Guthrie Alexander

CS 6316 – Machine Learning

Homework 3 writeup

Kernel accuracy ratings

Best C:

C = 13

Best gamma:

Gamma = 0.0594

Best poly:

Degree = 1

Kernel Choice	C param	degree param	Gamma param	CV Train accuracy	CV Test accuracy
Rbf	4	n/a	0.1	0.86	0.85
Rbf	4	n/a	0.05	0.85	0.84
Rbf	4	n/a	0.1	0.86	0.85
Rbf	4	n/a	1	0.89	0.83
Rbf	4	n/a	2	0.89	0.82
Rbf	4	n/a	3	0.90	0.82
Linear	4	n/a	n/a	0.85	0.85
Linear	8	n/a	n/a	0.85	0.85
Linear	13	n/a	n/a	0.85	0.85
Linear	15	n/a	n/a	0.85	0.85
Linear	17	n/a	n/a	0.85	0.85
poly	13	2	0.1	0.87	0.85
poly	13	4	0.1	0.88	0.84
poly	13	6	0.1	0.88	0.84
poly	13	8	0.1	0.88	0.83
poly	13	10	0.1	0.88	0.83
Poly	13	1	0.1	0.85	0.85
rbf	13	n/a	0.1	0.88	0.85
rbf	13	n/a	0.3	0.89	0.84
rbf	13	n/a	0.5	0.89	0.84
rbf	13	n/a	0.7	0.90	0.83
rbf	13	n/a	0.01	0.85	0.84
rbf	13	n/a	0.25	0.89	0.84

rbf	13	n/a	0.5	0.89	0.84
rbf	13	n/a	0.75	.90	0.83
rbf	13	n/a	0.1	0.88	0.85
rbf	13	n/a	0.1	0.88	0.85
rbf	19	n/a	0.1	0.88	0.85
rbf	25	n/a	0.1	0.88	0.85
rbf	7	n/a	0.1	0.87	0.85
rbf	4	n/a	0.1	0.86	0.85
linear	5	n/a	n/a	0.85	0.85
linear	6	n/a	n/a	0.85	0.85
linear	7	n/a	n/a	0.85	0.852
linear	8	n/a	n/a	0.85	0.85
linear	9	n/a	n/a	0.85	0.85
sigmoid	5	n/a	0.1	0.77	0.77
Sigmoid	6	n/a	0.1	0.78	0.78
Sigmoid	7	n/a	0.1	0.77	0.78
Sigmoid	8	n/a	0.1	0.77	0.77
sigmoid	9	n/a	0.1	0.77	0.77
sigmoid	6	n/a	0.05	0.79	0.79
Sigmoid	6	n/a	0.15	0.75	0.76
Sigmoid	6	n/a	0.25	0.68	0.68
Sigmoid	6	n/a	0.5	0.64	0.64
sigmoid	6	n/a	1	0.61	0.61
rbf	3	n/a	0.1	0.856	0.844
rbf	5	n/a	0.1	0.862	0.847
rbf	7	n/a	0.1	0.868	0.848
rbf	9	n/a	0.1	0.871	0.848
rbf	11	n/a	0.1	0.873	0.848
rbf	9	n/a	0.00001	0.76	0.76
rbf	9	n/a	0.0001	0.829	0.829
rbf	9	n/a	0.001	0.838	0.838
rbf	9	n/a	0.01	0.843	0.841
rbf	9	n/a	0.1	0.871	0.848
rbf	9	n/a	0.12	0.875	0.847
rbf	9	n/a	0.14	0.878	0.846
rbf	9	n/a	0.16	0.880	0.845
rbf	9	n/a	0.07	0.865	0.849
rbf	9	n/a	0.09	0.869	0.848
rbf	9	n/a	0.05	0.859	0.849
rbf	9	n/a	0.06	0.862	0.850
rbf	9	n/a	0.07	0.865	0.849
rbf	9	n/a	0.08	0.867	0.849
rbf	9	n/a	0.09	0.871	0.848
rbf	4	n/a	0.06	0.852	0.844
rbf	6	n/a	0.06	0.857	0.847
rbf	7	n/a	0.06	0.859	0.848

