Classification Tasks

ML Exercise 2 - Group 19

December 29, 2016

Group	Lukas Stanek	1027203
	Thomas Appler	
	Marten Sigwart	1638152

1 Introduction

Our task was to apply 4 different classification algorithms to 4 different data sets. Hereby, we had to experiment with different parameter settings of the classification algorithm, evaluate the performance by choosing appropriate performance measures and compare results among classifiers and data sets. We also had to perform some pre-processing steps on the data prior to applying the classifiers and study the impact of those pre-processing methods.

1.1 4 data sets

We chose four different data sets for the task. We chose data sets which varied in size, varied in the number of attributes and varied in the number of instances. Further we tried to pick data sets with different numbers of classes. Some data sets had missing values to take into account.

The four data sets we chose are:

- **Letter Recognition** This data set contained attributes about letters shown on a rectangular, black and white display. The classification task is to determine the right letter (A-Z)
- **Internet Advertisements** This data set contained information about images of different web pages. The classification task is to determine whether or not an image is an advertisement.
- **Leukemia** This data set contained medical information of cancer patients. The classification task is to determine whether or not a certain patients has cancer.
- **Congressional Voting** This data set contains information on different voters. The classification task is to determine the party affiliation of the voters.

Each data set will be described in more detail in sections below.

1.2 4 classification algorithms

We chose 4 different classifiers for the task. Hereby, it was important that at least 3 of the classifiers derived from totally different learning algorithms. The classifiers we chose are:

- **K-Nearest Neighbors**
- **Random Forrest**
- **Decision Tree**
- **Bayes Network**

1.2.1 Algorithm 1: K-Nearest Neighbors

KNN itself doesn't deal with missing values, it's the distance function. E.g., in case of euclidean distance, the following applies to missing values while computing the distance of the same attribute between two weka.core.Instance objects: - attributes in both Instance objects have missing value =¿ 0 - only one value is missing   nominal attribute =¿ 1   numeric attribute =¿ 1 in case of normalization, maxRange-minRange for that attribute otherwise TODO

1.2.2 Random Forest: Random Forrest

TODO

1.2.3 Decision Tree - 48J: Decision Tree

TODO

1.2.4 Bayes Network: Bayes??

TODO

1.3 Performance Measures

TODO

2 Data Set 1: Letter Recognition

The objective for this data set is to predict a letter shown on a rectangular, black and white, display. There are 16 numerical features provided for determining the correct letter. These attributes provide information about statistical properties of the letter like total number of pixels, width, mean of x-axis pixels, etc. All of these values were then scaled into a range from 0 through 15. The 26 capital letters to predict in this set were taken from 20 different fonts. The letters are more or less evenly spread over the 20.000 instances, within a range of 734 to 813 occurrences. This data set contains no missing values. For comparable results 10-fold cross validation was used to measure the algorithms precision. As for performance criteria with this data set only the percentage of correctly classified instances will be considered. The reason for this being that if a letter is wrongly classified, it doesn't matter which other letter was recognized instead of the correct one.

2.1 Preprocessing

As the data set was only provided as CSV file without header row, we added a header row for displaying the attribute names in Weka and therefore facilitating the analysis of the results.

2.2 K-Nearest Neighbors

In order to be able to compare the results we always used the same number of neighbors: 1,2,3,4,5,6,8,10,16,20,40,100.
The algorithm is generally very fast, even with 100 neighbors, it is usually finished within 30 seconds.

2.2.1 Default implementation

At first we started out using the default KNN implementation of Weka with default settings. The defaults are a *LinearNN* search algorithm with the *Eucledian distance* as distance function. We started out with increasing the amount of neighbors starting at one. Though with this settings we got the best results using just the nearest neighbor.

K	**% Correct**	**% Wrong**
1	95.96	4.04
2	94.925	5.075
3	95.635	4.365

In order to improve the results we started with weighting the neighbors after their distance which helped us further improving the results. The best results where achieved by weighting the neighbors *1/distance*. While the results yielded by the weighting function *1-distance* where also better than without weighting, the were not as good.

