            fig = plt.figure()
            plt.axis('off')
            plt.title(f"Image: {name}")
            plt.imshow(value, interpolation='nearest', cmap=colors)
            plt.colorbar()

        plt.figure()
        plt.axis('off')
        plt.title("MAT Data: Ground Truth")
        plt.imshow(
            ground_truth, interpolation='nearest',
            origin='lower', cmap=true_cmap)
        plt.colorbar(ticks=range(5))
        plt.clim(0.5, 4.5)
```

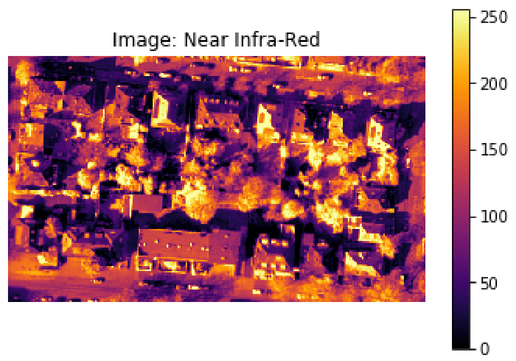


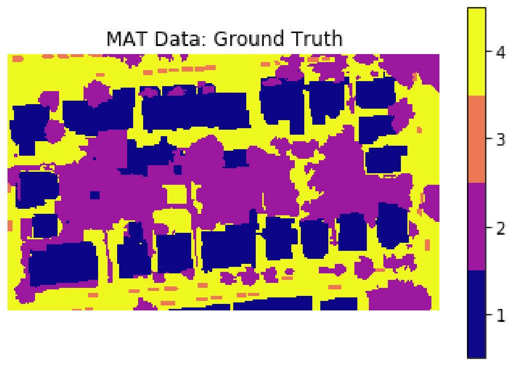












The above images demonstrate the input data, and show a variety of common patterns that we might expect the learning to pick up on. For example, road is consistently at a much lower elevation as shown in the LIDAR responses, and foliage is more green.

2 Objective 1: Select training samples from the given source data based on information in the given ground truth.

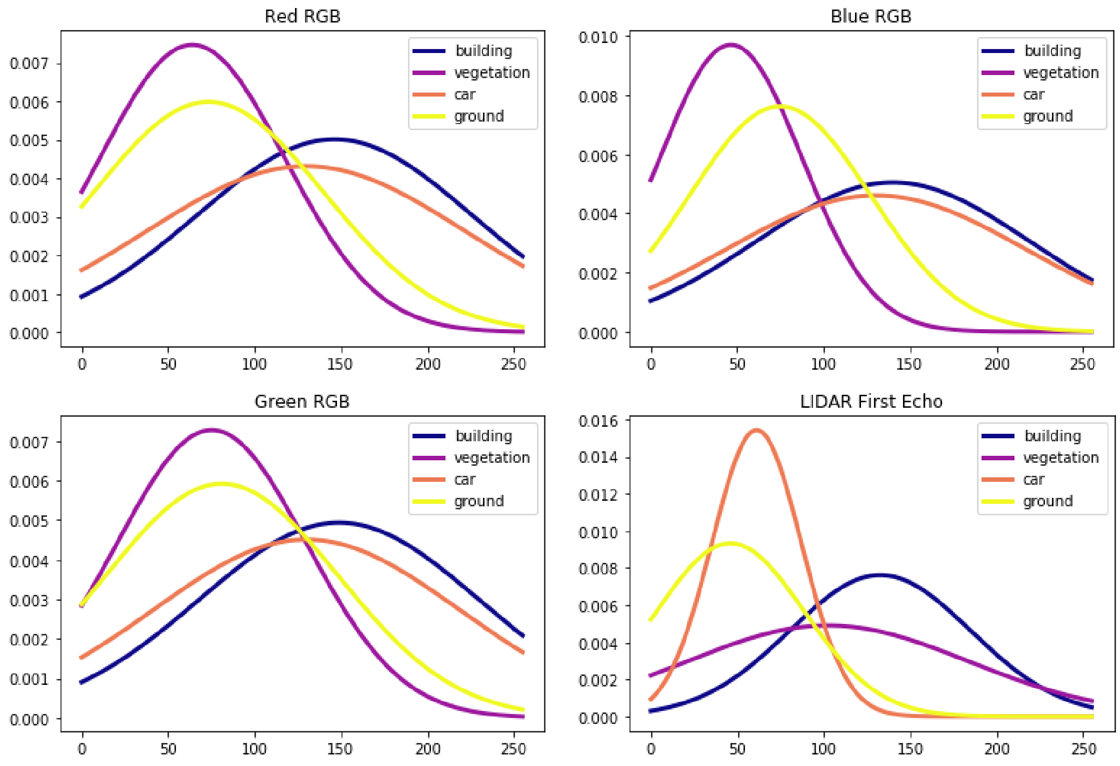
```
In [5]: # Select example coordinate indexes
        def accumulate_coordinates(acc, item):
            coord, value = item
            acc[value].append(coord)
            return acc

