# Assignment 2: Learning Bayesian Networks

Peter Szabo

## 1. Introduction

In the report our task was to implement the Navie Bayes and Linear Gaussian Model.
The Naive Bayes is a simple probabilistic classifier, it is called naive, since it relies on the assumption of independence between features. It means, that we assume that every feature contributes independently to the final probability. During the training we independently manage each feature. The training set is separated for each class, and in each joint for x, y and z a different gaussian will be fitted using all the variables belonging to them.
But in the reality there may me some correlation between the classes. This is true for our dataset as well, since the bodyparts possible position highly depends on each other. It means, that the linear Gaussians takes into consideration the dependencies. Each child will depend on its parent. In our example we had maximum one parent per each variable.
During my work I used MSRC-12 Kinect gesture data set of Microsoft Research Cambridge. The dataset contained 4 types of movement, recorded from 30 individuals. The type of movements can be seen from multiple point on Figure 1 and Figure 2.

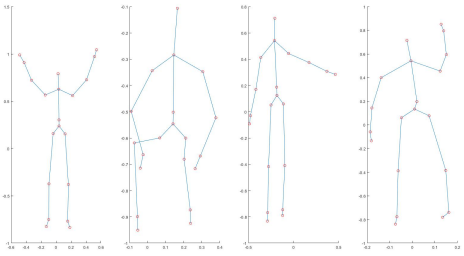

**Figure 1:** Positions in order: Class 1: "arms lifted"; Class 2: "crouched"; Class 3: "right arm extended to the side"; Class 4: "right arm extended to the front"

The class of each person is defined by the three dimensional coordinate of 20 joints.
The connectivity of the joints resulted a dependency between each other. This connectivity was also given in a variable.

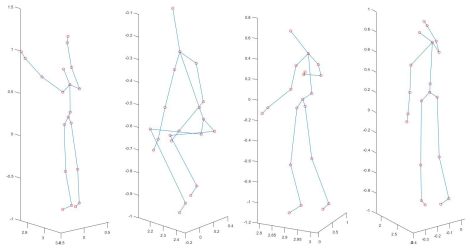

**Figure 2:** Positions in order: Class 1: "arms lifted"; Class 2: "crouched"; Class 3: "right arm extended to the side"; Class 4: "right arm extended to the front"

## 2. Code description and documentation

In this chapter I will summarize my code, and describe the way of its working and the decisions I made during coding.
Due to the inspection of the data a few observations could be made. Our dataset will contain N instances with the dimesnion of 20x3. Moreover, each joint has maximum one parent.
The $learn\_model()$ function is responsible for learning the model. First of all I convert the connectivity matrix to a form, that's given in the example. Since priors were not given, they are set equally. Afterwards I initialize the empty model, and then the program branches depending on the connectivity matrix. If there are none, the simple Naive Bayes will be used, where each joint will only depend on itself, and not the others. For each x,y,z coordinate of the joints, we fit a Gaussian. The parameters of the Gaussian are calculated from the training data using $fit\_gaussian()$ function.
But if there is a connectivity matrix, it will be build in the model, and each coordinate of each joint will be calculated based on the connectivity. I will separate each class and will do the learning separately. I find the parent of each join (assuming they only have maximum one), and then using the $fit\_linear\_gaussian()$ I calculate the parameters for each joints coordinates, the learned way. This is shown on Figure 1.

Otherwise, if a given joint does not have a parent,

$$\begin{pmatrix} E_D[Y] \\ E_D[X_1 Y] \\ \vdots \\ E_D[X_K Y] \end{pmatrix} = \begin{pmatrix} 1 & E_D[X_1] & \cdots & E_D[X_K] \\ E_D[X_1] & E_D[X_1 X_1] & \cdots & E_D[X_1 X_K] \\ \vdots & \vdots & \ddots & \vdots \\ E_D[X_K] & E_D[X_K X_1] & \cdots & E_D[X_K X_K] \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_K \end{pmatrix}$$

Solve with linear algebra $\quad \mathrm{b} = \mathrm{A}\ \mathrm{x} \qquad \mathrm{x} = \mathrm{A}^{-1} b$

$$\sigma^2 = Cov_D[Y;Y] - \sum_i \sum_j \beta_i \beta_j Cov_D[X_i; X_j]$$

$$\mathrm{Cov}_D[X;Y] = E_D[XY] - E_D[X]E_D[Y]$$

**Figure 3:** Fitting of Linear Gaussian

the Naive Bayes will be used.
Finally we return model, as it is specified in the description.
The classification is done by *classify_instances*() function. It takes the model and the data, and return the likelihoods. It works with both Navie Bayes and Linear Gaussian. The function calls *normalize_logprobs*() which calculates the likelihood of a given instance belonging to for each class.
As for me, I modified the logprob calculation, normalizing the values between 1 and 0, creating probabilities. Also different evaluations can be found in the code.
For more detailed description, in the code more explanation and reasoning can be found.
For coding I used python, and then I converted to jupyter notebook. During the assignment I attach both.

