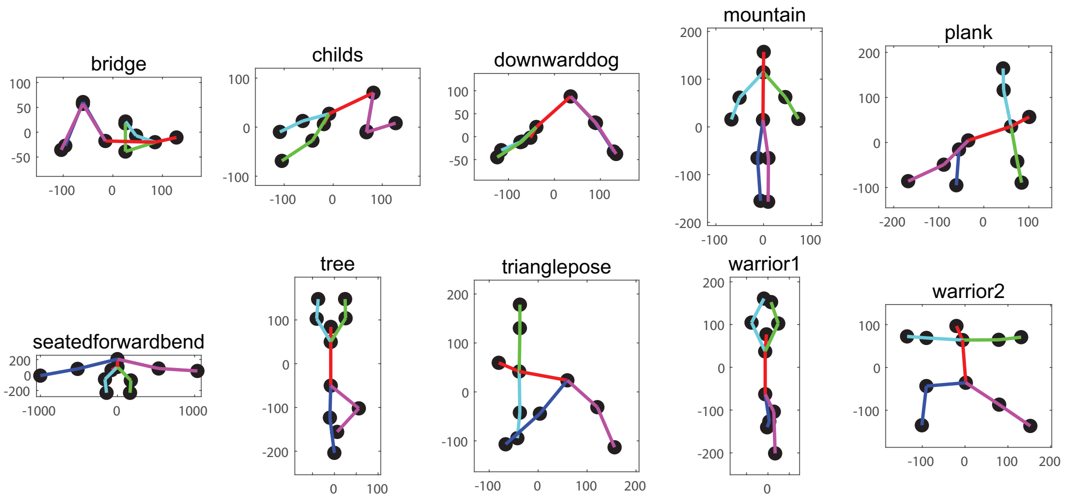
[redacted]

Project 1: Pose classification with naïve Bayes

[redacted] [redacted]
[redacted] [redacted]
[redacted] [redacted]
[redacted] [redacted]



Example instances from the yoga pose classification dataset

Overview

Human pose recognition, and related tasks like gesture recognition and action recognition, are important tasks for AI systems that interact with people. There are many algorithms for classifying pose; for example, deep convolutional neural networks can be trained to classify pose directly from images. However, these methods tend to be “black boxes” and it’s difficult to understand what features they are using. Another approach, which we will use in this Project, uses neural networks to first identify keypoints corresponding to the main parts of the body (as shown above), and then learns to recognize pose based on the positions of these body parts.

In this project, you will implement a supervised naïve Bayes learner to classify pose from keypoints provided by a deep convolutional neural network. You will train, test, and evaluate your classifier on a provided dataset, and then you will have a choice of either extending this basic model in various ways, or using it to answer some conceptual questions about naïve Bayes.

Naive Bayes classifier

There are some suggestions for implementing your learner in the "Naïve Bayes" and "Discrete & Continuous Data" lectures, but ultimately, the specifics of your implementation are up to you. Your implementation must be able to perform the following functions:

- `preprocess()` the data by reading it from a file and converting it into a useful format for training and testing
- `train()` by calculating prior probabilities and likelihoods from the training data and using these to build a naive Bayes model
- `predict()` classes for new items in a test dataset
- `evaluate()` the prediction performance by comparing your model's class outputs to ground truth labels

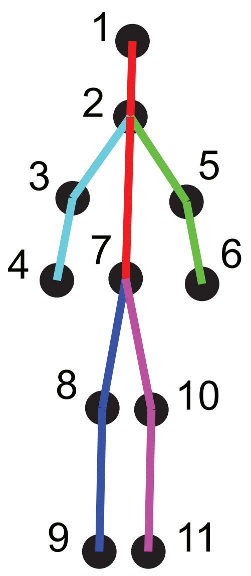
Your implementation should be able to handle numeric attributes and it should assume that numeric attributes are Gaussian-distributed. Your model will not be expected to handle nominal attributes. Your implementation should actually compute the priors, likelihoods, and posterior probabilities for the naïve Bayes model and not simply call an existing implementation such as `GaussianNB` from `scikit-learn`.

Data

The data for this assignment is drawn from a yoga pose classification dataset created by Anastasia Marchenkova and released online here. Separate training and test datasets are provided; please report your results on the provided test set.

The pose keypoints were produced by a computer vision algorithm based on OpenPose. The algorithm identifies 11 keypoints on the body and returns their x and y values. The data is provided as a csv file; the first column is the name of the yoga pose and the remaining columns are the keypoints (11 x values followed by 11 y values). The body parts represented by each of the 11 keypoints are shown to the right.

Since the keypoint data was generated by a computer vision algorithm, it may contain some errors. Some instances have missing keypoints because a part of the body was not visible in the original image or the algorithm failed to detect it. Missing keypoints have x and y values of 9999.



Keypoint diagram

Implementation tips

In the training phase of your algorithm, you will need to set up data structures to hold the prior probabilities for each class, and the likelihoods $P(x_i|c_j)$ for each attribute x_i in each class c_j. Recall that you will need two parameters (mean and standard deviation) to define the Gaussian distribution for each attribute $\times$ class. A 2D array may be a convenient data structure to store these parameters.

Multiplying many probabilities in the range $(0, 1]$ can result in very low values and lead to *underflow* (numbers smaller than the computer can represent). When implementing a naïve Bayes model, it is strongly recommended to take the $log()$ of each probability and sum them instead of multiplying. E.g., instead of computing:

$$P(c_j)\prod_i P(x_i|c_j) \tag{1}$$

compute:

$$log(P(c_j)) + \sum_i log(P(x_i|c_j)) \tag{2}$$

Questions

The following problems are designed to pique your curiosity when running your classifier(s) over the given dataset and suggest methods for improving or extending the basic model.

1. Since this is a multiclass classification problem, there are multiple ways to compute precision, recall, and F-score for this classifier. Implement at least two of the methods for computing multiclass precision and recall from the "Model Evaluation" lecture slide 37 (e.g., macro-averaging) and discuss any differences between them. (The implementation should be your own and should not just call a pre-existing function.)

2. The Gaussian naïve Bayes classifier assumes that numeric attributes come from a Gaussian distribution. Is this assumption always true for the numeric attributes in this dataset? Identify some cases where the Gaussian assumption is violated and describe any evidence (or lack thereof) that this has some effect on the NB classifier's predictions.

3. Implement a kernel density estimate (KDE) naïve Bayes classifier and compare its performance to the Gaussian naïve Bayes classifier. Recall that KDE has kernel bandwidth as a free parameter – you can choose an arbitrary value for this, but a value in the range 5-25 is recommended. Discuss any differences you observe between the Gaussian and KDE naïve Bayes classifiers. (As with the Gaussian naïve Bayes, this KDE naïve Bayes implementation should be your own and should not just call a pre-existing function.)

4. Instead of using an arbitrary kernel bandwidth for the KDE naïve Bayes classifier, use random hold-out or cross-validation to choose the kernel bandwidth. Discuss how this changes the model performance compared to using an arbitrary kernel bandwidth.

[redacted]