Machine Learning: Exercise Set III

Georgios Manos
csd4333

27 October 2022

Note that all code files are related to the 2nd exercise, except for the run_lr_penalty_experiments.py. This and run_analysis.py are the 2 runnable python scripts to generate all the requested results.

1 Logistic Regression (Theoretical)

Assume that we have a binary classification problem and the data are linearly separable, centered around $x = 0$. We also have the log likelihood function, $l(w)$:

$$l(w) = \sum_{l} y^l ln(\frac{1}{1+e^{-wx^l}}) + (1-y^l) ln(\frac{e^{-wx^l}}{1+e^{-wx^l}})$$

We know that logistic regression learning algorithm will attempt to find the w that maximizes $l(w)$. Since the data are linearly separable, this means that we can split $l(w)$ into 2 parts, for $x^l > 0$ and $x^l < 0$, associated with class $y = 1$ and $y = 0$ respectively.

Starting off with class $y = 1$, we need to find w that maximizes $ln(\frac{1}{1+e^{-wx^l}})$. But knowing that $x > 0$, $\forall x$ associated with class $y = 1$, $ln(\frac{1}{1+e^{-wx^l}})$ is a monotonically increasing function w.r.t. w. This means that as much as w is increasing, the more that expression increases, so the maximum is when $w \to \infty$:

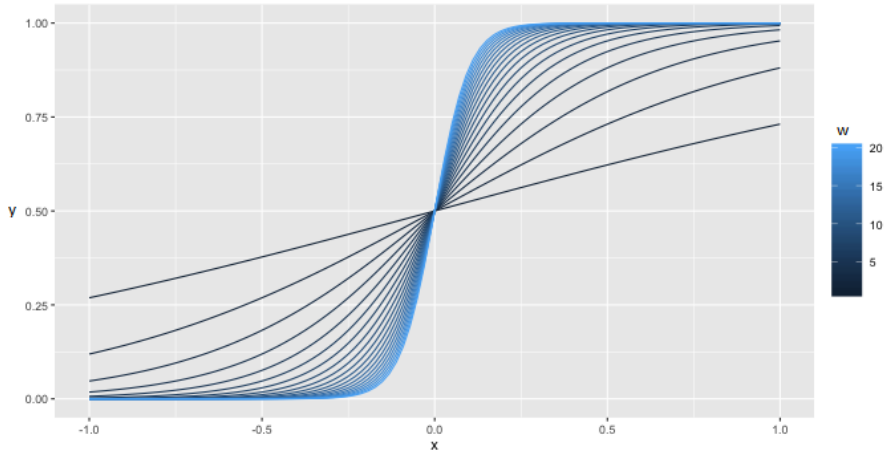
$$\lim_{w \to \infty} ln(\frac{1}{1+e^{-wx}}) \underset{x>0}{=} 0$$

Respectively, for class $y = 0$, $ln(\frac{e^{-wx^l}}{1+e^{-wx^l}}) = ln(1 - \frac{1}{1+e^{-wx^l}})$ is also a monotonically increasing function w.r.t. w, given that $x < 0$. So again:

$$\lim_{w \to \infty} ln(1 - \frac{1}{1 + e^{-wx}}) \underset{x<0}{=} 0$$

Essentially since both functions are monotonically increasing, it means that as the algorithm will attempt to find the w that maximizes $l(w)$, w will always be going up as $l(w)$ will also be monotonically increasing to ∞.

Geometrically, what essentially happens is that the algorithm will attempt to approximate $x = 0$ as decision boundary line. To interpret that line as a function expressed as $y = wx$, w will be $\to \infty$:



2 Evaluation of Classifiers (Programming)

I implemented this part of the assignment using my own implementation of Naive Bayes, adjusting it so it can accept Mixed input. The problem however was to create some assumptions on our input and the different values that categorical variables may take. This had to happen before our train and test split, although snooping into the test data is not a good practice in general, we had to do it this time around just in case some X values are left out of the training set. Same applies to the one hot encoding for logistic regression.

To comment on the results, lets start off with the plots.

