WORK 3

K-Nearest Neighbor

Group members:

Santiago Bernal, Rodrigo Arias

Implementation Details

3. Write a Python function for classifying, using a KNN algorithm, each instance from the TestMatrix using the TrainMatrix to a classifier called kNNAlgorithm(...). You decide the parameters for this classifier. Justify your implementation and add all the references you have considered for your decisions.

The function `knn()` located in `lazy.py` computes the class of a test instance, by comparing the training data, and finding the best class among the `k` closest neighbours. For our implementation, the knn(...) methods receives the following parameters:

- Training_set:
  - Is the complete training dataset for the n-fold value. This data is used to find the nearest neighbors to a testing instance by calculating the different distance functions, that is why the complete matrix is needed for this function.
- Train_nominal:
  - This parameter is the nominal data (if any) related to the training_set parameter
- Testing_instance:
  - Refers to a particular testing row. It differs from the training_set because in this case, instead of being a matrix, it is only a vector, containing exactly only one instance of data. This instance is taken from the testing matrix at the n-fold position, and then each one of the rows of the testing matrix is passed to this function. The reason for this is that we are interested in finding the nearest neighbor to each one of the instances, and then validate that it was properly classified. By doing it this way, the function is called for every instance so then we build a vector of classification results, and compare it to the testing matrix actual classes vector and determine the accuracy of the algorithm.
- Testing_nominal:
  - This parameter is the nominal data (if any) of the testing_instance parameter
- Conf:
  - This is used as a configuration for the method in order to decide: the distance function to use, decision policy and the values for k. Each combination of those three variables must be calculated for evaluating the algorithm, so this parameter would be an array with one possible combination. The possible values for each is:
    - Distance: Cosine, manhattan, euclidean and canberra. The hamming distance is used only for calculating the distance in nominal data
    - Voting policy: most common and voting policy
    - K: values for the K nearest neighbors are: 1, 3, 5 and 7
- Training_set_classes:
  - This parameter corresponds to the classes of the training set parameter mencioned before. We use this parameter for the weighing and feature selection algorithms, and also for determining the selected classification of the testing instance after using the selection policy.
- Gamma:
  - A weight value that is used for calculating distance of nominal data.
- Use_weight:
  - Boolean to decide if the algorithm should be executed as a weighted Knn algorithm or not.
- Use_feature_selection:

- Boolean to decide if the algorithm should be executed with feature selection preprocessing or not.

This method returns the predicted class for the testing instance received based on the training matrix using the specified distance, k and selection policy.

4. For the similarity function, you should consider the Hamming, Euclidean, Cosine, and another EXTRA (you decide which one) distance functions. Adapt these distances to handle all kind of attributes (i.e., numerical and categorical). Assume that the KNN algorithm returns the K most similar instances (i.e., also known as cases) from the TrainMatrix to the current_instance. The value of K will be setup in your evaluation to 1, 3, 5, and 7.

In order to measure the distance from two instances, we used different functions. The distance D(u, v) between the instances u and v is computed as follows:

$$D(u,v) = d(u, v) + \gamma \cdot d_n(u,v)$$

Where the function d is used in the numerical part of the instances, and is one of Euclidean, Cosine, Manhattan and Canberra, which are selected successively. The nominal distance is computed as the sum of the elements that are equal in both instances. The value of K was set to 1, 3, 5 and 7.

a. To decide the solution of the current_instance, you may consider using two policies: the most similar retrieved case and a voting policy.

Two policies were used in order to determine the class of a test instance. The most common policy selects the most popular class in the k neighbours, without taking into account the distance. The voting policy, let each neighbour vote for his class with a weight defined as:

$$w(u, v) = D(u, v)^{-2}$$

After the vote, the class with the maximum value reached in the process is the one selected. The distance depends on the function selected as previously described.

b. For evaluating the performance of the KNN algorithm, we will use the percentage of correctly classified instances. To this end, at least, you should store the number of cases correctly classified, the number of cases incorrectly classified. This information will be used for the evaluation of the algorithm. You can store your results in a memory data structure or in a file. Keep in mind that you need to compute the average accuracy over the 10-fold cross-validation sets.

In order to evaluate the results, a table including the 10 best results based on the accuracy is generated for each dataset.

```
Dataset: data/autos
Using weights: False
Using feature selection: False
Sorting by time: False

  k  select           distance      mean %     time ms
---  ---------------  ----------  --------    --------
  1  most_common      cosine         96.36      24.93
  1  weighted_voting  cosine         95.06      24.27
  3  most_common      cosine         91.70      26.36
```

```
  3  weighted_voting  cosine          91.50      23.99
  5  most_common      cosine          91.44      25.89
  5  weighted_voting  cosine          91.28      24.50
  7  most_common      cosine          91.16      26.44
  7  weighted_voting  cosine          91.00      25.15
  1  most_common      manhattan       91.00     190.59
  1  weighted_voting  manhattan       90.81     189.00



Dataset: data/hepatitis
Using weights: False
Using feature selection: False
Sorting by time: False


  k  select           distance      mean %    time ms
---  ---------------  ----------  --------  ---------
  1  weighted_voting  manhattan      95.83      20.62
  1  most_common      manhattan      95.40      17.90
  3  weighted_voting  manhattan      93.84      18.88
  3  most_common      manhattan      93.81      19.02
  5  weighted_voting  manhattan      92.73      20.99
  5  most_common      manhattan      92.70      18.34
  7  weighted_voting  manhattan      92.59      18.55
  7  most_common      manhattan      86.36      19.81
  1  weighted_voting  cosine         92.07      52.64
  1  most_common      cosine         90.70      53.44
```

