Sigurður Ágúst Jakobsson

Project: Assignment 4 - Classification Based on Probability

Section 2.2

Compare the accuracy of your `maximum_likelihood` and `maximum_aposteriori` by comparing the predictions from both methods to the actual test labels.

1. What is the accuracy of each method?
2. How are the confusion matrices? Use your own code from assignment 1 and compare the matrices.
3. How do you interpret the differences? Why is/is not a difference in accuracy?

I ran both methods a number of times with training ratio 0.6 in the given data splitter using Iris data. The results from one such run can be seen below.

```
Accuracy Maximum Likelihood: 0.9830508474576272

Accuracy Maximum Aposteriori: 0.9830508474576272

Confusion Matrix Maximum Likelihood (Rows=Actual, Columns=Classification):

[22  0  0]

 [ 0 16  1]

 [ 0  0 20]

Confusion Matrix Maximum Aposteriori (Rows=Actual, Columns=Classification):

[22  0  0]

 [ 0 16  1]

 [ 0  0 20]
```

Both usually scored in the high 90% with regards to accuracy and usually gave the same results. It was not often that they had different confusion matrices. The data splitter splits differently in each run so it can happen though. Similar performance in accuracy is to be expected however since we have the same number of flowers from each class in the Iris dataset; it is well balanced.

Maximum Aposteriori likelihood looks to maximize P(C_k|x)=P(x|C_k)*P(C_k), while Maximum Likelihood only maximizes P(x|C_k). If the dataset is balanced like the Iris dataset is, this means that P(C_k) is the same (or similar with different splits) for all classes. Therefore, we scale P(x|C_k) evenly with the Aposteriori method and get the same results as just looking as this value as is done in the ML method.

Since the similarity is expected I didn't do a more thorough evaluation than described above.

Independent Section

In what kind of situations would a posteriori classification be better than maximum likelihood classification? Find or create a dataset (or alter the Iris dataset) to make a posteriori classifications outperform maximum likelihood predictions.

Demonstrate this by comparing accuracy, confusion matrices, plots across different data configurations and draw your conclusions.

I would expect that a posteriori classification would perform better than maximum likelihood classification in an unbalanced dataset with outliers. I started by trying to manipulate the Iris dataset but found it hard to get significant differences in performance like this. I then tried some other datasets from sklearn, but that didn't work out well either.

I settled on generating an unbalanced dataset with outliers to classify aliens from humans. This is somewhat frivolous, but it is more fun to have some narrative in the data than just generating random points.

There are more humans than aliens in the dataset and the humans have much more difference in features of height and weight between subgroups of males, females and babies. Aliens were assigned a mean of 50kg weight, and 190cm of height. Human males were assigned a mean of 85kg weight, and 180cm of height, with higher variance than aliens. Human females were assigned a mean of 70kg weight, and 165cm of height, with higher variance than aliens. Human babies were assigned a mean of 4kg weight, and 50cm of height, with the lowest variance of all subgroups. These means were assigned relatively randomly by gut feel but were meant to have some ties to reality and fiction.

