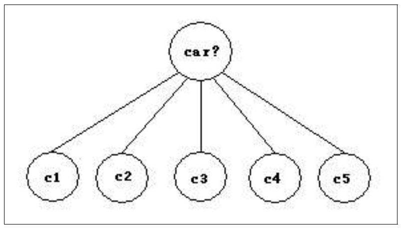
Writeup

Simon Berring, Anya Petrovskaya, Daniel Tarlow
{simonb@, dtarlow@, anya@cs.} stanford.edu

Naïve Bayes:
At a high level, we combined an assortment of classifiers using a naïve Bayes model. We have a number of child classifiers, each of which processes the current image, and returns a binary image marked with its prediction for each pixel. We then learn the parameters of the conditional probability tables using the 10 labeled training images. This gives us a modular model that allowed us to easily experiment with a number of different simple classifiers.



Inference in such a model is easy. We compare the likelihood of the features given that the hidden parent actually was a car to the likelihood of the features given that the hidden parent was not a car, then choose the most likely explanation. The true prior for car=yes is very low, however, so we boosted it (somewhat arbitrarily) by a factor of 10 to coincide more closely with the priorities of the scoring algorithm (and practical considerations, if this was a real system). The effect of this is to make us more aggressive in identifying a pixel as a car.

The advantage of taking this approach is that it gives a natural and probabilistically justified means of combining information from a number of different sources. Rather than spending a lot of time hand-tuning complex interactions between the various methods, we structured our approach in this modular manner that allowed us to easily add and subtract components. There are limitations to choosing this approach, however. Most importantly, we are assuming that each of our features is independent and that there are no complex interactions between the individual features. Because of this, we cannot express a number of interactions that would likely be important in identifying a car. Based on the scope of this assignment and the amount of time that we had to do it, however, we felt that this decision was well-justified. Experimental results also suggested that this approach performs well.

Haar Training:
The capability to construct a classifier by Haar training is built in to OpenCV. By appropriately selecting and modifying training examples, we were able to build a 14-stage classifier with very accurate results.

We experimented with a variety of training sets, including 600 images of cars and 4000 images of non-cars we found on the web. The best results were obtained however by the following training set. Negative images set consisted of the first 10 images from the sequence, where we wiped out the cars by pasting patches of similar terrain over them. We took samples of 80 cars from the image sequence for the positive set. To accommodate for frame misalignment we randomly clipped around the cars in these 80 images to generate a positive training set of 7000 images. We required a positive rate of 0.9995 and made the non-symmetry assumption.

To find cars for which only the very front makes it into the pictures, , we trained two additional classifiers: one for left side and one for right side. These classifiers were restricted to identification in the left and right regions immediately next to Stanley.

Color Segmentation:
We used the built in functionality of OpenCV to polarize the image into 4-5 different colors. We made the assumption that pixels in the rectangle that covers the bottom middle of the image are always road images, so we subtracted these colors from the image. We called the remaining colors cars.

Corner Finding:
We ran the built-in GoodFeatureToTrack functionality of OpenCV to find corners in the images. We grouped these together by moving a scrolling window of size dependent on the y-value across the image, and we called regions cars when there was a sufficient density of features. As one further refinement, we took the convex hull of these points, once we had decided that it was a car. This cut down on wasted space around cars.

Road Finding:
Since the images are so similarly oriented, we could do a fair job of road finding by simply picking the same triangle (or pentagon) out of each image. However, we improved on this strategy by: 1. finding edges (Canny) in a blurred grayscale image, 2. taking the (standard) Hough transform of the edge picture, 3. taking all edges within handcoded angle ranges corresponding to the usual placement of the road, and 4. selecting the best left- and right- edges based on a scoring (weighted Euclidean distance in $<\rho,\theta>$ space).