Figure 1: 10 best results of knn algorithm on the autos and hepatitis datasets

5. Modify the kNN algorithm so that it includes a weighted similarity, you will call this algorithm as weightedKNNalgorithm(...). The weights will be extracted using weighting or feature selection algorithms.

For the weighted calculations of the Knn algorithm, we used the same knn(...) function mentioned before, we setup a parameter as a flag to decide whether the algorithm would use weights or not. In the case that it does, we use the Sklearn feature selection algorithm “SelectKBest” using the training set and classes in order to get the weights to use. We apply the weights to the training set and testing instance and then the rest of the algorithm would go the same as the norman knn. The reason we chose this method for calculating the weights, was to effectively obtain the importance of each feature in the training set, and we obtain that with the “scores_” attribute that the method has.

b. Analyze the results of the weightedKNNalgorithm in front of the previous kNNAlgorithm implementation. To do it, setup both algorithms with the best combination obtained in your previous analysis. In this case, you will analyze your results in terms of classification accuracy.

Results when using weights in the Knn algorithm:

```
Dataset: data/autos
Using weights: True
Using feature selection: False
Sorting by time: False
```

```
  k  select           distance      mean %    time ms
---  ---------------  ----------  --------  ---------
  1  weighted_voting  euclidean      96.98      42.32
  1  most_common      euclidean      95.77      41.26
  3  weighted_voting  euclidean      93.84      39.70
  3  most_common      euclidean      93.25      39.13
  5  weighted_voting  euclidean      93.25      39.76
  5  most_common      euclidean      91.95      39.74
  7  weighted_voting  euclidean      91.95      41.61
  7  most_common      euclidean      91.73      40.42
  1  weighted_voting  manhattan      91.40      80.69
  1  most_common      manhattan      91.21      81.35



Dataset: data/hepatitis
Using weights: True
Using feature selection: False
Sorting by time: False


  k  select           distance      mean %    time ms
---  ---------------  ----------  --------  ---------
  1  most_common      cosine         93.85      62.13
  1  weighted_voting  cosine         93.85      63.39
  3  most_common      cosine         93.85      62.44
  3  weighted_voting  cosine         93.85      63.83
  5  most_common      cosine         93.85      63.07
  5  weighted_voting  cosine         93.85      62.96
  7  most_common      cosine         92.60      62.52
  7  weighted_voting  cosine         92.60      61.96
  1  most_common      canberra       92.42      49.53
  1  weighted_voting  canberra       92.42      49.77
```

Figure 2: 10 best results of the weighted Knn algorithm for the datasets: autos and hepatitis.

6. Modify the weightedKNN algorithm so that it only uses a subset of the most relevant features, you will call this algorithm as selectionKNNalgorithm(...). The selection will be extracted using weighting with an appropriate policy to discard features or feature selection algorithms.

For the selection Knn, we modified our knn(...) method and added a flag to determine whether to use or not feature selection. In the case it is, we use Sklearns “SelectKBest” using the training set and training classes and get the weights of each feature with the “f_classif” scoring function and the “SelectFromModel” is also used with the threshold being the median of the data and with feature selection based on l1. We remove the features that have less weight from the training dataset and the testing instance as a preprocessing step. For both algorithms we remove those that are less than the median of the important features. For the SelectKBest, this means those that are less than the median weight, and for the SelectFromModel, those that the algorithm determine are of less importance.

b. Analyze the results of the selectionKNNalgorithm in front of the previous

weightedKNNalgorithm implementation. To do it, setup both algorithms with the best combination obtained in your previous analysis. In this case, you will analyze your results in terms of classification accuracy.

Results when using weights and selection of features in the KNN algorithm:

```
Dataset: data/autos
Using weights: False
Using feature selection: True
Sorting by time: False


  k  select           distance      mean %    time ms
---  ---------------  ----------  --------  ---------
  1  most_common      cosine         96.27     231.13
  1  weighted_voting  cosine         95.06     234.03
  3  most_common      cosine         94.47     231.40
  3  weighted_voting  cosine         93.75     226.19
  5  most_common      cosine         93.75     226.98
  5  weighted_voting  cosine         93.22     224.79
  7  most_common      cosine         93.22     228.06
  7  weighted_voting  cosine         92.89     237.14
  1  most_common      canberra       92.33     189.57
  1  weighted_voting  canberra       91.33     189.45


Dataset: data/hepatitis
Using weights: False
Using feature selection: True
Sorting by time: False

  k  select           distance      mean %    time ms
---  ---------------  ----------  --------  ---------
  1  most_common      cosine         94.96      68.34
  1  weighted_voting  cosine         93.85      72.34
  3  most_common      cosine         93.71      70.14
  3  weighted_voting  cosine         92.96      69.79
  5  most_common      cosine         92.74      67.63
  5  weighted_voting  cosine         92.60      68.69
  7  most_common      cosine         92.60      68.45
  7  weighted_voting  cosine         92.60      68.14
  1  most_common      canberra       91.49      54.36
  1  weighted_voting  canberra       91.49      58.06
```

