Zeszyty Naukowe WWSI, No 21, Vol. 13 2019, s. 7-22
DOI: 10.26348/znwwsi.21.07

Ensemble Classification: Example and Python Implementation

Piotr Andziak[*], Ewa Figielska[**]

Warsaw School of Computer Science

Abstract

The paper presents an ensemble classification method based on clustering, along with its implementation in the Python programming language. An illustrative example showing the method behavior is provided, and the results of a computational experiment performed on real life data sets are reported.

Keywords **–** Ensemble Classification, Clustering, Python

1. Introduction

Machine learning is a science area which aims to get computers to learn like humans do or even better. The learning process is expected to improve over time, without human interaction, fed by new data sets which may contain information about the world, some observations or other kind of data used in the learning process [1].

[*] E-mail: p_andziak@poczta.wwsi.edu.pl
[**] E-mail: efigielska@poczta.wwsi.edu.pl

Manuscript received September 8, 2019.

Supervised learning and unsupervised learning are two of the machine learning tasks [2].

The objective of supervised learning is to discover a function mapping an input to an output in a way learned from a training set. A training set consists of pairs: desired output values for input values. An algorithm learning the mapping function from the training dataset, in consecutive iterations is predicting the output which is compared to the desired output. The iterations continue until the algorithm achieves acceptable accuracy. The training set can be considered as a teacher supervising the learning process.

One of the subcategories of supervised learning is classification. The task of classification is to assign a new observation – a new input object – to one of the known, predefined categories. Assignment is performed based on the independent variables of the input object and the existing assignments learned from a training set.

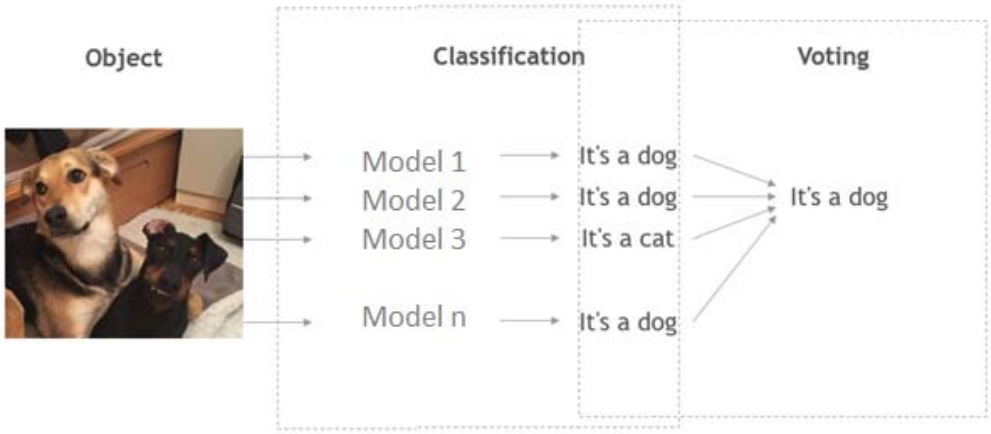

Figure 1. Ensemble classification

To perform more accurate classification, a set of classification models can be used instead of a single model and their outputs combined by a voting method. Various ways of making a final output prediction exist in the literature, for example the voting method may choose as the final prediction the one which has more than half of the

votes (majority voting) or one with the greatest number of votes (plurality voting) or it can determine the final prediction with regard to weights of classifiers depending on their importance (weighted voting). The technique of combining the results from a number of models is called ensemble learning. In Figure 1, the idea of the ensemble classification is presented.

Unsupervised learning, in oppose to supervised learning, does not use a training set. This process is expected to deduce and uncover an insightful structure or distribution in a provided data set. There is no teacher/supervisor, there are no wrong and correct answers. Algorithm is left to itself to learn the structure in data and present it. One of the unsupervised learning techniques is clustering. The objective of clustering is to discover useful groups in the input dataset, which consist of similar, alike objects.

The aim of this paper is to present the concept of ensemble classification as well as to show how an ensemble classification algorithm can be implemented in the Python programming language. As an example of an ensemble classification algorithm we have chosen the method proposed by Xiao *et al.* in [3]. This method builds a set of classifiers using training sets produced by a clustering algorithm. Beside the details on the implementation of this method, we provide an illustrative example and report some results of a computational experiment.