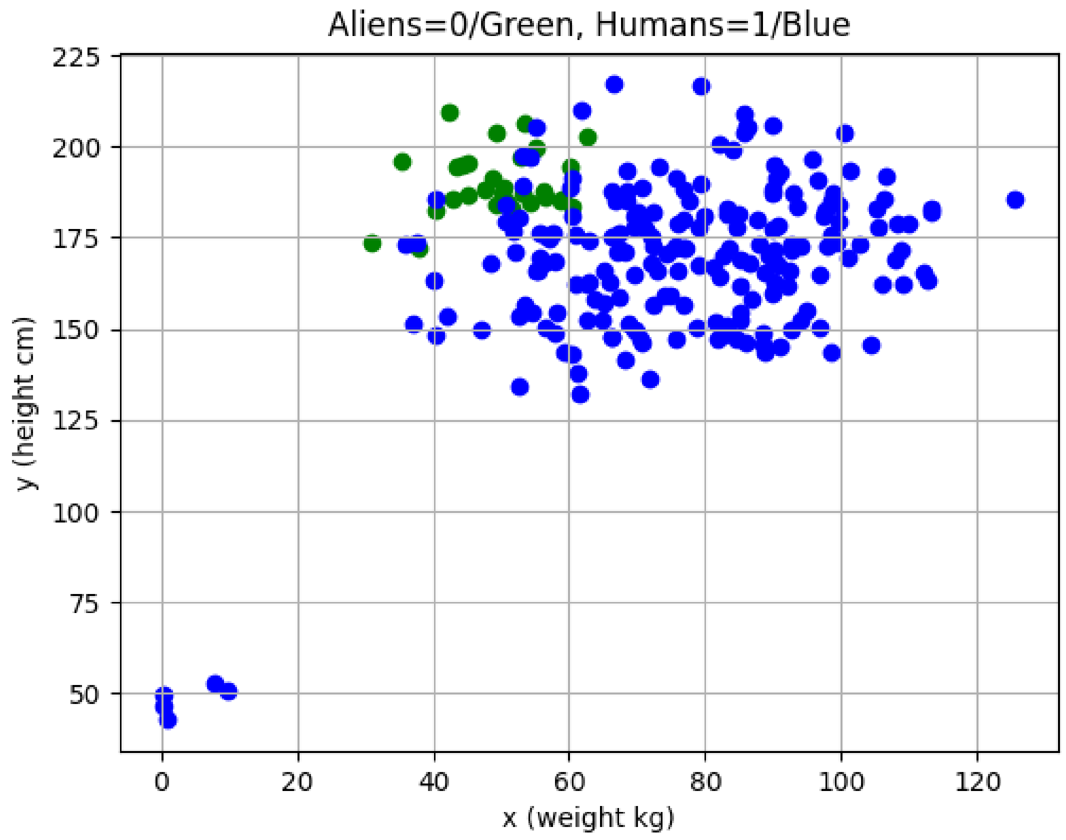
```
    classes = [0, 1]

    n_aliens = 30
    n_male = 100
    n_female = 100
    n_babies = 5 #'outliers'

    alien_features = gen_data(n_aliens, 2, np.array([50, 190]), 9) #Tall green
men
    male_features = gen_data(n_male, 2, np.array([85, 180]), 16)
    female_features = gen_data(n_female, 2, np.array([70, 165]), 16)
    baby_features = gen_data(n_babies, 2, np.array([4, 50]), 4)
```

Comparisons of the methods were run with 1000 iterations from generated data with a split ratio of 0.8 in the given data splitter. There were differences in the performance between early iterations, so some statistical analysis was needed with these repeated runs.

An example of the distribution can be seen visually below from one of the runs.



With this dataset the aposteriori method performed slightly better than the maximum likelihood method, but not with high confidence from a one-tailed test.

The maximum likelihood method had a mean of 89,8% accuracy with a one-tailed 95% upper confidence limit of 97,5%. The aposterioiri method had a mean of 93,4% accuracy with a one-tailed 95% lower confidence limit of 87,2%. While the methods are within range of each other, the aposteriori method always performed better when many iterations were averaged.

ML Mean,Max 95% One Tail: [0.8980434782608695, 0.9753182801658359]

AP Min,Mean 95% One Tail: [0.8719520678946396, 0.9339130434782609]

Confusion matrices from the runs were averaged to obtain the results below:

Averaged Confusion Matrix Maximum Likelihood (Rows=Actual, Columns=Classification):

```
[[ 5.503  0.312]

 [ 4.378 35.807]]
```

Averaged Confusion Matrix Maximum Aposteriori (Rows=Actual, Columns=Classification):

```
[[ 4.588  1.227]

 [ 1.813 38.372]]
```

The aposteriori method predicted fewer aliens than the maximum likelihood method since prior probabilities were incorporated.