rbf	11	n/a	0.06	0.865	0.850
rbf	13	n/a	0.06	0.866	0.851
rbf	13	n/a	0.06	0.866	0.851
rbf	15	n/a	0.06	0.867	0.851
rbf	17	n/a	0.06	0.869	0.851
rbf	19	n/a	0.06	0.869	0.851
rbf	21	n/a	0.06	0.870	0.850
rbf	13	n/a	0.058	0.866	0.851
rbf	13	n/a	0.0585	0.866	0.851
rbf	13	n/a	0.059	0.866	0.851
rbf	13	n/a	0.0595	0.866	0.851
rbf	13	n/a	0.06	0.866	0.851
-	--------	n/a	------	-------	-------
rbf	13	n/a	0.0594	0.866	0.851
linear	4	n/a	n/a	0.853	0.852
linear	7	n/a	n/a	0.853	0.852
linear	**13**	n/a	n/a	**0.853**	**0.852**
Linear	15	n/a	n/a	0.853	0.852
Linear	13	n/a	n/a	0.853	0.852
Poly	28	0	0.0594	0.76	0.76
Poly	29	0	0.0594	0.76	0.76
Poly	28	1	0.0594	0.853	0.851
Poly	29	1	0.0594	0.853	0.851

Best model: linear, C=13

Comments:

I did exhaustive testing on RBF kernel models, but I couldn't find one that performed better than a linear kernel with the same penalty term (which is C=13). In the end, the differences were very, very small (i.e. the accuracy ratings were different by thousandths of a point).

I also tested poly, and sigmoid. Some poly kernels were fairly accurate, but not quite as good as the best linear that I found. The sigmoid kernels were never very accurate at all (comparatively).

To find the best SVM model, I'm strictly using the accuracy_score() function. If I were to go back, I might use a different scoring metric, as most of the different kernels seemed to plateau right around 0.85. Briefly explaining my CV code, I find the accuracy score for test folds for three different thirds of the original data (after processing), and then average them together. The model with the highest accuracy is trained on the entire data set, and THAT model is then returned from the "train_and_select_model()

function. The returned model is used on the “salary.2Predict.csv” file which yields the predictions.txt file that has been uploaded.

Addendum -- Update (Sunday night…):

I just realized that I hadn’t randomized the input dataframe… The above table is with a version of the income_classifier where I did NOT randomize the input data beforehand. I was perusing Piazza and noticed that the random.seed() was set in the provided code. I didn’t see any instructions in the HW3 pdf on randomizing the input, so I didn’t do that. I spent all the time getting the data frame preprocessed correctly, and running simulations…

I wrote a newer version of income_classifier, which randomized the data, and then ran the following kernels (just on one fold):

Kernel Choice	C param	degree param	Gamma param	CV Train accuracy	CV Test accuracy
Linear	13	n/a	n/a	0.77	0.753
Linear	4	n/a	n/a	0.767	0.746
Linear	7	n/a	n/a	0.773	0.753
Rbf	4	n/a	0.0594	0.785	0.716
Rbf	4	n/a	0.1	0.794	0.718
Rbf	7	n/a	0.1	0.801	0.717
Rbf	9	n/a	0.1	0.804	0.716
Rbf	6	n/a	0.1	0.798	0.717
Linear	9	n/a	n/a	0.773	0.753
rbf	9	n/a	1	0.855	0.708

Honestly, I ran out of time after realizing I needed to randomize the input. I had forgotten that was necessary. I would have ran more test had I realized.

I also find it curious that it takes MUCH, MUCH longer to train the model after I randomized. The models all have a much lower accuracy score as well…

In any case, I uploaded the income_classifier.py file from before randomizing the input, along with the “predictions.txt” file for kernel=linear, C=13. But I wanted to write this addendum explaining my situation.