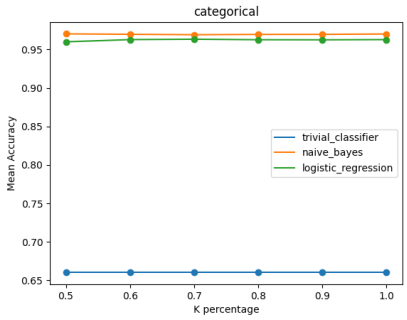


Figure 1: Average performance over different K percentages for categorical variables input

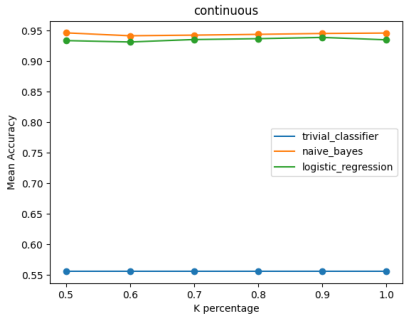


Figure 2: Average performance over different K percentages for continuous variables input

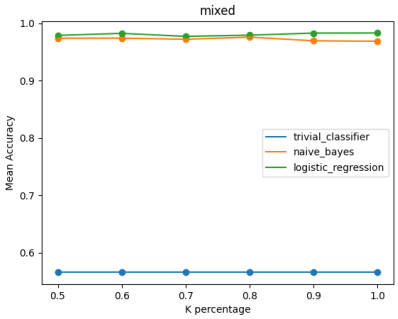


Figure 3: Average performance over different K percentages for mixed variables input

In theory, we expect that Naive Bayes as a generative classifier will have a higher asymptotic error as the number of training examples become large than the Logistic Regression as the discriminative model, but the generative model should approach its asymptotic error much faster than the discriminative model*. So although we would expect for Naive Bayes' performance to not be impacted much by the training set size, we should be seeing an increase on logistic regression. Finally, bias vs variance tradeoff comes again in hand. We know that Naive Bayes has a higher bias but lower variance compared to logistic regression. So depending on the dataset, we should see the change different performances.

In practice, we observe no such phenomenon. Both classifiers on all 3 datasets seem to perform much better to the trivial classifiers, in terms of mean classification accuracy on the test set, and more specifically Logistic Regression outperforms Naive Bayes only on the mixed dataset. They also appear to have almost the same performance over all K values, with a slight increase as K increases, except maybe in the continuous plot. Again, for more robust results we could increase the number of repeats, as my multiple runs tend to show a little different results, but the pattern of NB outperforming LR in categorical and continuous dataset while both classifiers having an almost constant performance does not change.

Both Naive Bayes and Linear Regression finally use linear decision surfaces. Given the high mean accuracy of both our classifiers, we could say that our dataset should follow an almost linear pattern, so the results of them having similar performance with smaller training dataset, given the random sampling on train-test split, the results are to be expected. Note that Logistic regression for this experiment used L2 regularization with 1.0 strength, helping generalize our model better and making sure we don't get a complicated and overfit model to our training set. We will talk more on Regularization on the next section.

*On Discriminative vs. Generative classifiers: A comparison of logistic regression and naive Bayes, Andrew Y. Ng & Michael I. Jordan

3 Regularization of Logistic Regression

First off, starting with the requested plot, I made 2 different plots as on the 2nd I included the bias term as well. Please note that those plots are actually for inverse λ values as defined by hyperparamter C in logistic regression from SKLearn (where $C = \frac{1}{\lambda}$). I noticed it a bit too late, so I'll start my analysis from here, its worth seeing what happens for both really big and really small regularization strength values.

