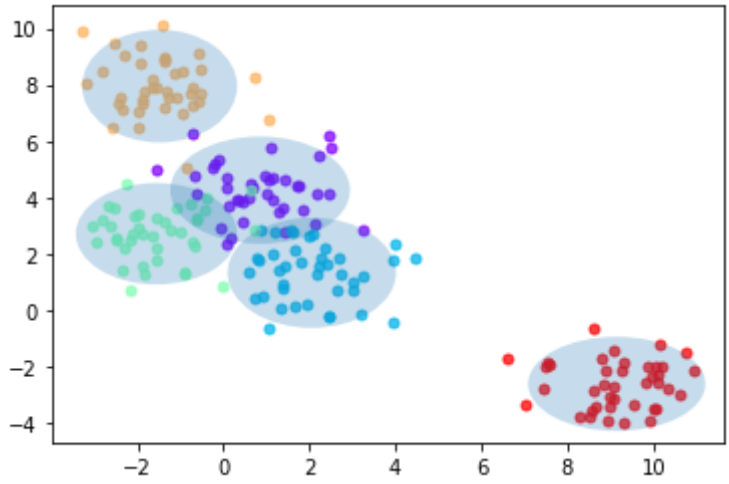
DD2421 Lab 3

Chen Yuan, Yuxuan Zhang

Assignment 1



Assignment 2

```python
def computePrior(labels, W=None):
    Npts = labels.shape[0]
    if W is None:
        W = np.ones((Npts,1))/Npts
    else:
        assert(W.shape[0] == Npts)
    classes = np.unique(labels)
    Nclasses = np.size(classes)

    prior = np.zeros((Nclasses,1))

    # TODO: compute the values of prior for each class!
    # ==========================
    for jdx, cla in enumerate(classes):
        idx = np.where(labels == cla)[0]
        prior[cla] = idx.size / Npts
    # ==========================

    return prior

def mlParams(X, labels, W=None):
    assert(X.shape[0]==labels.shape[0])
    Npts,Ndims = np.shape(X)
    classes = np.unique(labels)
    Nclasses = np.size(classes)

    if W is None:
        W = np.ones((Npts,1))/float(Npts)
```

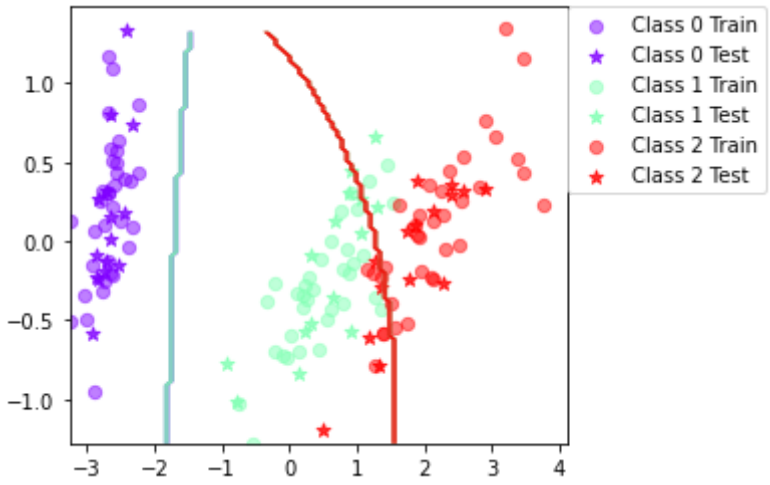
```
    mu = np.zeros((Nclasses,Ndims))
    sigma = np.zeros((Nclasses,Ndims,Ndims))

    # TODO: fill in the code to compute mu and sigma!
    # ===========================
    for jdx, cla in enumerate(classes):
        idx = np.where(labels == cla)[0]
        xlc = X[idx, :]
        mulc = np.mean(xlc, axis=0) # calculate each mu in current class
        mu[cla] = mulc
        sigma[cla] = (np.dot(np.transpose(xlc-mulc), xlc-mulc))/xlc.shape[0] #
calculate and save sigma in current class
        dsigma = np.diag(sigma[cla])
        sigma[cla] = np.diag(dsigma) # only keep diagonal
    # ===========================

    return mu, sigma

def classifyBayes(X, prior, mu, sigma):

    Npts = X.shape[0]
    Nclasses,Ndims = np.shape(mu)
    logProb = np.zeros((Nclasses, Npts))

    # TODO: fill in the code to compute the log posterior logProb!
    # ===========================
    for x in range(Npts):
        for cla in range(Nclasses):
            logProb[cla]
[x]=-0.5*np.log(np.linalg.det(sigma[cla]))-0.5*np.linalg.multi_dot([X[x]-
mu[cla],np.diag(1/np.diag(sigma[cla])),np.transpose(X[x]-
mu[cla])])+np.log(prior[cla])
    # ===========================

    # one possible way of finding max a-posteriori once
    # you have computed the log posterior
    h = np.argmax(logProb,axis=0)
    return h
```