K	**% Correct**	**% Wrong**
1	95.96	4.04
2	95.99	4.01
3	96.04	3.96
4	**96.115**	**3.885**
5	96.03	3.97
6	96.015	3.985
8	95.75	4.25

With high values for K the prediction rates get worse.

K	**% Correct**	**% Wrong**
40	90.915	9.085
100	84.3	15.7

2.2.2 Manhattan Distance

did not improve our prediction, although again when the neighbors were weighted by their distance the success rate improved.

2.2.3 Chebyshev Distance

also did not improve the result, as it only takes the distance between the attributes which are farthest away from each other as result. As for weighting again, the results improved slightly when using weighted neighbors.

2.2.4 KD Tree Search Method

improves the run time of the classification significantly. The time for evaluating the models with 10-folds cross validation decreased by 66%.

2.3 Random Forest

The default settings for Random Forests are 100 Trees with unlimited depth, the number of randomly chosen features per tree is calculated by *log_2(#predictors) + 1*.

Number of Trees We started out with the default settings and varying the number of trees, starting at 100 trees. The result yielded was already at 96.41% accuracy. Further increasing the number of trees improved the result continuously, but also increased the computation time.

# Trees	% Accuracy	Computation time (s)
50	96.10%	3.1s
100	96.41%	6.55s
200	96.53%	12.33s
300	96.51%	19.13s
400	96.54%	24.46s
500	96.58%	30.66s

Table 1: Results for default settings and different # of trees

Though the performance is improving with the number of trees, the improvements are getting smaller. While an increase of trees from 50 to 100 trees results in a gain of 0.3 percent, the increase from 400 to 500 trees only gains 0.04 percent accuracy.

2.3.1 Increasing Execution Slots

For computing a 10-fold cross-validation at 500 trees on one single thread, the taken time was 5:55min. In order to improve this, the number of execution slots (=threads) can be increased. When repeating the same operation with 10 slots our computation time improved to 1:56min.
Using 50 threads we calculated a model with 1000 trees, which yielded the best result so far with an accuracy of *96.63%*.

2.3.2 Modifying number of features per tree

Through varying the number of features which will be selected per tree, we tried to increase the accuracy.
Results for *#Features: 3*:

# Trees	% Accuracy
100	96.51%
200	96.66%
300	96.78%
400	96.75%
500	96.75%

Table 2: Results for 3-feature trees

The best results were achieved with a number of 3 features per tree, other numbers only yielded worse results.

2.3.3 Modifying tree depth

Through modifying the maxial depth a tree is allowed to have we also tried to improve the results of our predictions.We observed that when reducing the depth drastically to something below 10, the prediction rate drops drastically.

Max depth	**Accuracy**
3	56.49%
6	73.97%

Table 3: Results below 10

But when raising the depth to 20 - 30 the rates start to normalize again also clieb a bit higher than on runs with the default settings.

Max depth	**Accuracy**
25	96.415%
26	96.42%
27	96.42%
28	96.41%

Table 4: Results above 25

2.3.4 Best combination

The best combination of settings we found was a Tree depth of 27 and the number of features set to 3. The result was a accuracy of **96.775%**.

2.4 Decision Tree - 48J

As for this implementation of the decision tree we also started with the default settings for this data set. The first results were significantly worse compared to the other algorithms. The prediction accuracy was down to 87.92% and also the ROC Area had dropped to 0.954 (The other algorithms were basically at 0.99 or 1). The tree had 1226 leafs and a size of 2451.

2.4.1 Number of objects per leaf

At first we tried to modify the minimum number of objects per leaf. With the default being two we tried to set it to one, which increased the number of leafs to 1771 and the size of the tree to 3541. The result was slightly better but it seemed like overfitting.
Although increasing the number of minimum nodes per tree resulted in worse results.