        true_value_coordinates = reduce(
            accumulate_coordinates,
            np.ndenumerate(ground_truth),
            {k: [] for k in np.unique(ground_truth)})

        # Randomise the samples, and then select the first 20
        for coordinates in true_value_coordinates.values():
            shuffle(coordinates)

        # Select 20 of those random samples
        random_sample_coordinates = {
            k: v[:20]
            for k, v in true_value_coordinates.items()
        }

        # Sample Data, with the structure True, X, Y, R, G, B, FE, LE, NIR
        samples = np.ndarray(shape=(20*4, 9), dtype=float)
        row = 0
        for true, random_samples in random_sample_coordinates.items():
            for x, y in random_samples:
                samples[row] = [
                    true,
                    x,
                    y,
                    images["r"][1][x, y],
                    images["g"][1][x, y],
                    images["b"][1][x, y],
                    images["fe"][1][x, y],
                    images["le"][1][x, y],
                    images["nir"][1][x, y]
                ]
                row += 1

        samples = pd.DataFrame(samples, columns=[
            "Ground Truth", "X", "Y", "Red RGB",
            "Green RGB", "Blue RGB", "LIDAR First Echo",
            "LIDAR Last Echo", "Near Infra-Red"])
```

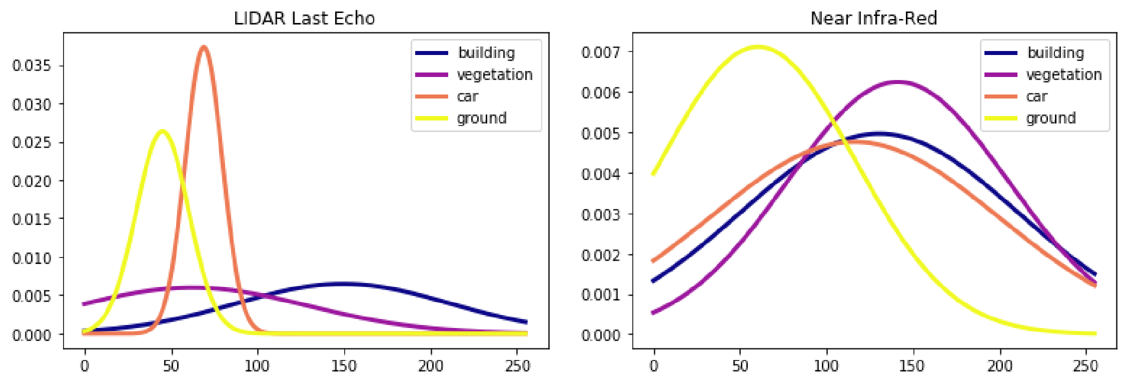
3 Objective 2: Establish Gaussian Model for Each Class with the training samples.

