Data Mining

Assignment2

Spyridon Roumpis

181004877

Exercise 0: What are the new correctly classified instances, and the new confusion matrix? Briefly explain the reason behind your observation

Logistic Regression/Maximum Entropy classifier with Test Options set to Cross-validation and Folds=10.

a. *Logistic Regression* classifier

Correctly Classified Instances	752	75.2 %
Incorrectly Classified Instances	248	24.8 %
Kappa statistic	0.375	
Mean absolute error	0.3098	
Root mean squared error	0.4087	
Relative absolute error	73.727 %	
Root relative squared error	89.1751 %	
Total Number of Instances	1000	

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class	
0.864	0.51	0.798	0.864	0.83	0.785	good	
0.49	0.136	0.607	0.49	0.542	0.785	bad	
Weighted Avg. 0.752	0.398	0.741	0.752	0.744	0.785		

=== Confusion Matrix ===

a	b	<-- classified as
605	95	a = good
153	147	b = bad

For the logistic Regression classifier, it can be seen that the accuracy is 75.2% while the error rate is 24.8%. If we compare the metrics with the KNN classifier it seems like that this classifier (logistic regression) has a better accuracy and lower misclassification rate.

b. KNN Classifier

- ***KNN* classifier. The default setting is K=1**

Correctly Classified Instances 720 72 %
Incorrectly Classified Instances 280 28 %
Kappa statistic 0.3243
Mean absolute error 0.2805
Root mean squared error 0.5286
Relative absolute error 66.7546 %
Root relative squared error 115.3422 %
Total Number of Instances 1000

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.81	0.49	0.794	0.81	0.802	0.66	good
	0.51	0.19	0.535	0.51	0.522	0.66	bad
Weighted Avg.	0.72	0.4	0.716	0.72	0.718	0.66	

=== Confusion Matrix ===

a	b	<-- classified as
567	133	a = good
147	153	b = bad

- ***KNN* classifier. The default setting is K=4**

Correctly Classified Instances 745 74.5 %
Incorrectly Classified Instances 255 25.5 %
Kappa statistic 0.2698
Mean absolute error 0.3151
Root mean squared error 0.4338
Relative absolute error 74.9928 %
Root relative squared error 94.6713 %
Total Number of Instances 1000

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class

	0.944	0.72	0.754	0.944	0.838	0.718 good
	0.28	0.056	0.683	0.28	0.397	0.718 bad
Weighted Avg.	0.745	0.521	0.732	0.745	0.706	0.718

=== Confusion Matrix ===

a	b	<-- classified as
661	39	a = good
216	84	b = bad

- ***KNN* classifier. The default setting is K=999**

Correctly Classified Instances	700 70 %
Incorrectly Classified Instances	300 30 %
Kappa statistic	0
Mean absolute error	0.42
Root mean squared error	0.4583
Relative absolute error	99.9578 %
Root relative squared error	100 %
Total Number of Instances	1000

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	1	1	0.7	1	0.824	0.5	good
	0	0	0	0	0	0.5	bad
Weighted Avg.	0.7	0.7	0.49	0.7	0.576	0.5	

=== Confusion Matrix ===

a	b	<-- classified as
700	0	a = good
300	0	b = bad

Comparing the accuracy and the error rate for the KNN classifier for the different values of K, it can be seen that the best accuracy – 74.5% and the lowest error rate– 24.5% is for k=4 while for k=1 and accuracy and error rate accounted for 72% and 28% respectively. For k=999 we had the worst accuracy – 70% and the lowest misclassification rate – 30%.

Let's analyse the metrics for k=999,

TP (the number of correct positive predictions) =70

Sensitivity: 700/1000= 0.7

TN (the number of correct negative predictions) = 0

Specificity: cannot be calculated as both of FP and TP are zero.

It can be seen that the TN and the FN are both 100% accurate, so we can now see that when an attribute is characterised as FN automatically, we know the TP. While the classifiers accuracy and the other metrics seem to be the worst, actually by knowing that TN and FN will always be 100% accurate we always know if a sample is TN or TP.