2. Ensemble classification based on clustering

In this section, we present a scheme of the ensemble classification algorithm based on clustering and the details of its implementation in the Python language. The results of the successive steps of the algorithm are shown using an illustrative example. To make the explanation of the examples and implementation easier, a two-class classification problem is considered.

The presented algorithm proceeds in six steps as shown in Figure 2. In the first four steps, a classifier consisting of a set of single base classifiers is constructed using the training data set. In the last two steps, the class prediction for a new sample is carried out.

Constructing a set of classifiers

1. Divide the data set into two subsets S^+ and S^- containing samples with class labels 1 and 0, respectively (i.e. a subset contains samples with the same class label);
2. Group data from S^+ and S^- into, respectively, K^+ positive clusters and K^- negative clusters;
3. Construct $K^+ \cdot K^-$ training sets by performing pairwise combination of positive and negative clusters;
4. For each training set, TS_j $(j = 1, \ldots, K^+ \cdot K^-)$, construct a base classifier G_j with a classification algorithm;

Classifying a sample X with unknown class label

5. Determine the voting weight W_j of each base classifier G_j on the basis of its accuracy in the neighborhood of sample X;
6. Predict a class label of sample X by combining the voting results of the base classifiers taking into account their weights.

Figure 2. The algorithm

In the next sub-sections, we explain the algorithm steps throughout their implementation, which is done in the Python 3.7 language. For the justification of the algorithm the reader is referred to [3].

2.1. Clustering

For grouping data into clusters (Step 2 of the algorithm) the k-means algorithm [2] is used. The calculations are performed for different numbers of clusters, K, ranging from 2 to $\sqrt{|S^+|}$ and to $\sqrt{|S^-|}$ for sets S^+ and S^-, respectively. In order to assess the clustering result, the validity function $VF(K)$ is calculated:

$$VF(K) = Intra \times Inter, \tag{1}$$

where $Intra$ and $Inter$ are, respectively, an index of compactness of the clusters and an index of separation between clusters.

$Intra$ represents the relative distance between each sample and the centroid of a cluster, with respect to the maximal distance between all samples and the centroid,

averaged over all clusters. $Inter$ considers the average distance between each pair of cluster centroids and the average distance of each cluster centroid to the center of all cluster centroids. Formally, the values of $Intra$ and $Inter$ are given by the following formulas [3]:

$$Intra = \frac{1}{K}\sum_{k=1}^{K}\frac{\sum_{i=1}^{m_k}|X_i^k - r^k|}{m_k \max\limits_{j\in[1,\ldots,m_k]}\left|X_j^k - r^k\right|}, \tag{2}$$

$$Inter = \exp\left(-\frac{2\sum_{1\le i<j\le K}|r^i - r^j|/(K(K-1))}{\sum_{k=1}^{K}|r^k - r|/K}\right), \tag{3}$$

where:

m_k is the number of samples in cluster of index k,

X_i^k denotes sample i belonging to cluster k,

r^k is the centroid of cluster k,

r is the center of all the cluster centroids.

The values of $Intra$ and $Inter$ range from 0 to 1.

Smaller values of $Intra$ and $Inter$ indicate better compactness and separation, respectively. So, the best clustering result is the one with the minimal value of $VP(K)$.

In Figure 3, the implementation of the clustering function is shown. Parameter s stores the data set containing samples with either 1 or 0 class label. During the execution of the program, the clustering function is called twice, once with s containing data from set S^+ and once with data from S^-. The function successively checks the values of $VP(K)$ for every K bounded by k_min and k_max (which are equal to 2 and $\sqrt{|S|}$, respectively). It chooses and returns the number of clusters, K, and the clusters themselves (in dictionary variable k_opt) for which $VP(K)$ has the smallest value.

```
def make_clusters(k_min, k_max, s):
    k_opt = {'VF': 1, 'K': 0}
    for K in range(k_min, k_max + 1, 1): #checking different numbers of clusters
        kmeans = KMeans(n_clusters=K, random_state=0).fit(s) #calling k-means alg.

        cluster_map = pandas.DataFrame()
        cluster_map['data_index'] = s.index.values
        cluster_map['cluster'] = numpy.array(kmeans.labels_).transpose()

        centers = kmeans.cluster_centers_

        intra_k = 0
        for k in range(0, K, 1): #calculating intra
            dist_sum = 0
            dist_max = 0
            samples = cluster_map[cluster_map.cluster == k].reset_index(drop=True)

            for i in range(0, len(samples.index), 1):
                dist = abs(distance.euclidean(s.loc[samples.loc[i,
                                              'data_index'], :].values, centers[k]))
                dist_sum += dist
                if dist > dist_max:
                    dist_max = dist
            if dist_max > 0:
                intra_k += dist_sum / (len(samples.index) * dist_max)
        intra = intra_k / K
        r_dist_sum = 0
        for x in itertools.product(centers, centers): #calculating inter
            if not (x[0] == x[1]).all():
                r_dist_sum += abs(distance.euclidean(x[0], x[1]))
        D = 2 * r_dist_sum / (K * (K - 1))
        r0 = numpy.mean(centers)
        r_to_r0 = 0
        for r in centers:
            r_to_r0 += abs(distance.euclidean(r, r0))
        b = r_to_r0 / K
        inter = math.exp(-D / b)
        vf = intra * inter
        if k_opt['VF'] > vf: #uppading clustering results
            k_opt['VF'] = vf
            k_opt['K'] = K
            k_opt['clusters'] = cluster_map
            k_opt['centers'] = centers
    return k_opt
```

