Work 3: Instance-Based Learning Algorithms

Introduction to Machine Learning, 2021-2022

Augusto Moran Calderon Santiago del Rey Juárez
Yazmina Zurita Martel

December 2021

1 Datasets

In order to test the different algorithms implemented in this project, we had to choose two different datasets to be able to compare the results. Originally, we wanted to use big datasets such as the Pen-based or the Adult from the UCI Machine Learning Repository [1]. However, due to the high number of samples of these datasets and the computational cost of some of the algorithms, we decided to use smaller datasets to be able to run the different configurations in a more reasonable time. Also, we expect that by using 10-fold cross-validation we can compensate for the reduction in data. Thus, the datasets used in this project are the Vowel, which is a mixed dataset, and the TAO-Grid, which is a numerical dataset.

2 IBL implementation

In this section, we describe the implementations of the IB1, IB2 and IB3 algorithms.

The first thing we decided on was how to handle categorical features. One option was to apply a technique such as One Hot Encoding to transform these features into numerical ones. Another way of solving this problem was to separate the features into two groups (i.e. numerical and categorical) and compute the similarities separately and then join the results. Since for the first two projects we applied the first option, we wanted to try a different approach. Thus, we decided to implement our IBL algorithm with the second option. Another reason to choose this approach was the space reduction that it offered in front of other approaches that can increment drastically the number of features used. The next matter we had to deal with was how to treat noisy data since IBL algorithms are able to manage it. At first, we wanted to implement the algorithms in such a way that they could handle noise (i.e. missing values) without previous pre-processing. However, as time passed we were unable to find an optimal solution for this problem in numerical features. For this reason, we finally decided to replace missing values with the mean of the feature. For categorical features, it was much easier to solve this problem since it was a simple comparison between categories.

Once we had this matter decided, we proceeded to implement each one of the IBL algorithms using the descriptions provided by Aha *et al.* in their work [2].

2.1 IB1

The IB1 algorithm is the simplest version of the three. It is identical to the Nearest Neighbour algorithm, except that it normalizes its attributes ranges to [0, 1], processes instances incrementally, and has a simple policy for tolerating missing values. In Figure 1 we can see the pseudo-code for this algorithm.

Table 1. The IB1 algorithm (*CD* = Concept Description).

CD ← ∅
for each $x \in$ Training Set **do**
1. **for each** $y \in CD$ **do**
   Sim[y] ← Similarity(x, y)
2. y_{max} ← some $y \in CD$ with maximal Sim[y]
3. **if** class(x) = class(y_{max})
   then classification ← **correct**
   else classification ← **incorrect**
4. $CD \leftarrow CD \cup \{x\}$

Figure 1: IB1 algorithm extracted from [2].

The critical part of this algorithm is the computation of the similarities between the current sample and the samples in the Contend Descriptor. To compute this similarity, we have used the Euclidean distance. However, since we decided to use raw categorical data instead of discretizing it, we had to compute the similarity in two steps.

In the first step, we computed the difference for the numerical features and the difference for the categorical ones. In the latter, we considered three situations when calculating the difference: (i) if one of the two values is missing then the difference is 1, (ii) if the two values are equal then the difference is 0, (iii) if the two values are different then the difference is 1.

The second step was to join the two measures simply by concatenating them and then apply the rest of the Euclidean distance (see Eq. 1).

$$D(x, y) = \sqrt{\sum_{i=0}^{m}(x_i - y_i)^2} \tag{1}$$

In our implementation, the IB1 works in two steps. First, we initialize the content descriptor (CD) by running the algorithm with the training samples. In this way, the CD is filled incrementally. Then, once the CD is filled with training samples, we run the algorithm again with the test set and measure the accuracy of the predictions.

2.2 IB2

The IB2 algorithm is an improved version of the IB1 which seeks to improve memory and computation efficiency. This algorithm is based on the idea that it is not necessary to use all data points for classification. Thus, it will only save those examples that are misclassified by the current CD. In Figure 2 we can see the pseudo-code for this algorithm.