Exercise 1: Try (by varying the values of weights in the code), and describe the effect of the following actions on the classifier (with a brief explanation):

- **Changing the first weight dimension, i.e., w0**

  Changing the first weight dimension w0- intercept affects the 'position' of the actual boundary. How left or right it will be in the classifier to separate our data set. When w0 is zero the boundary is set somewhere in the middle of the blue and the red class. The lower the value of w0<0, the bigger the blue class becomes, so the boundary moves towards the red class. Exactly the opposite happens when w0>0 and has a big value, the higher the value the bigger the red class becomes, with the boundary moving towards the blue class

- **Changing the second two parameters, w1, w2**

  w1 is the weight of attribute 1 and w2 is the weight of attribute 2, both are responsible for the rotation of our boundary. How the boundary will start rotating to the right or to the left depends on the values of w1 and w2

- **negating all the weights**

  Negating all the weights will make the classifier behave wrongly. It will think that the red attributes belong to the blue class and the blue attributes to the red class. On the left side will be the red class while on the right, the blue, which is the opposite of how it should be.

- **scaling the second two weights up or down (e.g. dividing or multiplying both w1 and w2 by 10)**

  Scaling w1 and w2 by 10 the values very close to the boundary will have an uncertainty about their true identity because of the noise territory. Dividing them by 10 will cause more attributes to have probability close to 0.5 near to the boundary, something that does not happen for highest value of w1 and w2 as the probability will tend to be 1 or 0.

Exercise 1: (a) How many times did your loop execute? (b) Report your classifier weights that first get you at least 75% train accuracy.

(a) My loop executed 8000 times.
(b) a new best found = 54.13629 --- accuracy: 0.75 --- weights = [-0.579 1.000 0.789]

Exercise 2: (a) Notice that we used the negative of the gradient to update the weights. Explain why. (b) Given the gradient at the new point, determine (in terms of "increase" or "decrease", no need to give values), what we should do to each of w0, w1 and w2 to further improve the loss function.

(a) Our goal with gradient descent is to find the optimal weights: minimize the loss function we've defined for the model, and practically if you try to run the classifier without negating the gradient to update the weights the accuracy and the performance is very poor.

(b) To see how we will improve the loss function, we ran some tests increasing and decreasing the w0, w1 and w2. We should increase w1 and w2 and keep w0 as it is, zero.

Exercise 3: Suppose we wanted to turn this step into an algorithm that sequentially takes such steps to get to the optimal solution. What is the effect of "step-size"? Your answer should be in terms of the trade-offs, i.e., pros and cos, in choosing a big value for step-size versus a small value for step-size.

Below it can be seen that having small values of the step-size, between (0,1), it improves the performance of our model, while having values higher than one it's not performing that well, the loss function goes really high and the accuracy drops.

step_size = 0
loss function (negative-log-likelihood): 38.00
gradient of the loss function at this new point: [-10.5707144681 0.2991945346 0.1011619691]
Train accuracy: 0.81

step_size = 0.1
loss function (negative-log-likelihood): 31.00
gradient of the loss function at this new point: [-2.3417195399 0.4844386443 0.2279333400]
Train accuracy: 0.82

step_size = 0.5
loss function (negative-log-likelihood): 34.22
gradient of the loss function at this new point: [8.0509116767 0.1573034140 0.1413389576]
Train accuracy: 0.84

step_size = 1
loss function (negative-log-likelihood): 185.58
gradient of the loss function at this new point: [-39.0177512575 -9.9569884271 -4.8678732933]
Train accuracy: 0.60

Exercise 4: Answer this question either by doing a bit of research, or looking at some source codes, or thinking about them yourself: (a) Argue why choosing a constant value for step_size, whether big or small, is not a good idea. A better idea should be to change the value of step-size along the way. Explain whether the general rule should be to decrease the value of step-size or increase it. (b) Come up with the stopping rule of the gradient descent algorithm which was described from a high level at the start of this section. That is, come up with (at least two) rules that if either one of them is reached, the algorithm should stop and report the results.