## 3. Description of evaluation

During my work I experimented with different kind of evaluation techniques. The algorithm outputs for each instance a log-likelihood for each class. During evaluation procedure we had to split the data set in order to create a training and testing dataset and avoid the possibility of over fitting. To begin with, I experimented with the simple split, assigning two-third of the variables to the training and the rest to the testing class. Since our dataset was ordered it was incredibly important to pay special attention to the way of splitting. If I were just cut the dataset at some point, it would have resulted that some labels are over-represented and some of them would be missing completely from the training or the testing dataset. In order to avoid that in each splitting I always applied some kind of random shuffling or stratification method. The show the importance of this, if I were simply split the dataest at two-third, it would mean, that the test would only contain the mostly one label, which is not present at the training resulting an approximately zero percent accuracy.

To split the dataset I employed other splitting methods: StratifiedKFold cross validation. First, the dataset is shuffled, and then split into K parts. It is important to point out, that the data is shuffled only once, so there is no re-sampling. It is prohibited, in order to prevent over-fitting. It also has stratification. The data into multiple subsets, and on each subset there will be a training and testing separately, and the final accuracy is the average of each.

The choice of accuracy measure was also an important metric. First of all I used the most simple technique, accuracy. I took the highest log-likelihood, and consider correct, if it is the expected label.
To get a more accurate perspective on the data I decided to use Receiver Operating Curve (ROC). It displays true positive rate on the X axis and the false positive rate on the X axis. The ideal plot is, when the curve goes to the top left corner, meaning no false positive rate, and 1 true positive rate. It also means a larger area under the curve.
The steepness has also a meaning, the steeper the more accurate we are, and having only the x=y curve would be the random result in binary classification. However in our case it was a bit different, since we had a multiclass problem. For multiclass problem I used One-vs-Rest. This way I reduced the problem to a binary one, and did the one-vs-rest for each separate class.

## 4. Results

The simple accuracy of the Naive Bayesian with simple splitting resulted 91.581 percent accuracy. Whereas the simple Gaussian model resulted a 94.71 percent.
Since the cross validation proved to be a more accurate measurement, I prefered to use it throughout the evaluation. I split the data into K=5 parts. The accuracy of the Naive bayes and the Simple Gaussian can be seen in the first table, with the last column as the average.

| | | | | | | |
|---|---|---|---|---|---|---|
| Naive Bayes | 0.9756 | 0.9073 | 0.9657 | 0.9926 | 0.8725 | 0.942778 |
| Simple Gaussian | 1.0 | 0.9585 | 0.9828 | 0.9877 | 0.9313 | 0.972107 |

**Table 1:** Result of Cross Validation

Increasing K increases the running time, but it results a better and more accurate picture of the error. Since I didnt had the hardware needed, I didnt run leave on out cross validation.

The Table 2. and Table 3. shows the confusion matrices.

| | 1 | 2 | 3 | 8 |
|---|---|---|---|---|
| 1 | 94.4 | 0 | 0.4 | 6.8 |
| 2 | 0 | 99.8 | 0 | 0 |
| 3 | 0 | 0 | 94 | 10.4 |
| 8 | 0 | 0 | 5.6 | 97.4 |

**Table 2:** Confusion matrix of Naive Bayes

| | 1 | 2 | 3 | 8 |
|---|---|---|---|---|
| 1 | 96 | 0 | 0 | 5.6 |
| 2 | 0 | 100 | 0 | 0 |
| 3 | 0 | 0 | 102.2 | 2.2 |
| 8 | 2 | 0.4 | 1.2 | 99.4 |

**Table 3:** Confusion matrix of simple Gaussian

## 5. Description of results

As we can see from the results, we achieved with both approaches a decent accuracy.

During the evaluation it is a good idea to take a look at the samples, the model got wrong, since it gives some intuition about the kinds of examples the model finds difficult.

As we can see from the Table 2. the Navie Bayes had the most of the problem with the 8th class in terms of false positive classification. Even though in most of the cases it predicted correctly, we can say that this label was confused in the most of the time. 10 percentage of the third class got wrongly classified, and it was also significant in other cases. I could mean, that the third position is easily mistaken with the last one. by inspecting the data, it can be easier to explain, since in both of the position is the right arm extended into different direction. While the other parts of the body is not that important in the pose, these classes will be harder to differentiate.

It is also worth to notice, that our classifier was the most accurate in the case of the second position. As we can see, while classifying the second position it made no error. This can be due to the fact, that the crouched position is quite unique, and is more characteristic and more differentiable from the rest.

Finally, it is also observable, that the false negative errors are negligible.

The simple Gaussian model had the similiar distribution of erros, but with lower occurence. And the other difference was, that it had more false positive results with the eighth body position.