Obj. per leaf	**% Accuracy**	**# Leafs**	**Treesize**
1	88.66%	1771	3541
2	87.92%	1226	2451
3	87.20%	1015	2029
4	86.36%	852	1703
5	85.82%	750	1499
8	84.07%	563	1125

Table 5: Results for varying obj per leaf

2.4.2 Pruning

Pruning helps to prevent overfitting a decison tree, so that it is specialized to one dataset and does not generalize well. We adjusted a setting called confidence factor to influence the amount of pruning done during model creation. A higher confidence factor results in less pruning.

Confidence Factor	**% Accuracy**	**# Leafs**	**Treesize**
0.1	87.85%	1138	2275
0.2	87.88%	1198	2395
0.3	87.96%	1255	2509
0.4	87.94%	1255	2509
0.5	87.96%	1267	2533
0.6	87.98%	1303	2605

Table 6: Ajusting the confidence factor

The accuracy is increasing when pruning is decreased, but this would also increase the chances of the tree being overfitted.
It is also possible to change the pruning mode from C4.5 pruning to reduced-error pruning. It yields slighly worse results than the default pruning method.

2.5 Bayes Network

When using da Bayes Network implementation in Weka we can adjust 2 settings, the *estimator* and the *searchAlgorithm*. With the default settings, which use the K2 search algorithm, the result is an accuracy of *74.31%*. We were able to improve the prediction by increasing the size of parents every network node is allowed to have.

Max # parents	**% Accuracy**
1	74.31%
2	83.69%
3	86.72%
4	86.76%
5	86.76%
10	86.76%

Table 7: Ajusting the ne maximum number of parents of a node

After using a random order of node in the network the results got slightly better, but unfortionatly the random function takes no seed, so that the results vary over time.

Round	**% Accuracy**
1	88.455%
2	88.26%
3	88.46%
4	88.185%
5	87.88%

Table 8: Accuracy of random node order

2.5.1 HillClimber

By using the Hill climber algorithm we could achieve once again better results. But we needed to raise the number of parent nodes agian.

# Parents	**% Accuracy**
1	74.31%
4	89.45%
6	89.45%

Table 9: Accuracy HillClimber search algorithm

We could achieve similar results with the other Hillclimber algorithms *LAGDHill-climber* and *Repeated HillClimber* . They also achieved results of 89% and above.

2.5.2 Tabu search

When using the Tabu Search algorithm we could also improve our results by increasing the number of runs and the length of the tabu list.

# Parents	**# Runs**	**Length Tabu list**	**% Accuracy**
1	10	5	74.31%
4	10	5	84.26%
4	100	10	89.45%

Table 10: Accuracy Tabu search algorithm

So for Bayes Networks and this data set you should be using HillClimber based algorithms as they perform slighlty better than a K2. We could not test the genetic search algorithm as the computation time was to long.

2.6 Conclusion

2.6.1 Letter mix-up

When predicting letters there are some which are pretty well distinguishable from one another and some are not. So it happens that its easy to confuse two letters which look similar to each other. This also applys to our machine learning algorithms. Some letters have a worse prediction rate than others. We

can tell which characters are often confused by looking at the confusion matrix. It tells us how often a character was classified correctly or if not for which letter it was mistaken. This allows us also to tell whether the confusion is only going one-way. Meaning that one character is often mistaken for another one, but the other mostly classified correctly.
The top candiates for confusion are:

J	I
P	F
H	K
Q	O

J and *I* are mistaken with each other equally often, it seems to be generally harder to distinguish these two letters. This also seems to be logical as they look very similar and their geometrical properties could be very similar.
Interestingly why *Q* is often mistaken for *O*, *O* is much less mistaken for *Q*. The same can be said about *P* and *F*, where *P* is more often mistaken for *F*, and *H* and *K*.