Since this is a binary classifier it was easy to calculate precision and recall on data as well and analyze.

Precision

If an alien is counted as a positive then we can use the formula TP / (TP+FP) to calculate precision.

ML Precision: 0.556927436494282

AP Precision: 0.7167630057803469

The a posteriori has better precision than the ML method, so it would be especially suitable if we want to minimize false positives.

Recall

If an alien is counted as a positive then we can use the formula TP / (TP+FN) to calculate recall.

ML Recall: 0.9463456577815993

AP Recall: 0.7889939810834049

The a posteriori has worse recall than the ML method, so it is a less suitable method if we would rather like to minimize false negatives in alien detection.

So to summarize, for the generated dataset, the a posteriori method performed somewhat better than ML with regard to accuracy, but depending on the importance of precision and recall it is debatable what method would best suit specific requirements.

Generated code related to the independent section may be found below:

```python
def prior(targets: np.ndarray, classes: list) -> np.ndarray:
    '''
    Calculate the prior probability of each class type
    given a list of all targets and all class types
    '''

    #Make sure that script handles array as np array for bool summation
    targets = np.array(targets)

    #Initialize parameter values for arrays
    target_num = len(targets)
    class_num = len(classes)
    return_array = np.zeros(class_num)

    #Traverse classes
    for index, class_inst in enumerate(classes):

        #Create a bool filter and sum hits
        filter = targets[:] == class_inst
        class_count = filter.sum()

        #Add prior propability to return array for each class
        return_array[index] = class_count / target_num

    return return_array

def accuracy(test_targets, predictions):
    #Compare targets to prediction
    n = predictions.shape[0]
    hits = (test_targets == predictions).sum()

    return hits / n

def confusion_matrix(test_targets, predictions, classes):
    D = len(classes)
    confusion_matrix = np.zeros((D,D), dtype=int)

    for index in range(len(predictions)):
        confusion_matrix[test_targets[index], predictions[index]] += 1

    #Rows = actual, Columns = predictions - different from example, but there
are different axis conventions.
    return confusion_matrix

def gen_data(
    n: int,
    k: int,
    mean: np.ndarray,
    var: float
) -> np.ndarray:
    '''Generate n values samples from the k-variate
    normal distribution
    '''
    cov_identity = np.identity(k) * var**2
    data = np.random.multivariate_normal(mean, cov_identity, n)

    return data

def scatter_2d_alien_data(data: np.ndarray, targets):
    fig = plt.figure()
    ax = fig.add_subplot()

    colors = ['green', 'blue']
    for i in range(data.shape[0]):
        [x, y] = data[i,:]
        plt.scatter(x, y, c=colors[targets[i]])

    plt.title('Aliens=0/Green, Humans=1/Blue')
    plt.grid()
    ax.set_xlabel('x (weight kg)')
    ax.set_ylabel('y (height cm)')
    plt.show()

def ml_ap_compare(plot=0):

    #Try to build unbalanced dataset with outliers
    classes = [0, 1]

    n_aliens = 30
    n_male = 100
    n_female = 100
    n_babies = 5 #'outliers'

    alien_features = gen_data(n_aliens, 2, np.array([50, 190]), 9) #Tall green
men
    male_features = gen_data(n_male, 2, np.array([85, 180]), 16)
    female_features = gen_data(n_female, 2, np.array([70, 165]), 16)
    baby_features = gen_data(n_babies, 2, np.array([4, 50]), 4)