Assignment 3

```
In [22]: testClassifier(BayesClassifier(), dataset='iris', split=0.7)
```

```
Trial: 0 Accuracy 84.4
Trial: 10 Accuracy 95.6
Trial: 20 Accuracy 93.3
Trial: 30 Accuracy 86.7
Trial: 40 Accuracy 88.9
Trial: 50 Accuracy 91.1
Trial: 60 Accuracy 86.7
Trial: 70 Accuracy 91.1
Trial: 80 Accuracy 86.7
Trial: 90 Accuracy 91.1
Final mean classification accuracy  89 with standard deviation 4.16
```

```
In [24]: testClassifier(BayesClassifier(), dataset='vowel', split=0.7)

Trial: 0 Accuracy 61
Trial: 10 Accuracy 66.2
Trial: 20 Accuracy 74
Trial: 30 Accuracy 66.9
Trial: 40 Accuracy 59.7
Trial: 50 Accuracy 64.3
Trial: 60 Accuracy 66.9
Trial: 70 Accuracy 63.6
Trial: 80 Accuracy 62.3
Trial: 90 Accuracy 70.8
Final mean classification accuracy  64.7 with standard deviation 4.03
```



(1) When can a feature independence assumption be reasonable and when not?

Ideally, these features would indeed be independent of each other in the real world. However this situation is almost impossible to exist. Therefore, when encountering difficulties in finding explicit mathematical relationships for the given features, we can try to apply Bayesian classification. We may set a certain error range, and the assumption of feature independence can be considered reasonable when the error of the results obtained using the Bayesian classifier is within the error range. A real-life use case can be for sentiment analysis to avoid curse of dimensionality.

(2) How does the decision boundary look for the Iris dataset? How could one improve the classification results for this scenario by changing classifier or, alternatively, manipulating the data?

It works for class 0 & 1, but doesn't work well for class 1 & 2. Improve by applying a boosting classifier or increasing the dimension of the data.

Assignment 4

```
for jdx, cla in enumerate(classes):
        idx = np.where(labels == cla)[0]
        xlc = X[idx, :]
        wlc = W[idx, :]
        mulc = np.sum(wlc * xlc, axis=0) / np.sum(wlc) # calculate each mu in
current class
        mu[cla] = mulc

        for m in range(Ndims):
            summation = 0 # SUM w(x - mu)^2
            for i in idx:
                sqrMul = (X[i][m] - mulc[m]) ** 2 # (x - mu)^2
                summation += W[i] * sqrMul
            sigma[cla][m][m] = summation / np.sum(wlc) # calculate diagonal
values for sigma in current class
```

Assignment 5

(1) Is there any improvement in classification accuracy? Why/why not?

