Classification Based on Probability

The aim of the project is to implement a classifier for the Iris dataset based on

- Maximum likelihood
- Maximum A-posteriori Classification

Section 1 - Maximum Likelihood

The maximum likelihood for classification only relies on the class conditional probabilities and ignores prior probabilities. We assume that the class conditional probability density function for the Iris dataset classes are Gaussian:

$$ p(\mathbf{x}|\mathcal{C}_k)=\frac{1}{(2\pi)^{D/2}}\frac{1}{|\mathbf{\Sigma}_k|^{1/2}} e^{{-\frac{1}{2}(\mathbf{x}-\mathbf{\mu}_k)^T\mathbf{\Sigma}_k^{-1}(\mathbf{x}-\mathbf{\mu}_k)}} $$

We will start by estimating $\mu_1, \Sigma_1, \mu_2, \Sigma_2, \mu_3, \Sigma_3$ using the training data.

Section 1.1

Create a function `mean_of_class(features, targets, class)` which returns the mean of all features which targets correspond to the given `class`.

Example inputs and outputs:

First load the data

```
features, targets, classes = load_iris()
(train_features, train_targets), (test_features, test_targets)\
    = split_train_test(features, targets, train_ratio=0.6)
```

`mean_of_class(train_features, train_targets, 0)` -> `[5.005 3.4425 1.4625 0.2575]`

Section 1.2

Create a function `covar_of_class(features, targets, class)` which returns the covariance of features which targets correspond to the given `class`. Take a look at `help.estimate_covariance` for more information on covariance.

Inputs and outputs: `covar_of_class(train_features, train_targets, 0)` ->

```
[[0.11182346 0.09470383 0.01757259 0.01440186]
 [0.09470383 0.14270035 0.01364111 0.01461672]
 [0.01757259 0.01364111 0.03083043 0.00717189]
 [0.01440186 0.01461672 0.00717189 0.01229384]]
```

Section 1.3

Create a function `likelihood_of_class(feature, class_mean, class_covar)` that returns the probability that the feature belongs to the class with the given mean and covariance matrix. To achieve this you should use `scipy.stats.multivariate_normal`, see `help.pdf`.

Example inputs and outputs:

```
class_mean = mean_of_class(train_features, train_targets, 0)
class_cov = covar_of_class(train_features, train_targets, 0)
```

`likelihood_of_class(test_features[0, :], class_mean, class_cov)` -> `7.174078020748095e-85`

Section 1.4

Create a function `maximum_likelihood(train_features, train_targets, test_features, classes)` that:

1. Estimates the mean and covariance of all classes using the training data
2. For each test sample, estimate the likelihoods that the feature belongs to any of the given classes. For n test points and c classes you should return a $[c \times n]$ numpy array.

`maximum_likelihood(train_features, train_targets, test_features, classes)` ->

```
[
    [2.3146900488825263e-149, 0.0036329728501139275, 0.09701357803849536],
    [1.8635307438480972e-67, 2.4090713729753066, 0.00026197385870806855],
    ...
    [8.159929006721418, 3.8167195682014385e-17, 6.1308262198933825e-34],
    [1.6369648758616588e-75, 0.42242605396419014, 6.512799125377976e-05]
]
```

Section 1.5.

Finally create a function `predict(likelihoods)` that, using the given likelihoods, determine a class prediction using

$$\hat{k}_n=\arg \max_k p(\mathbf{x}_n|\mathcal{C}_k)$$

Example inputs and outputs:

```
likelihoods = maximum_likelihood(train_features, train_targets, test_features,
classes)
```

`predict(likelihoods)` -> `[0 2 0 ... 0 1 2]`

Section 2 - Maximum Aposteriori classification

The problem with maximum likelihood classification is that the prior probabilities are ignored. This can be justified if the prior probabilities are the same but it is often better to discover this through the data. The maximum likelihood estimate of the prior probability of a class is simply the proportion of the number of samples of that class in the training data, i.e.

$$ p(\mathcal{C}_k) = \frac{N_k}{N} $$

Where N_k is the number of samples belonging to class $\mathcal{C}_k$ and N is the number of samples in the training sample.

Section 2.1

Create a function `maximum_aposteriori(train_features, train_targets, test_features, classes)` that predicts class likelihoods of samples from `test_features` using a-posteriori probabilities.

This function should be very similar to your `maximum_likelihood` function.

Section 2.2

This question should be answered in a pdf (you can use the same pdf as for the independent section)

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?

Independent Section

A rough idea of an independent section idea is listed below. You are free to explore different approaches.

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.

Upload a pdf with your solution Your solution should contain an explanation of what you did, any plots or relevant explanations and the results of your experiments. Include all code you generated as well.