Table 2. The IB2 algorithm (*CD* = Concept Description).

```
CD ← ∅
for each x ∈ Training Set do
    1. for each y ∈ CD do
       Sim[y] ← Similarity(x, y)
    2. y_max ← some y ∈ CD with maximal Sim[y]
    3. if class(x) = class(y_max)
       then classification ← correct
       else
           3.1 classification ← incorrect
           3.2 CD ← CD ∪ {x}
```

Figure 2: IB2 algorithm extracted from [2].

Our implementation of this algorithm has been identical to the one described in the previous section, with the difference that now the CD is only updated with misclassified examples, as we have already explained.

2.3 IB3

The last of the standard IBL algorithms implemented has been the IB3, which is again an improvement from the IB2. In this case, the algorithm saves a counter for

the number of times an example participated in correct and incorrect classifications. Then, by using two predefined thresholds it is able to filter noisy examples and discard instances that do not perform well. In Figure 3 we can see the pseudo-code for this algorithm.

Table 5. The IB3 algorithm (*CD* = Concept Description).

```
CD ← ∅
for each x in Training Set do
   1. for each y ∈ CD do
      Sim[y] ← Similarity(x, y)
   2. if ∃{y ∈ CD| acceptable(y)}
      then y_max ← some acceptable y ∈ Cd with maximal Sim[y]
      else
         2.1 i ← a randomly-selected value in [1,|CD|]
         2.2 y_max ← some y ∈ CD that is the i-th most similar instance to x
   3. if class(x) ≠ class(y_max)
      then classification ← correct
      else
         3.1 classification ← incorrect
         3.2 CD ← CD ∪ {x}
   4. for each y in CD do
      if Sim[y] ≥ Sim[y_max]
      then
         4.1 Update y's classification record
         4.2 if y's record is significantly poor
             then CD ← C − {y}
```

Figure 3: IB3 algorithm extracted from [2].

Our implementation of the algorithm is very similar to the one used for the IBL 2. The main differences are that it targets the problem of keeping noisy instances of the training data. To fix this problem Aha *et al.* developed some new concepts that describe the instance with respect to the instances in the concept descriptor Aha *et al.*, Aha. These are acceptable instances and significantly poor classification records.

To know if an instance of the CD is acceptable or poorly classified, we need to obtain the confidence interval on the accuracy and the frequency resulting in 4 values upper and lower bounds. There are 2 conditions to determine the "state" of an instance:

1. $LowerBound_{ACCURACY} > UpperBound_{FREQUENCY}$
2. $UpperBound_{ACCURACY} < LowerBound_{FREQUENCY}$

After evaluating the first condition, if it is true then an instance is considered to be acceptable. So at each iteration, we pick the nearest acceptable instance on the CD. To calculate the significantly poor instances we evaluate the second condition.

Both bounds are calculated using the following formula:

$$\frac{p + z^2/2n \pm z\sqrt{\frac{p(1-p)}{n} + \frac{z^2}{4n^2}}}{1 + z^2/n}$$

Where the parameters alter depending on if we are computing the bounds for the accuracy or frequency the only one that stays the same is the "z" which is the confidence (0.9 for acceptance and 0.7 for dropping).

When computing the bounds for the accuracy n is the number of times it was least as close to the training sample of that iteration as the nearest instance was, p is the number of times the instance class matched the nearest instance's class divided by n.

When computing the bounds for the frequency n is the number of previously processed instances, and p is the portion of instances so far that are of this class.

Given the formula and the conditions we get, at every iteration, we work with the next training sample, and after evaluating if it should be included in the CD or not, we update the CD. At the end of the algorithm, we remove all non-acceptable instances of CD.

Given the fact that IB3 stores a smaller number of instances, the concept descriptor is reduced thus saving space and because it is robust to noisy instances it can assure a slightly better accuracy than IB2. The downside is that the computational expenses and time allocated to perform it can be impractical for lazy learning implementation.