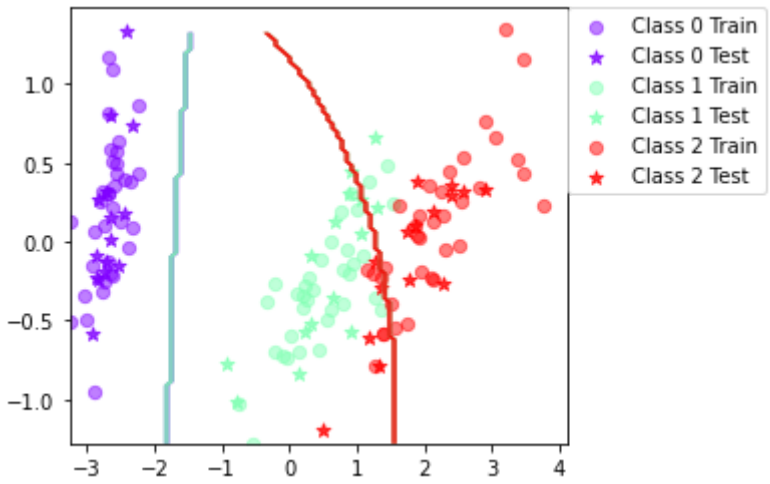
Yes, classification accuracies of both Iris and Vowel datasets increase. This is because boosting improves the performance of weak classifiers. By increasing the weights of misclassified instances, the classifier focuses more on dealing with these misclassified instances and thus improving the performance in the next iteration.

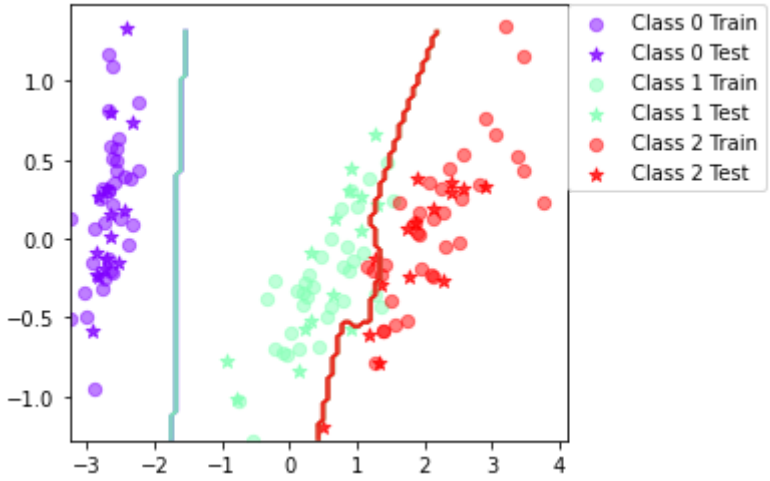
```
In [32]: testClassifier(BoostClassifier(BayesClassifier(), T=10), dataset='iris',split=0.7)

Trial: 0 Accuracy 95.6
Trial: 10 Accuracy 97.8
Trial: 20 Accuracy 93.3
Trial: 30 Accuracy 93.3
Trial: 40 Accuracy 97.8
Trial: 50 Accuracy 93.3
Trial: 60 Accuracy 95.6
Trial: 70 Accuracy 95.6
Trial: 80 Accuracy 93.3
Trial: 90 Accuracy 93.3
Final mean classification accuracy  93.9 with standard deviation 6.73
```

```
In [33]: testClassifier(BoostClassifier(BayesClassifier(), T=10), dataset='vowel',split=0.7)

Trial: 0 Accuracy 67.5
Trial: 10 Accuracy 79.9
Trial: 20 Accuracy 77.3
Trial: 30 Accuracy 75.3
Trial: 40 Accuracy 71.4
Trial: 50 Accuracy 68.2
Trial: 60 Accuracy 77.9
Trial: 70 Accuracy 79.9
Trial: 80 Accuracy 77.3
Trial: 90 Accuracy 80.5
Final mean classification accuracy  77.2 with standard deviation 3.69
```

(2) Plot the decision boundary of the boosted classifier on iris and compare it with that of the basic. What differences do you notice? Is the boundary of the boosted version more complex?