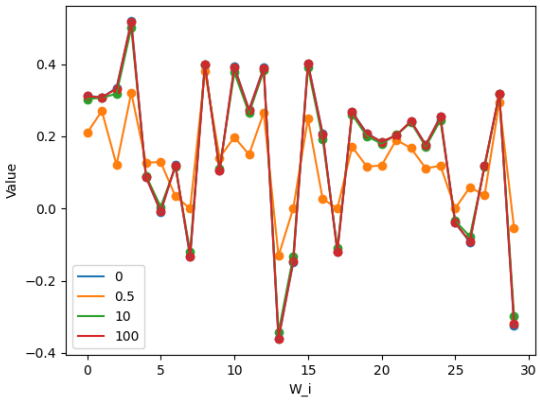


Figure 4: Logistic regression coefficient values with increasing C parameter

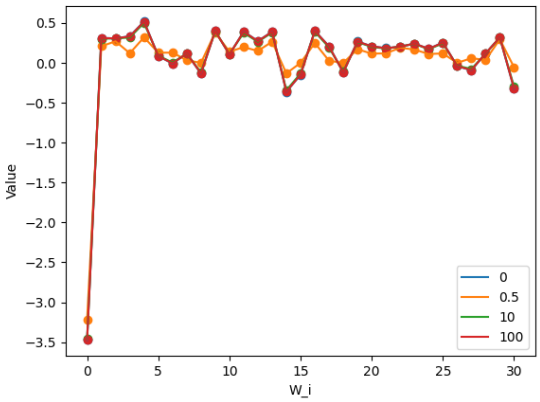


Figure 5: Logistic regression coefficient values with increasing C parameter (including bias term)

The lines are pretty similar in general, and especially for regularization strength values 0, 10 and 100, where they have almost 100% overlap, while the orange one is quite different.

Regularization in general is a technique used to help maintain a simple, more generalizable model and prevent overfitting to the training set, as it acts as a complexity penalty for extreme parameters. A high value of C usually tells the model to give high weight to the training data and a lower weight to the complexity penalty, and this is exactly our case when comparing no regularization to really high regularization strength values (hence really small C value), we get the exact same model which "trusts" the training data and (probably over-)fits to it. On the other hand, the orange model with small C value gives more weight to the complexity penalty applied from the L1 regularization at the expanse of fitting to the training data.

As I mentioned above, I ran this experiment for different C values, not λ. If we do the inverse experiment, increasing the λ value, hence C values in $[0, 2, 0.1, 0.01]$, we get different results as the actual regularization strength increases a lot:

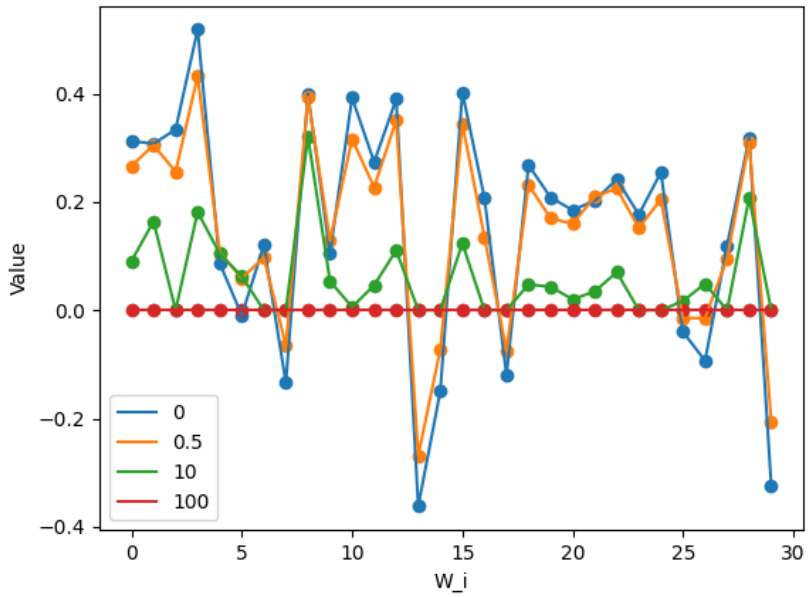


Figure 6: Logistic regression coefficient values with increasing regularization strength

On that second plot, as the regularization strength increases we get even simpler models, with smaller variance while also underfitting the data for really big values. We can see that as the regularization strength increases, the weight variance is decreasing, eventually leading to 0 as all the weights become 0 in the extreme case of really big regularization strength. On the other hand, the model with the biggest weight variance is the blue one, which has no regularization, reaching bigger weight values and probably overfitting to the training data.

4 Appendix

Figure 7: Much luv to my TA