2.4 Results

After running the three algorithms on the two selected datasets, we obtained the results shown in Table 1. To measure the difference between the different algorithms

we use two metrics: average accuracy and average execution time. As we can see in the table, the most accurate algorithm is the IB1 and the worst is the IB2. This result is expected due to the IB1 having a greater knowledge base (i.e. more examples in the CD) than the IB2. This increases the number of near examples when doing a query which makes it easier for the algorithm to correctly determine the class. On the other side, if we look at the execution time the IB2 is the clear winner, with times under 1 second. This tremendous reduction in the execution time comes as a consequence of the reduced number of examples that are added to the CD. Lastly, we observe that the IB3 does not suppose a great improvement from the IB2, in this case. Moreover, its computational cost is too high in comparison to the increase in accuracy it brings. Thus, we consider IB2 a better choice than IB3.

From these results, we can conclude that IB1 is the best algorithm among the three in terms of accuracy.

	IB1	IB2	IB3
Avg. accuracy	0.950	0.933	0.942
Avg. execution time (s)	3.063	0.255	11.360

(a) TAO-Grid dataset.

	IB1	IB2	IB3
Avg. accuracy	0.973	0.935	0.938
Avg. execution time (s)	1.255	0.270	4.192

(b) Vowel dataset.

Table 1: Comparison of the different IBL algorithms.

3 K-IBL

In this section, we will describe our implementation of the K-IBL algorithm. This variant of the IBL algorithm uses the idea of K-Nearest Neighbors to choose the most appropriate class when doing a prediction. Since we tried three different IBL algorithms (see Section 2), we choose to use IB1 as the base since it was the one that gave us the best results.

The main difference between the K-IBL and the IB1 is the use of different distance metrics and the incorporation of voting policies to choose the sample's class. Besides these two additions, the implementation of the IB1 is the same as the one described in Section 2.1.

3.1 Distance measures

	Euclidean	Manhattan	Canberra
$d(x, y)$	$\sqrt{\sum_{a=1}^{m}(x_a - y_a)^2}$	$\sum_{a=1}^{m}\lvert x_a - y_a\rvert$	$\sum_{a=1}^{m}\frac{\lvert x_a - y_a\rvert}{\lvert x_a + y_a\rvert}$

Table 2: Equation of three out of the four distance metrics tested.

To determine the distance between two instances, we defined four different similarity measures: the Euclidean, Manhattan and Canberra distances and the Heterogeneous Value Difference Metric (HVDM). Given two instances $x, y \in \mathbb{R}^m$, we defined their distance as shown in Table 2. In the case of the HDVM metric, we used the same formulation as in [4]:

$$HVDM(x, y) = \sqrt{\sum_{a=1}^{m} {d_a}^2(x_a, y_a)}$$

where

$$d_a(x, y) = \begin{cases} 1 & \text{if } x \text{ or } y \text{ is unknown} \\ normalized\_vdm_a(x, y) & \text{if } a \text{ is categorical} \\ normalized\_diff_a(x, y) & \text{if } a \text{ is numerical} \end{cases}$$

None of these metrics are able to handle categorical data except for the HVDM. In these cases, we treated nominal values as if they were one-hot encoded:

$$|x - y| = \begin{cases} 1 & \text{if } x_i \neq y_i \\ 0 & \text{if } x_i = y_i \end{cases} \quad \text{and} \quad (x + y) = \begin{cases} 1 & \text{if } x_i \neq y_i \\ 2 & \text{if } x_i = y_i \end{cases}$$

This allows us to apply all the aforementioned metrics on nominal values but without increasing the feature space as performing one-hot encoding would.

3.2 Voting policies

We analysed three different voting policies. The first of them returns the most voted class. If classes are tied, we select the one that has the nearest instance to the example considered. We based this decision on thinking that the closer the samples, the more similar they should be. Thus, the closest class should take precedence in a tie.