The boundary between class 0&1 is similar but the boundary between class 1&2 works better. The boundary of the boosted version seems to be more complex as it's more specific to the data points.





(3) Can we make up for not using a more advanced model in the basic classifier (e.g. independent features) by using boosting?

It depends. When the training data contains few noises and outliers, boosting is useful as we can only apply weak classifiers to avoid overfitting and save computing resources. However, when the training data contains lots of noises, boosting may enhance the focus of the noises and thus leading to big errors. Then, more advanced models might be in need.

Assignment 6

(1) Is there any improvement in classification accuracy? Why/why not?

Yes. For Iris, improvement is not that much because the dataset can originally be classified well by decision tree, for Vowel, improvement is still significant. This is because boosting improves the performance of weak classifiers. By increasing the weights of misclassified instances, the classifier focuses more on dealing with these misclassified instances and thus improving the performance in the next iteration.

```
In [9]: testClassifier(DecisionTreeClassifier(), dataset='iris', split=0.7)
```

```
Trial: 0 Accuracy 95.6
Trial: 10 Accuracy 100
Trial: 20 Accuracy 91.1
Trial: 30 Accuracy 91.1
Trial: 40 Accuracy 93.3
Trial: 50 Accuracy 91.1
Trial: 60 Accuracy 88.9
Trial: 70 Accuracy 88.9
Trial: 80 Accuracy 93.3
Trial: 90 Accuracy 88.9
Final mean classification accuracy  92.4 with standard deviation 3.71
```

```
In [10]: testClassifier(BoostClassifier(DecisionTreeClassifier(), T=10), dataset='iris',split=0.7)
```

```
Trial: 0 Accuracy 95.6
Trial: 10 Accuracy 100
Trial: 20 Accuracy 95.6
Trial: 30 Accuracy 93.3
Trial: 40 Accuracy 93.3
Trial: 50 Accuracy 95.6
Trial: 60 Accuracy 88.9
Trial: 70 Accuracy 93.3
Trial: 80 Accuracy 93.3
Trial: 90 Accuracy 93.3
Final mean classification accuracy  94.6 with standard deviation 3.65
```

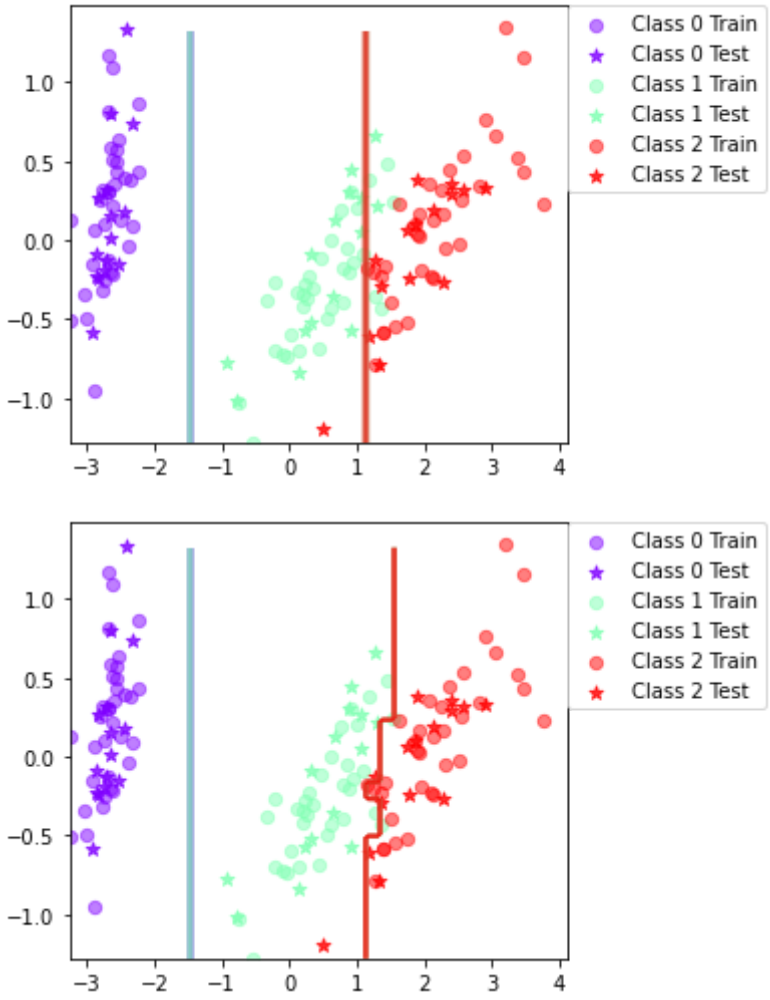
```
In [11]: testClassifier(DecisionTreeClassifier(), dataset='vowel',split=0.7)
```

```
Trial: 0 Accuracy 63.6
Trial: 10 Accuracy 68.8
Trial: 20 Accuracy 63.6
Trial: 30 Accuracy 66.9
Trial: 40 Accuracy 59.7
Trial: 50 Accuracy 63
Trial: 60 Accuracy 59.7
Trial: 70 Accuracy 68.8
Trial: 80 Accuracy 59.7
Trial: 90 Accuracy 68.2
Final mean classification accuracy  64.1 with standard deviation 4
```

```
In [12]: testClassifier(BoostClassifier(DecisionTreeClassifier(), T=10), dataset='vowel',split=0.7)

Trial: 0 Accuracy 85.1
Trial: 10 Accuracy 89.6
Trial: 20 Accuracy 87
Trial: 30 Accuracy 92.2
Trial: 40 Accuracy 84.4
Trial: 50 Accuracy 80.5
Trial: 60 Accuracy 86.4
Trial: 70 Accuracy 85.7
Trial: 80 Accuracy 89
Trial: 90 Accuracy 83.1
Final mean classification accuracy  86.6 with standard deviation 2.86
```

