HW4- Machine Learning

I chose the regression approach, more specifically the many vs one algorithm. I ran 20 regressions corresponding to the 20 different cuisines in our dataset then picked the regression with highest predicted probability for a given test value (the code can actually handle any number of cuisines, and ingredients, as both of these values are read from the input files). I ran into several issues. The first being runtime. The algorithm took over 40 minutes on 1675 ingredients as independent variables in the model. In order to run the regression I had created a dataset with 1675 columns and 500+ rows. The value of most of these columns is “0” as each recipe has on average just 1020 ingredients. It turns out the columns are the driving factor in speed as this corresponds to more coeficient values to optimize over.

On 1675 columns with the original 874 row data set the runtime was 41 minutes. Originally I thought this is because for 20 cuisines I was regressing over 784 * 1675 = 1,313,200 data points. However this is a number computers can handle. It was pointed out on piazza that with that many independent variables the magnitudes on the coefficients would explode in magnitude, leading to the computer needing more time to do arithmetic with such large numbers. Indeed this is what I observed. I was also ending up with over 800 variables having coefficients that approached a singularity the computer could not compute, so they had to be thrown out. Note: changing to 0.1 and 0.9 cut the runtime in half!

To avoid the singularities caused by large number of “0” values in the dataset, and to cut down the runtime I tried sorting the data columns by the sum of the number of ‘1’ values in a given column that corresponded to an ingredient. I then played with arbitrary cutoff points at 100, 300, 600, 800 columns (ingredients). There are many problems with this approach. For example, by cutting out “rare” ingredients I lost information that could be used to uniquely identify a cuisine. For example suppose “sake” is only in 5 recipes and did not make the arbitrary cutoff but is in 100% of Japanese recipes. Then intuitively the presence of “sake” could be leveraged to persuade the predictor the recipe might be Japanese. Instead this information is Lost by this approach.

I decided to cross validate my data by first partitioning the dataset into sample data and sample inputs. Of the 784 original recipes, I randomly selected 200 to be test input and the rest became the partition data set. By including only 300 independent variables my cross validation prediction was at 36% success, low but higher than 1/20 naive guess! The way I implemented this is by taking half of the number of observations as the number of variables to include, more on this plus code is provided at the end of the document. Next I will argue that this improvement is not by coincidence.

To see why fewer independent variables can actually increase successful categorization we must consider a second issue that appears when the full 1675 independent variables (ingredients) are included in the regression. In a regression the ratio of observations to independent variables is extremely important in fitting the coefficients. Our dataset only has 784 observations, and after I partitioned the data it only had 584. This means there are 1675 independent variables but only 584 observations. To avoid overfitting the data a good rule of thumb is to have 10 observations per independent variable. Clearly this is not the case. Some ingredients have 100’s of observations but most have fewer than 6. Worse, some have 0. So by arbitrarily setting a cutoff point in the dataset I “accidentally” improved the model and reduced the time the algorithm takes.

In conclusion, the long runtime and disappointingly low success rate can be attributed to the following. By treating ingredients as either present, “1”, or not present, “0”, I had to construct new files that had (i*r) data points where, i = number_ingredients, and r = number_recipes. For the training data given this means over a million data points per cuisine, of which there are 20. However the sum of most columns is less than 6, an undesirable condition when fitting a regression. The latter of these problems can most likely be fixed by a larger dataset. My workaround is to include only half the number of observations as variables in the model, luckily in the test dataset the first 300 tend to have column sums > 10 on average, while the rest of the dataset is quite sparse. This obviously need not be true for an arbitrary dataset. And this likely leads to a bias since it looks like the latter 800 ingredients have lower column sums than the first 800 ingredients. It is important to note there is nothing inherently wrong with a regression that has a large number of binary independent variables that take on a value of “0”, and my algorithm should work nicely on a sufficiently larger dataset. The problem is the dataset is too small to deal with this intrinsic fact of the problem space, i.e recipes only have ~20 ingredients of the 1675 in question.

As mentioned above, the runtime is highly dependent on the number of variables (ingredients) in consideration. With the first ~ 300 ingredients runtime is around 40 seconds. At 800 ingredients runtime is ~2 mins. With 1675 ingredients runtime is 21 mins, but the last 800 coefficients end up getting thrown out as they approach a singularity the computer can’t compute because there are so few positive observations at this end of the data. I started running the regression with only 800 variables but then I noticed the overfitting result where if i reduced the number of variables in relation to the number of observations I got better results. The reasoning for this is given in the preceding paragraph. For this reason I made the number of ingredients considered a function of the number of observations in our dataset as follows:

```
if number_observations / MAX_INGREDIENTS < 2:
    included_vars = number_observations / 2
else:
    included_vars = MAX_INGREDIENTS
```

This is by no means desirable, but I tried many workarounds including different packages that claimed to speed up logistic regressions (they didn’t). Changing 0/1 to 0.1 and 0.9 respectively, gave a much welcomed 2x speedup. This was the best I could come up with that still appeared to result in machine learning (36% is better than naive guessing, and better than guessing based on the distribution of cuisines).

Runtime: 46 sec on 2012 Dell Intel core i5 laptop on 854 recipe dataset when only first 392 ingredients are considered. When 1675 ingredients are considered runtime is 21 mins.

Performance: 90% of CPU1 on 4 CPU machine so 25% of total CPU. 50% memory

Running it: Please see the README.txt for requirements and how to run the program (no knowledge of R is actually needed as the python driver program does everything for you)