Another one is the Borda count. It ranks the k options in descending order according to the number of votes and assigns them a decreasing amount of points ($k-1$, $k-2$, ..., 0). The winner is the one with the highest number of points. In the case of a tie, we follow the same procedure as in the previous policy.

The last one is the Modified Plurality policy. It also computes the most voted option but if classes are tied, it removes the last s nearest neighbours and recalculates the class. We decided to set s to 1 so the number of voters removed to break the tie is as low as possible. This makes the method more robust against noise.

3.3 Statistical analysis

Besides measuring the accuracy and efficiency of different K-IBL configurations, we performed a statistical analysis of the results. The goal is to know whether the differences in accuracy between algorithms are statistically significant or not. We needed to compare 36 and 3 settings in the case of the K-IBL and selection K-IBL respectively. The tests we found for comparing such a number of classifiers against each other on a set of repeated measures are the ANOVA and Friedman tests. The measures are the accuracies per fold in our case. However, the ANOVA assumes properties in the variables compared (normality and sphericity) that can not be guaranteed when working with learning algorithms. Therefore, we implemented its non-parametric equivalent, the Friedman test.

We first defined the Friedman statistic as in [5] considering r_{ij} to be the rank of the j-th of k algorithms on the i-th of N data sets:

$$\chi^2{}_F = \frac{12n}{k(k+1)} \sum_{j=1}^{k} \left(\frac{1}{n} \left(\sum_{i=1}^{N} r_{ij} \right) - \frac{k+1}{2} \right)^2$$

If $k > 4$ or $N > 15$ is large, we can assume a χ^2 distribution. However, this was not always the case so we chose a more suitable form of the statistic ([6]):

$$F_F = \frac{(N-1)\,\chi^2{}_F}{N\,(k-1) - \chi^2{}_F}$$

F_F follows a F-distribution with $k-1$ and $(k-1)(N-1)$ degrees of freedom. If F_F is higher than the critical value of the F-distribution given a significance level, it means that the algorithms compared are statistically different.

We extracted the critical values using a method of the SciPy Python library [7]. To know which of the algorithms differ, we performed a *post-hoc* Nemenyi test. It compared the difference in the rank means between algorithms with the critical distance ([6]):

$$CD = q_\alpha \sqrt{\frac{k\,(k+1)}{6N}}$$

The values for q_α are available in [8] as a CSV table.

3.4 K-IBL performance

After implementing all the aforementioned metrics and voting policies, we proceeded to run the K-IBL with all the possible combinations using k = 3, 5, 7 and all the distance metrics and voting policies. In total, we had 36 different combinations that were run on both datasets.

Since the amount of combinations is too great to compare them with reasonable detail, we will only discuss the best 10 configurations for each of the datasets. In addition, to better compare these results we performed a statistical analysis between the different configurations.

3.4.1 Performance

We can see the results for the TAO-Grid dataset In Table 3. The average accuracy is very similar in the ten best configurations. After applying the statistical analysis. we found that these results are not statistically different from each other as we expected ($F_F = 1.139 < 1.461$). Then, we can choose any of these configurations as the best one. Since we must select one, we will do it in terms of accuracy first and then in terms of execution times if there is a tie. Under this criterion, the best configuration for the TAO-Grid dataset is given by $K = 7$, the Euclidean distance and the most voted solution policy.

We observe that most of the results in Table 3 correspond to $k = 7$. This could indicate that the dataset is noisy and that we need to consider a higher amount of neighbours to make an accurate prediction. Also, the Canberra distance is not present in any of the best configurations. Then, this weighted version of the Manhattan distance does not seem to be a suitable metric for this dataset.

Looking at the results for the Vowel dataset in Table 4, we see that the accuracies differ more than in the TAO-Grid dataset but not by much. We found that there is statistical difference between the 36 configurations ($F_F = 8.951 > 1.461, CD = 18.080$) but not between the ten best. Nonetheless and following the same criterion as before, the best configuration for the Vowel dataset is given by $K = 3$, the Euclidean distance and the Modified Plurality policy.