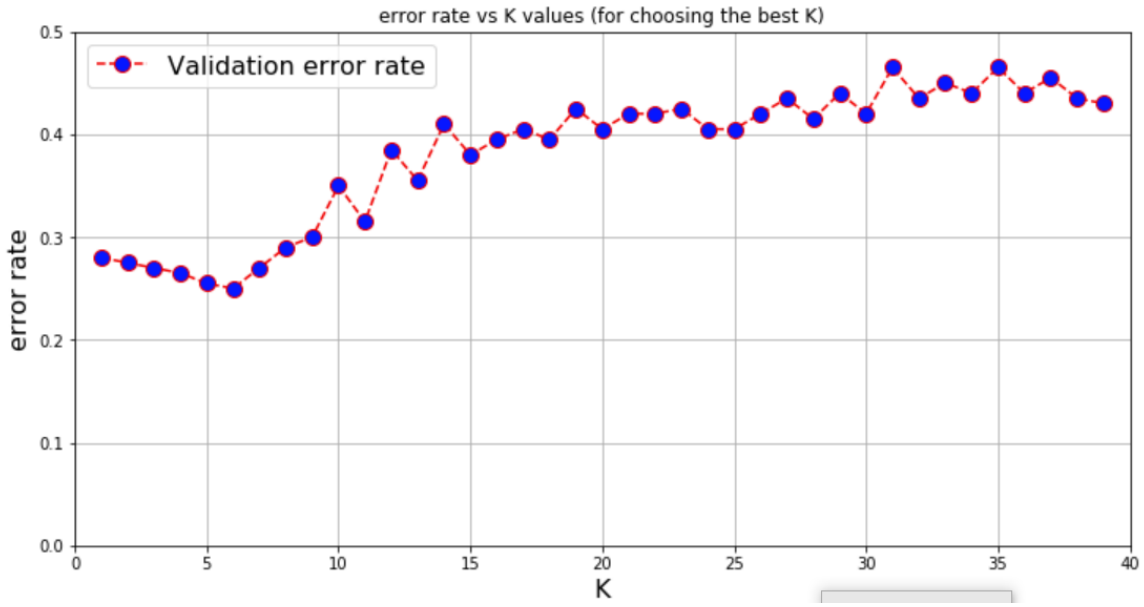
(a) By using a small step size, we will make small progress every time, which means we are acting carefully and being consistent. By choosing a large step we attempt to descend faster although this may not pay off. Every model is different so I do not think it can be a general rule for the size of the step, to find the best step for your model you should try and test different values of the step.
(b) We can define a threshold and check it's time if our improvement is better or not than this threshold, the iteration will stop when the improvement drops below the threshold.

Exercise 5: Provide one advantage and one disadvantage of using exhaustive search for optimisation, especially in the context of training a model in machine learning.

The developed exhaustive search method performs an iterative search of all values combinations and determines which combination performs best; the main advantage of this optimization method is that it will always find the optimal solution for given discrete space.

One of its disadvantages is that exhaustive search can be computationally costly and very slow as it might need to do thousands of iterations.

Exercise 6: (a) What is the best value for K? Explain how you got to this solution. (b) Interpret the main trends in the plot. In partiular, specify which parts correspond to under-fitting and which part with over-fitting.



It can be seen that the lowest error rate is when k=6, so we can say that 6 is the best value for k, although for k=5 the error rate is also very low and quite close to the error rate for k=6.

We can also assume for values lower than 5, the model is under-fitting, while for values over 6, or maybe 7, the model starts overfitting.

Exercise 7: Given what you learned from this entire lab(!), provide one advantage and one disadvantage of using MaxEnt vs KNN.

The main advantage of using MaxEnt against KNN is that the model uses search-based optimization to find weights for the features that maximize the likelihood of the training data, selecting the one which has the largest entropy. One disadvantage is that datasets like the one in this example are not linear separable so KNN performs significantly better.