```
In [6]: class Gaussian:
            """
            The Gaussian class provides a simple abstraction over a gaussian
            distribution.
            """

            def __init__(self, data):
                """
                Initialise the Gaussian Distribution

                Parameters
                ----------
                data : array
                    The datapoints for which the gaussian distribution is being fit
                    over
                """
                self.mean = np.mean(data)
                self.std = np.std(data)

            def check_fit(self, data):
                """
                Return a likeliness estimation for a given datapoint compared with the
                gaussian distribution.

                Parameters
                ----------
                data : number
                    The data being compared to the distribution.

                Returns
                -------
                float
                    The likeliness that the point lies within the distribution.
                """
                return sst.norm.pdf(data, self.mean, self.std)

            def __repr__(self):
                return f"(m={self.mean}, s={self.std})"
```

```
In [7]: # Store and compute the gaussian distributions for each input space and ground
        # category.
        sample_sets = {
            "building": samples[samples["Ground Truth"] == 1].drop(
                ["Ground Truth", "X", "Y"], axis=1),
            "vegetation": samples[samples["Ground Truth"] == 2].drop(
                ["Ground Truth", "X", "Y"], axis=1),
            "car": samples[samples["Ground Truth"] == 3].drop(
                ["Ground Truth", "X", "Y"], axis=1),
            "ground": samples[samples["Ground Truth"] == 4].drop(
                ["Ground Truth", "X", "Y"], axis=1),
        }
```

```
sample_set_gaussians = {
    category: {
        column: Gaussian(set[column])
        for column in set.keys()
    }
    for category, set in sample_sets.items()
}


# Graph each fitted gaussian.
x = np.linspace(0, 255, 256)
for dataset in data.drop(["X", "Y"], axis=1).keys():
    plt.figure()
    for (i, (_, gaussian)) in enumerate(sample_set_gaussians.items()):
        gset = gaussian[dataset]
        plt.title(f"{dataset}")
        plt.plot(x, gset.check_fit(x), color=true_cmap(i), linewidth=3)
    plt.legend(sample_set_gaussians.keys())
```





The above distributions show each image and how each ground category maps to it based on the random samples and the Gaussian distribution estimations.

These will be used to evaluate how likely it is that any given pixel fits into a given class.

4 Objective 3: Apply Maximum Likelihood and Classify Each Pixel into a Class

```
In [8]: # Compute the likelihood of each category
        category_likelihood = {}

        for pixel in data.to_dict(orient='records'):
            coordinate_likelihoods = {}
            for category, gaussian in sample_set_gaussians.items():
                coordinate_likelihoods[category] = {}
                for key, value in pixel.items():
                    if key in "XY":
                        continue
                    coordinate_likelihoods[category][key] = \
                        gaussian[key].check_fit(value)