    features = np.vstack((alien_features, male_features))
    features = np.vstack((features, female_features))
    features = np.vstack((features, baby_features))

    targets = np.concatenate((np.zeros(n_aliens, dtype=int), np.ones(n_male +
n_female + n_babies, dtype=int)), axis=None)

    #Plot if relevant
    if plot == 1:
       scatter_2d_alien_data(features, targets)

    (train_features, train_targets), (test_features, test_targets) =
split_train_test(features, targets, train_ratio=0.8)

    #Calculations
    likelihoods_ml = maximum_likelihood(train_features, train_targets,
test_features, classes)
    likelihoods_ap = maximum_aposteriori(train_features, train_targets,
test_features, classes)

    predict_ml = predict(likelihoods_ml)
    predict_ap = predict(likelihoods_ap)
    accuracy_ml = accuracy(test_targets, predict_ml)
    accuracy_ap = accuracy(test_targets, predict_ap)
    #print(accuracy_ml)
    #print(accuracy_ap)

    confusion_ml = confusion_matrix(test_targets, predict_ml, classes)
    confusion_ap = confusion_matrix(test_targets, predict_ap, classes)
    #print(confusion_ml)
    #print(confusion_ap)

    return accuracy_ml, accuracy_ap, confusion_ml, confusion_ap

def multiple_alien_compare():

    iter = 0
    runs = 1000
    ml_accuracy_array = []
    ap_accuracy_array = []
    ml_confusion_array = []
    ap_confusion_array = []

    #Run multiple ML/AP comparison tests for statistical comparison
    while iter < runs:
        accuracy_ml, accuracy_ap, confusion_ml, confusion_ap =
ml_ap_compare(iter) #Show dist on 2nd run
        ml_accuracy_array.append(accuracy_ml)
        ap_accuracy_array.append(accuracy_ap)
        ml_confusion_array.append(confusion_ml)
        ap_confusion_array.append(confusion_ap)

        iter += 1

    #Statistical analysis
    ml_accuracy_mean = np.mean(ml_accuracy_array)
    ap_accuracy_mean = np.mean(ap_accuracy_array)
    ml_accuracy_sd = np.var(ml_accuracy_array)**0.5
    ap_accuracy_sd = np.var(ap_accuracy_array)**0.5

    #One tailed comparison of results with 95% confidence
    dev_95 = 1.64

    min_ml = ml_accuracy_mean - ml_accuracy_sd * dev_95
    max_ml = ml_accuracy_mean + ml_accuracy_sd * dev_95
    min_ap = ap_accuracy_mean - ap_accuracy_sd * dev_95
    max_ap = ap_accuracy_mean + ap_accuracy_sd * dev_95

    ml_range = [ml_accuracy_mean, max_ml]
    ap_range = [min_ap, ap_accuracy_mean]
    print('ML Mean,Max 95% One Tail:', ml_range)
    print('AP Min,Mean 95% One Tail:', ap_range)

    #Calculate an overall confusion matrix
    average_confusion_ml = np.zeros((2,2))
    average_confusion_ap = np.zeros((2,2))

    for i in range(len(ml_confusion_array)):
        average_confusion_ml += ml_confusion_array[i]
        average_confusion_ap += ap_confusion_array[i]

    average_confusion_ml /= len(ml_confusion_array)
    average_confusion_ap /= len(ap_confusion_array)

    print('ML Confusion\n', average_confusion_ml)
    print('AP Confusion\n', average_confusion_ap)

    #Analyze overall precision and recall

    ml_precision = average_confusion_ml[0,0] / (average_confusion_ml[0,0] +
average_confusion_ml[1,0])
    ap_precision = average_confusion_ap[0,0] / (average_confusion_ap[0,0] +
average_confusion_ap[1,0])

    print('ML Precision:', ml_precision)
    print('AP Precision:', ap_precision)

    ml_recall = average_confusion_ml[0,0] / (average_confusion_ml[0,0] +
average_confusion_ml[0,1])
    ap_recall = average_confusion_ap[0,0] / (average_confusion_ap[0,0] +
average_confusion_ap[0,1])

    print('ML Recall:', ml_recall)
    print('AP Recall:', ap_recall)
```