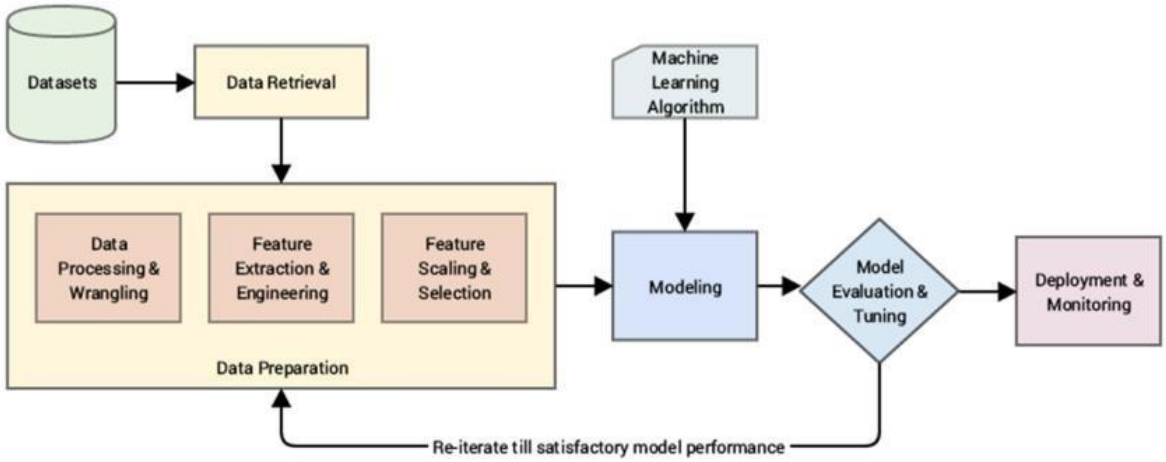
Classification of Arrhythmia: Complete Project Workflow

Work progress diagram



I. Define Problem and Get Data

Machine Learning models are generally classified into two categories, classification and regression models. However, there is some additional variation amongst classification models. For example, you could be predicting a binary output such as cat vs. dog, or many outputs such as the CIFAR-10 dataset, there are many other ways to map the outputs of the classifier, and this will be reflected in the final layer of your network.

Once you have defined your problem, you should begin to think of how you are going to get your data. One of the most popular approaches is to develop a web scraping algorithm that grabs images

off of <img url = '..'> tags on websites. You can also use Mechanical Turk to outsource your data collection needs. I usually begin projects just by simply taking screenshots of images online and changing the default storage location of screenshots to facilitate data collection this way. This process is obviously very arduous, however, I find that it works fine for getting a quick prototype of the project.

Additionally, once you collect a sizeable amount of data, you will need to think of alternative data storage methods such as the cloud.

II. Convolution Architecture, Aim for Incremental Complexity

Once you have gathered enough data to get started, you will build your convolutional network. This is surprisingly easy with the TFlearn, Keras, or PyTorch APIs. The article below will give you starter code that you can easily just copy and paste into your project to have a very simple CNN architecture.

Progressive Complexity: When you are experimenting with your convolutional network, it is best to begin with a simple structure, something like 1–4 convolutional layers. You may want to add more layers as you add more data, or if your initial results are unsatisfactory. When you are experimenting with your network architecture, it is important to have a dataset reserved for testing the model. This way you can use this performance as a control to experiment with your model complexity.

III. Data Augmentation

One of the easiest ways to improve your model in performance and robustness is to add synthetic data to your dataset. The simplest way to add more data is to simply flip your images horizontally, (and vertically if relevant to your problem). Another popular approach is to randomly generate noise as a numpy array the size of your images and then add this noise to your images. Finally, if you are feeling adventurous, there is a lot of research coming out about using Generative Adversarial Networks, (GANs), to add meaningful data.

IV. Interpret Results, Confusion Matrix

This is one of the most important steps although frequently overlooked. The reason this step is overlooked is primarily because data is expensive and difficult to obtain. Therefore, it seems foolish to take a portion of your data aside solely for testing. However, this is very important for establishing benchmarks with your performance.

However, in addition to a dependent performance variable to optimize, you can also use the confusion matrix visualization to get insight into your data. By visualizing misclassified instances, you may gain insight into what types of images are underrepresented when describing the class as a whole.

V. Data Acquisition and Automated Improvement Once your model is deployed into some kind of application. You can usually find a way to collect more data and in turn improve the performance and generalizability of the model.