I addition I also plotted some arbitrary poses I order to have a better look to the data. The results can be seen on the Figure 6.

As Figure 6 shows the data also shows a great variability which could result some error. It can be due to the noisy recordings. Also the intra class variability is a possible source of error.

On Figure 7. we can see different individuals on the same class, and how their position or alignment behaves.

They were not standing on the same position, which results different coordinates. It explains why the model, that takes the connectivity into consideration is more powerful. If we shift the person the relative position of the joints will not get distorted, while, without the connectivity, the points are harder to find. This shows, the benefit of the Linear Gaussian over the naive Bayes.

Plotting the persons also gave the insight, that during training it is worth to prefer to maximize the number of different person in a learning batch, since one individual shows less variability.

On Figure 4. and Figure 5. we can see the ROC results. They confirm our results, that our accuracy was

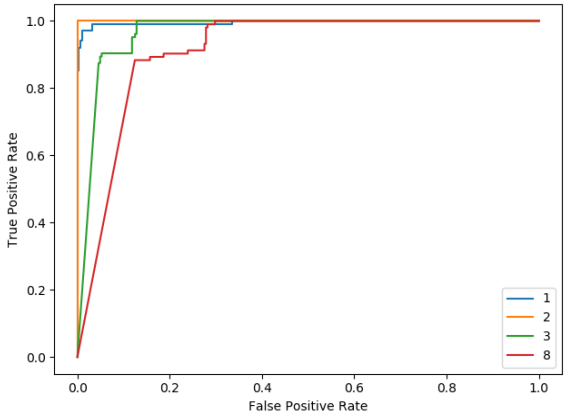


**Figure 4:** ROC curve of Naive Bayes

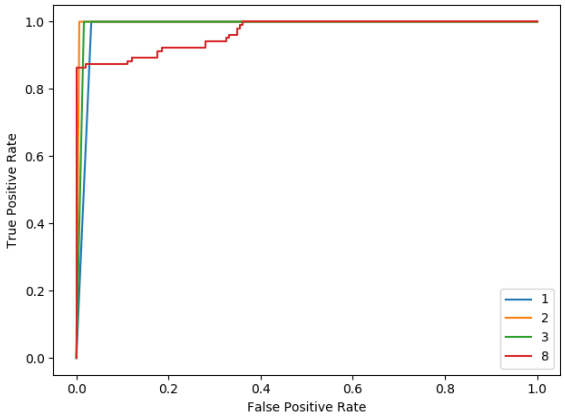


**Figure 5:** ROC curve with Linear Gaussian Model

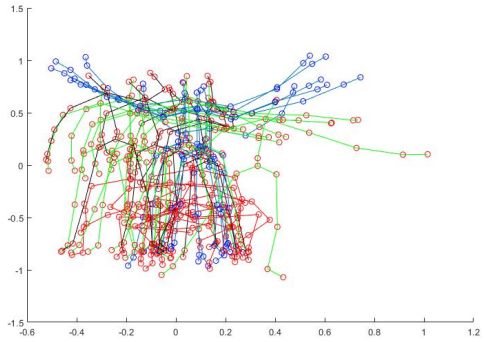

**Figure 6:** Different positions plotted on each other, first label in blue, second in red, third in green, and eigth is in black

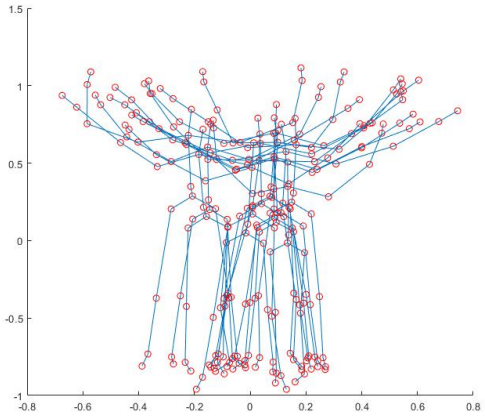

**Figure 7:** Intraclass variability in label 1, comparing recordings from multiple individuals

quite decent. Also we can observe that Naive Bayes has a smaller Area under the Curve, and also it is visible, that class 3 and 8 had the most error.

Finally more data could improve our results.

## 6. Conclusion

Despite the naive design and simplicity of the model we can clearly observe, that it also works well in complex scenarios. This can be due to the fact that our model was very simple, each variable had maximum one parent. On the other hand it was a realistic scenario, which proved, that the Naive Gaussian could be applied in those cases. But while choosing the model we should always take into consideration the complexity, in a more complex model, the Naive Bayesian could perform worse due to its primary assumption.
Having probabilities as outputs also counts as a benefit of the algorithm, since it provides more insight to the decision, and as we could see, in some false decision, the probability is also smaller.

As our results were quite accurate, we could imply, that in this case there is no need to use more complex models. Still doing so would result the possibility of over fitting or learning the noise.

## 7. Sources


1. Lesson materials
2. `https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html`