            category_likelihood[(pixel["Y"], pixel["X"])] = coordinate_likelihoods
```

```
In [9]: # Evaluate the most likely overall, based on the lowest SSE

        most_likely_category = {}
        for pixel, cats in category_likelihood.items():
            lowest_error = float("inf")
            lowest_error_cat = "ERR"
            for category, likelihoods in cats.items():
                sse = sum([
                    (1 - likelihood)**2
                    for likelihood in likelihoods.values()
                ])  # Compute SSE, lowest SSE is most likely
                if sse < lowest_error:
                    lowest_error = sse
                    lowest_error_cat = category
            most_likely_category[pixel] = lowest_error_cat
```

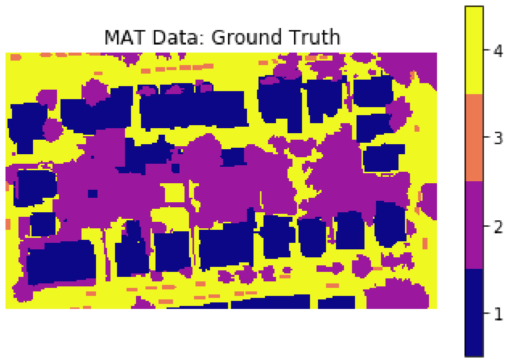
```
In [10]: ground_guess = np.ndarray(shape=ground_truth.shape)
```

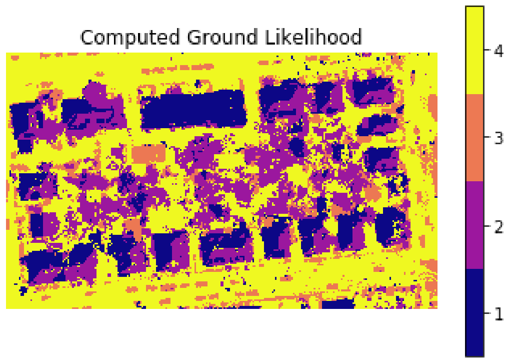
```
for coord, truth in most_likely_category.items():
    ground_guess[coord] = {
        "building": 1,
        "vegetation": 2,
        "car": 3,
        "ground": 4
    }[truth]
```

After the above code is evaluated, each pixel is given a unique prediction for which type of category it falls in. This is done by taking the sum of the squares of the probabilities that any given pixel does **not** lie within a category, and taking the lowest: and therefore the most likely category.

5 Objective 4: Evaluate the Classification Accuracy

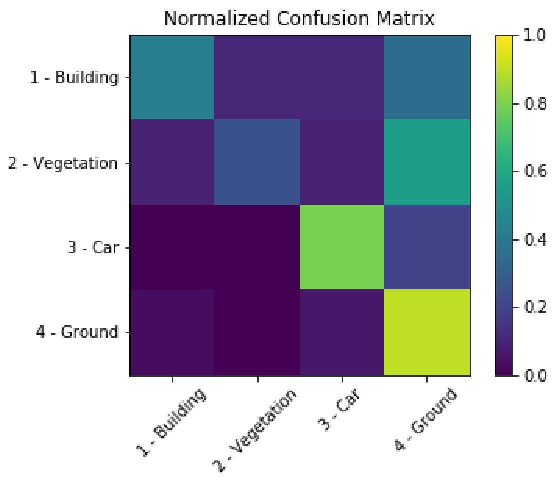
```
In [11]: plt.figure()
         plt.axis('off')
         plt.title("MAT Data: Ground Truth")
         plt.imshow(
             ground_truth, interpolation='nearest',
             origin='lower', cmap=true_cmap)
         plt.colorbar(ticks=range(5))
         plt.clim(0.5, 4.5)

         plt.figure()
         plt.axis('off')
         plt.title("Computed Ground Likelihood")
         plt.imshow(
             ground_guess, interpolation='nearest',
             origin='lower', cmap=true_cmap)
         plt.colorbar(ticks=range(5))
         plt.clim(0.5, 4.5)
```





The algorithm was highly effective at identifying the general positions of buildings, though was quick to mistake them with foliage: perhapse due to them both sharing higher Near Infrared Readings. The ground and cars were largely correct identified, likely due to their extremely low standard deviations in their LIDAR readings, as they are often of a consistent height. Foliage was the weakest to identify, likely due to it's varying colours and LIDAR response times.

```
In [12]: cm = skm.confusion_matrix(ground_truth[0], ground_guess[0])
         plt.figure()
         cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
         cm
         plt.imshow(cm, interpolation='nearest')
         plt.title("Normalized Confusion Matrix")
         plt.colorbar()
         plt.clim(0, 1)
         ticks_marks = np.arange(len(np.unique(ground_truth)))
         plt.xticks(
             ticks_marks,
             ["1 - Building", "2 - Vegetation", "3 - Car", "4 - Ground"],
             rotation=45)
         _ = plt.yticks(
             ticks_marks,
             ["1 - Building", "2 - Vegetation", "3 - Car", "4 - Ground"])
```



The above assessment based on visual inspection is validated by the confusion matrix, which shows a higher certainty regarding ground and cars than most other data points.

6 Conclusion

To conclude, supervised learning through maximum likeliness estimation over fitted Gaussian distributions was an effective means of estimating the ground level topology based on aerially sourced telemetry. The fitting process would likely benefit from a higher sample size and a wider range of images to be tested and trained against, however with the low supervised sample size of 20 I think it is highly unlikely that a large degree of over fitting occurred, though this might warrant further investigation