Contrary to what happened in the other dataset, we get the best accuracy for small values of k. The reason could be that the groups are small, so the neighbourhood should be tight to avoid the inclusion of instances of other classes. However, we know that there are 11 classes containing 90 instances each of them, which is much higher than the k values tested. Then, we believe that the groups are very close to each other. If so, we would consider instances of other classes when classifying borderline samples when k is high. Thus, smaller values are more suitable.

Overall, we note that the average execution time using the HVDM measure is disproportionately high compared with the others. It yields a similar accuracy to the

other metrics but it is a thousand times slower. Also, the most voted solution and the Modified Plurality policy seem to be the most suitable policies in both datasets, although this depends on the selection of the other parameters as well.

Configuration			Avg. accuracy	Avg. execution time (s)
K	Measure	Policy		
7	Euclidean	Most voted	0.9566	3.0818
7	Euclidean	Mod plurality	0.9566	3.0954
7	HVDM	Mod plurality	0.9566	118.1410
7	HVDM	Most voted	0.9566	120.7470
7	HVDM	Borda count	0.9555	118.0774
7	Manhattan	Mod plurality	0.9550	2.4639
7	Manhattan	Most voted	0.9544	2.4632
7	Euclidean	Borda count	0.9544	3.0825
7	Manhattan	Borda count	0.9518	2.4845
3	Euclidean	Most voted	0.9513	3.0642

Table 3: Top 10 configurations of the K-IBL algorithms on the TAO-Grid dataset.

Configuration			Avg. accuracy	Avg. execution time (s)
K	Measure	Policy		
3	Euclidean	Mod plurality	0.9576	1.2215
3	Manhattan	Mod plurality	0.9566	1.1129
3	Manhattan	Most voted	0.9545	1.0979
3	Euclidean	Most voted	0.9535	1.2404
3	Euclidean	Borda count	0.9525	1.2479
3	Manhattan	Borda count	0.9505	1.1264
3	HVDM	Mod plurality	0.9414	1034.9291
3	HVDM	Most voted	0.9414	1061.5125
3	Canberra	Most voted	0.9394	2.7004
3	Canberra	Mod plurality	0.9384	2.5985

Table 4: Top 10 configurations of the K-IBL algorithms on the Vowel dataset.

3.5 Selection K-IBL

The last one of the algorithms that we have implemented during this project has been the Selection K-IBL. This algorithm is an improved version of the K-IBL where we perform Feature Selection (FS) in the training phase.

At first, we wanted to implement two types of FS methods, a filter and a wrapper. However, due to time limitations, we decided to implement two filters instead, since they are usually less computationally expensive than wrappers.

There exist plenty of filter methods implemented in Python [9], [10]. In our implementation, we decided to use `SelectKBest` and `VarianceThreshold` from the Scikit-learn Python library [11] as the filter methods. For the `SelectKBest` filter we used mutual information classifier as a scoring function since it performed better in mixed datasets and we choose to keep the best 10 features. In the `VarianceThreshold`, we used a threshold of 0.03, since it seemed to perform well in both datasets.

3.5.1 Performance

To assess the performance of the Selection K-IBL and see if the FS step improved the results, we run the algorithm using the same parameters as the best K-IBL for each of the datasets.

In Table 5 we can see the results obtained for the TAO-Grid dataset. As we can see both selection methods obtain the same results as the K-IBL. This is due to the dataset containing only two features. In consequence, the FS does not reduce the dimensionality of the dataset, which results in a standard execution of the K-IBL algorithm. Consequently, the results are not statistically different ($F_F = -15.000 < 2.624$).

Configuration				Avg. accuracy	Avg. execution time (s)
K	Measure	Policy	Selection method		
7	Euclidean	Most voted	K-best	0.9566	3.1533
7	Euclidean	Most voted	Variance threshold	0.9566	3.1787

Table 5: Performance of the Selection K-IBL algorithm on the TAO-Grid dataset.