3 Data Set 2: Internet Advertisements

This data set contains information on different banners/images on websites. The classification task on hand is to determine whether or not an instance is an advertisement or not. A possible use case for this data set would be to remove all adverts from a given website. Some attributes include the geometry of the images (if available), as well as phrases occurring in the URL, the image's URL and alt text, the anchor text and words near the anchor text.

The data set contains a total of 3279 instances, 2821 of which are non-advertisements, 458 are advertisements. Further the data set contains a total of 1558 attributes, 3 of which are continuous variables, the other ones all binary. 28% of instances are missing one or more of the continuous variables.

3.1 Preprocessing

The data set, was contained in two files, one with file ending .name containing all the names of attributes, the other one with ending .data containing the data in a comma-separated format. As Weka prefers data in the ARFF format, we had to do some transformations on the data set prior to loading it into Weka. This was done by first adding the attribute names as header row into the .data file, and saving that file as a CSV file. This file could then be converted by Weka's CSV converter to get the desired ARFF format.

3.2 K-Nearest Neighbours

As first classification algorithm we used K-Nearest Neighbors. The parameters we experimented with were:

- The number of neighbors.
- The distance weighting.
- The search algorithm.
- The distance function.

We did not use any missing value replacement strategy for this classifier, as the distance function used by K-NN can handle missing values.

Number of Neighbors: First, we experimented with the number of neighbors used by the classification algorithm. We did experiments with 1,2,3,4,5,6,8,10,20,50 and 100 neighbors.

# of Neighbors	% Correctly Classified	TP Rate Ad	TP Rate Non-Ad	F-Measure
1	96.25%	0.854	0.98	0.966
2	95.58%	0.871	0.97	0.956
3	96.74%	0.804	0.994	0.966
4	96.68%	0.819	0.991	0.966
5	96.00%	0.747	0.995	0.958
6	96.00%	0.763	0.994	0.96
8	95.64%	0.710	0.996	0.953
10	95.27%	0.678	0.998	0.949
20	92.71%	0.486	0.999	0.916
50	88.9 %	0.211	0.999	0.857
100	88.2 %	0.161	0.999	0.843

Generally, we can observe that the classification task using K-NN rightly classifies the major part of instances. Depending on the number of neighbors up to almost 97% of instances is classified correctly. The best results seem to emerge when the number of neighbors is fairly low, starting from 1 neighbor we see a slight increase in rightly classified instances up until 3 neighbors. After that the percentage of rightly classified instances starts to decrease again. Looking at the F-Measure our assumption that we find the best classification results by using around 3-4 neighbors is further strengthened as we see F-Measure values of 0.966 for each case. The ROC curve however tells a slightly different story. Here we can we the highest values for 8-20 neighbors, but the overall the values lie in a range between 0.943 and 0.967 and are therefore quite small.

Another interesting observation is that the true positive rates for class Ad seem to decrease with an increasing number of neighbors where as the true positive rate for class Non-ad seems to increase up to 0.999 meaning that almost all "Non-ad" instances are classified correctly. We assume this is because our "ad" instances do not lie all closely together in the data space with respect to their attributes, but rather scattered around the space in smaller groups. Our assumption is that only a few images of class "ad" have very similar attributes, and therefore returning great true-positive rates for a small number of neighbors, whereas with more neighbors, the classifier has to deal with more images that are also quite similar to the given image but of different classes. As we have way more instances of class "non-ad" this comes down to a "majority decision" leaving little to no chance to the classifier but to classify the given instance as "non-ad", therefore explaining the drastic decrease in the true-positive rate for class "ad" and the increase of the same for class "non-ad".

For 2 - 4 neighbors we generally got very good classification results, which is why we think the "ideal" number of neighbors for k-NN lies in that region. However, you cannot choose the ideal number of neighbors without defining a clear idea of what you want to achieve. This may be different depending on the context. For example, looking at the general performance (F-Measure, ROC-Curve) we may conclude that 2-4 neighbors is ideal for the given data set. However, in the context of "ad" and "non-ad" images on web pages, one might want to achieve a high number of identified ads while keeping the number of non-ad images falsely classified as advertisements to a minimum as these might show important information to the user. In this case it might be better to set the number of neighbors to 10 or 20 as this will yield very high true positive rates for class "non-ad" and fairly good true positive rates for class "ad".