Figure 3: 10 best results of the feature selection SelectKBest for the datasets: autos and hepatitis.

```
Dataset: data/autos
Using weights: False
Using feature selection: True
Sorting by time: False

  k  select           distance      mean %    time ms
---  ---------------  ----------  --------  ---------
```

```
  1  most_common      euclidean      96.98     107.23
  1  weighted_voting  euclidean      95.77     104.95
  3  most_common      euclidean      93.84     104.07
  3  weighted_voting  euclidean      93.25     107.88
  5  most_common      euclidean      93.25     107.81
  5  weighted_voting  euclidean      91.95     106.43
  7  most_common      euclidean      91.95     107.97
  7  weighted_voting  euclidean      91.73     109.46
  1  most_common      canberra       91.40     240.40
  1  weighted_voting  canberra       91.21     240.97


Dataset: data/hepatitis
Using weights: False
Using feature selection: True
Sorting by time: False


  k  select           distance      mean %    time ms
---  ---------------  ----------  --------  ---------
  1  weighted_voting  cosine         93.85      68.87
  1  most_common      cosine         93.85      71.71
  3  weighted_voting  cosine         93.85      68.07
  3  most_common      cosine         93.85      67.51
  5  weighted_voting  cosine         93.85      70.92
  5  most_common      cosine         93.85      67.41
  7  weighted_voting  cosine         92.60      67.68
  7  most_common      cosine         92.60      68.90
  1  weighted_voting  euclidean      92.42      19.71
  1  most_common      euclidean      92.42      19.82
```

Figure 4: 10 best results of the feature selection SelectFromModel for the datasets: autos and hepatitis.

Compatibility

It seems that when running on python 2 we get inaccurate results due to the lack of automatic conversion to floating point numbers when needed. We corrected the issues to try to be compatible both with python 2 and 3, however we strongly recommend to execute the code in python 3.

Execution

In order to generate the tables a shell script `run.sh` executes all the combination of options for the datasets and adds the results to a “result.txt” file. The path of the datasets is already set inside the script. The main python program is in the `lazy.py` file:

```
$ python lazy.py -h
usage: lazy.py [-h] [-w] [-s] [-sm] [-t] dataset

Fit a dataset using KNN lazy learning

positional arguments:
  dataset             Folder containing the dataset in folds
```

```
optional arguments:
  -h, --help          show this help message and exit
  -w, --weights       use weights (default: no)
  -s, --select        use feature selection (default: no)
  -sm, --selectmodel  use feature selection with SelectFromModel (default:
no)
  -t, --time          sort and filter results by time (default: no (sorted
by correct percent))
```

Figure 5: execution of the python program

Results

In the following table, we show the best result for each algorithm based on the accuracy of the classification of the autos and hepatitis datasets respectively:

	K	Selection	Distance	Correct %	Time ms
Knn	1	most_common	cosine	96.36	24.93
WKnn	1	weighted_voting	euclidean	96.98	42.32
SKnn(SelectKBest)	1	most_common	cosine	96.27	231.13
SKnn(SelectFromModel)	1	most_common	euclidean	96.98	107.23

Figure 6: comparison of the best results with each algorithm on the autos dataset

	K	Selection	Distance	Correct %	Time ms
Knn	1	weighted_voting	manhattan	95.83	20.62
WKnn	1	most_common	cosine	93.85	62.13
SKnn(SelectKBest)	1	most_common	cosine	94.96	68.34
SKnn(SelectFromModel)	1	weighted_voting	cosine	93.85	68.87

Figure 7: comparison of the best results with each algorithm on the hepatitis dataset

By comparing the results, we can determine that generally, the normal Knn algorithm can work just fine without any weighing or feature selection algorithm, doing this increases the time required to classify the information substantially and doesn't always result in better classification accuracy. The feature selection algorithm takes more time than when using weighting, but both still take a lot more than with normal Knn

We can also notice that the best K for the accuracy in both datasets was 1, and the best distance functions for the datasets where cosine this must be because of how the data is distributed, for other cases, K can be best if it is higher and other distances such as manhattan can have better output.

As for the selection scheme, both seem to work with similar results, being most of the time decimals of percentage apart from each other when measuring results, so no major difference is made if selecting weighted voting or most common class.

For both datasets, the SelectKBest feature selection method worked best in accuracy by a small margin compared to the results from SelectFromModel for the hepatitis dataset. But, in the auto dataset, the SelectFromModel seems to work best, specially in time, since it took about half the time than SelectKBest.