The results of the Selection K-IBL on the Vowel dataset can be seen in Table 6. In this case, the two selection methods obtain different results. For the K-best method, we observe a slight improvement of 0.002 on the accuracy with respect to the K-IBL. However, the Variance threshold method, loses accuracy falling to 0.9475. This is because of the nature of the dataset, which contains features with very low variance and thus it becomes very difficult to select the most important ones based only on their variance. For this reason, the K-best method, which relies on other measures, performs better in this type of dataset. Although the accuracies of the methods differ, it is not enough to be statistically significant as it was confirmed by the Friedman test ($F_F = -19.465 < 2.624$).

Configuration				Avg. accuracy	Avg. execution time (s)
K	Measure	Policy	Selection method		
3	Euclidean	Mod plurality	K-best	0.9596	1.2645
3	Euclidean	Mod plurality	Variance threshold	0.9475	1.2815

Table 6: Performance of the Selection K-IBL algorithm on the Vowel dataset.

4 Conclusions

Throughout this work, we have tried different implementations of IBL algorithms which have proven to be very useful for classification tasks since all of them got accuracies over 0.9. In particular, we consider the IB1 and their improved version K-IBL to be very good classifier models based on our results.

Regarding the K-IBL, we noticed that the most important parameter was the K used in the voting system. This is to be expected since the different distance measures should obtain similar results when getting the nearest examples. A similar thing happens with the voting policies. Thus, what really defines the quality of a prediction, is the number of examples that we take into account when deciding the final class.

Another thing we noticed is the high computational cost of some of the K-IBL configurations. The most relevant case is when we use the HDVM distance measure. With this, we spent an average of one hour adding all the training examples to the CD and around fifteen minutes to run the query with the testing examples. This should be kept in mind when using these algorithms since, in some situations, it

might be preferable to obtain faster predictions at the expense of some accuracy.

One last thing we noticed when using the Selection K-IBL, is that the impact of reducing the dimensionality of the datasets in the model accuracy was not very relevant. However, these results are not concluding since the two datasets used in this project have a low amount of features. Thus, we think that it might be worth applying this same algorithm to a much bigger dataset in order to really appreciate its effect on the accuracy and execution time.

Finally, we have seen how this type of learning algorithms (i.e. lazy learners) perform very well as time goes on. This is because of the knowledge-base they internally store. In our example, this knowledge-base is the Content Descriptor that is updated with new examples with every query. This makes IBL algorithms very good predictors in the long term since the more queries they resolve the greater their knowledge-base. Nevertheless, this implies greater computational and storage costs as time goes on, as we have already mentioned.

References

[1] D. Dua and C. Graff, *UCI machine learning repository*, 2017. [Online]. Available: http://archive.ics.uci.edu/ml.

[2] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Machine learning*, vol. 6, no. 1, pp. 37–66, 1991.

[3] D. W. Aha, "Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms," *International Journal of Man-Machine Studies*, vol. 36, no. 2, pp. 267–287, 1992.

[4] D. R. Wilson and T. R. Martinez, "Improved heterogeneous distance functions," *Journal of artificial intelligence research*, vol. 6, pp. 1–34, 1997.

[5] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *Journal of the american statistical association*, vol. 32, no. 200, p. 678, 1937.

[6] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

[7] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: 10.1038/s41592-019-0686-2.

[8] N. Kourentzes, *Critical values for the nemenyi test*, https://kourentzes.com/forecasting/2014/05/01/critical-values-for-the-nemenyi-test/, Accessed on the 25/12/2021.

[9] P. Cunningham, B. Kathirgamanathan, and S. J. Delany, *Feature selection tutorial with python examples*, 2021. arXiv: 2106.06437 [cs.LG].

[10] *1.13. Feature selection*, en. [Online]. Available: https://scikit-learn.org/stable/modules/feature_selection.html (visited on 12/29/2021).

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.