Distance Weighting: Next, we experimented with different distance weighting settings. In Weka one can choose between no distance weighting, 1/distance weighting, and 1-distance weighting. Like we discussed in the section before, 2-4 neighbors generally showed "the best" results in the classification, which is why we did the following experiments all using 2-4 neighbors as well as 10 and 100 neighbors just for comparison. Below are the results of the classification task:

Distance Weighting	**# of Neighbors**	**% Correctly Classified**	**TP Rate Ad**	**TP Rate**
No distance	2	95.58%	0.871	0.
1/distance	2	96.43%	0.854	0.9
1-distance	2	96.46%	0.856	0.9
No distance	3	96.74%	0.804	0.9
1/distance	3	96.74%	0.808	0.9
1-distance	3	96.71%	0.800	0.9
No distance	4	96.68%	0.819	0.9
1/distance	4	96.71%	0.804	0.9
1-distance	4	96.71%	0.800	0.9
No distance	10	95.27%	0.678	0.9
1/distance	10	95.73%	0.712	0.9
1-distance	10	95.18%	0.671	0.9
No distance	100	88.2 %	0.161	0.9
1/distance	100	90.18%	0.305	0.9
1-distance	100	88.2 %	0.161	0.9

For just 2 neighbors we can see a slight increase in correctly classified instances, also the F-Measure and ROC Area seem to hint at an improved classifier. However, for 3 and 4 neighbors all classifiers seem to show very similar results, so if the distance weighting has an impact, it is now so low that it is basically negligible. We observe the same picture while looking at the results for 10 or 100 neighbors. In general, the F-Measure and ROC Area show very similar results. Only the inverse distance weighting seems to slightly improve the classification on all experiments, which might indicate a more robust method for classification.

So even though the overall performance of the classifier does not seem to be affected that much by the distance weighting we can, however, make out a factor where it does have an impact. In the section above we looked at different numbers of neighbors and we noticed that the true-positive rate for class "ad" seems to decrease when we increase the number of neighbors. We explained this by only a few very similar instances of the class "ad" lying closely together in the data space while there are still quite a few similar instances but of class "non-ad" also in the neighborhood, which leads to some kind of majority count while classifying. The same behavior can also be observed when using distance weighting, though, the decrease of the true-positive rate is not as strong as for no distance weighting. We think this confirms our assumption that the immediate neighbors of an instance of class "ad" are also of class "ad". When using distance weighting within the classifier these instances now get assigned a bigger weight which enhances the probability that the given instance will also be classified as "ad". This reduces the effect of the majority count we observed when not using any distance weighting.

In conclusion, we can say while the distance weighting does not seem to have any strong impact on the general performance, one might still make use of it when the classes are not fairly balanced. E.g. using inverse distance weighting on the classifier might reduce the effect of the majority count when using more neighbors.

3.3 Random Forest

TODO

3.4 Decision Tree - 48J

TODO

3.5 Bayes Network

TODO

3.6 Conclusion

TODO

4 Data Set 3: Leukemia

TODO

4.0.1 Preprocessing

TODO what preprocessing was done

4.0.2 K-Nearest Neighbours

TODO

4.0.3 Random Forest

TODO

4.0.4 Decision Tree - 48J

TODO

4.0.5 Bayes Network

TODO

4.0.6 Conclusion

TODO

5 Data Set 4: Congressional Voting

TODO

5.0.1 Preprocessing

TODO what preprocessing was done

5.0.2 K-Nearest Neighbours

TODO

5.0.3 Random Forest

TODO

5.0.4 Decision Tree - 48J

TODO

5.0.5 Bayes Network

TODO

5.0.6 Conclusion

TODO

6 Conclusion

TODO