(2) Plot the decision boundary of the boosted classifier on iris and compare it with that of the basic. What differences do you notice? Is the boundary of the boosted version more complex?

The boundary between class 0&1 is the same but the boundary between class 1&2 works better. The boundary of the boosted version seems to be more complex as it's more specific to the data points.



(3) Can we make up for not using a more advanced model in the basic classifier (e.g. independent features) by using boosting?

(Same as 5.3) It depends. When the training data contains few noises and outliers, boosting is useful as we can only apply weak classifiers to avoid overfitting and save computing resources. However, when the training data contains lots of noises, boosting may enhance the focus of the noises and thus leading to big errors. Then, more advanced models might be in need.

Assignment 7

If you had to pick a classifier, naive Bayes or a decision tree or the boosted versions of these, which one would you pick? Motivate from the following criteria:

If there exists certain number of **outliers**, do not use boosting as it increases the weights of outliers and thus increasing errors. For naive Bayes, a small proportion of outliers won't affect much, while a large proportion of outliers will affect the calculation of probability and thus overfitting. For decision tree, it has the pruning method to avoid overfitting.

If only **part of the feature space is irrelevant**, then naive Bayes may not perform well, as it assumes that features are conditionally independent. Therefore, decision tree splits w.r.t. one feature at a time, without considering the dependency of features, thus not being affected. Besides, applying boosting will be better.

Considering **predictive power**, as naive Bayes is based on probablistic methods, it can use relatively little data to generate parameters thus not requiring a large training set. Therefore, when training set is relatively small, naive Bayes might be better. But performance of them always depends on the situation, e.g., different situations with outliers, relation in feature space as before.

They both work for **mixed types of data: binary, categorical or continuous features, etc**.

For **scalability**, naive bayes takes measure to solve curse of dimensionality by assuming conditionaly indenpendent. However, this may also bring the risk when there exist many relations in the feature space. While decision tree can also use pruning to neglect unimportant features when the dimension is large. Besides, naive bayes works well with small dataset as mentioned before. While the performance of both may improve with larger dataset. But the pruning process for decision tree with larger dataset might also be more complex.