Figure 3. Clustering function

2.2. Multiple classifier construction

After grouping data into clusters, the training subsets for building classifiers are constructed by pairwise combination of clusters with different classes. For constructing

classifiers three algorithms are selected: C4.5, CART [2] and SVM [4]. To construct C4.5 trees the implementation from Chefboost library [5] was used, while CART and SVM were sourced from Anaconda library [6].

Figure 4 shows the function creating a multiple classifier. It takes as the parameters the numbers of clusters K^+ and K^- in sets S^+ and S^- and sets S^+, S^- themselves (parameters k_p_opt, k_n_opt, s_pos, s_neg, respectively). The information about the classification algorithm to be used is passed to the function as parameter alg. The constructed training sets TS_j and classifiers G_j ($j = 1, \dots, K^+ \cdot K^-$) are stored in lists ts and $classifiers$, respectively, and returned by the function.

```
def make_classifiers(k_p_opt, k_n_opt, s_pos, s_neg, alg):
    clasifiers = []
    ts = []
    gnn = 0
    for i in range(0, k_p_opt['K'], 1):#performing pairwise combination
        for k in range(0, k_n_opt['K'], 1):
            tsj = pd.concat([s_pos.loc[s_pos['cluster'] == i],
                             s_neg.loc[s_neg['cluster'] == k]], axis=0,
                             ignore_index=False)
            ts.append(tsj)
            #constructing classifier
            if alg == 1:  # c45
                config = {'algorithm': 'C4.5'}
                gn = chef.fit(tsj.loc[:, tsj.drop(['cluster'],
                                                  axis=1).columns], config)
                chef.save_model(gn, "model"+str(gnn)+".pkl")
                gnn += 1
            elif alg == 2:  # cart
                tree = DecisionTreeClassifier()
                gn = tree.fit(tsj.loc[:, tsj.drop(['Decision',
                          'cluster'], axis=1).columns], tsj['Decision'])

            elif alg == 3:  # svm
                svmc = svm.LinearSVC(C=1.0, max_iter=1000000)
                gn = svmc.fit(tsj.loc[:, tsj.drop(['Decision',
                          'cluster'], axis=1).columns], tsj['Decision'])
            clasifiers.append(gn)
    return ts, classifiers
```

Figure 4. Multiple classifier construction

2.3. Prediction of a class label

In ensemble classification, to determine the class label for a new sample X, the predictions from the multiple models are combined. To achieve this, first, the weight, W_j, of each classifier G_j is calculated on the basis of its accuracy in the neighborhood of sample X. Then, classifiers with the highest weights are selected and their predictions are combined in a plurality voting.

```
for i in range(0, len(dataset_test.index), 1): #classifying each test sample
    Wj = []
    accuracy = []
    Xc = dataset_test.loc[i, 'Decision']
    X = dataset_test.loc[i,dataset_test.drop(['Decision'],axis=1).columns]
    X = pd.DataFrame(X.values.tolist(), X.index).transpose()
    for j, g in enumerate(classifiers): #computing accuracy of classifiers
        correct = 0                     #in the neighborhood of a test sample
        tss = ts[j].reset_index(drop=True)
        neighbors = len(tss) // 2  # M
        nn = NearestNeighbors(n_neighbors=neighbors)
        nn.fit(tss.loc[:,tss.drop(['Decision','cluster'],axis=1).columns])
        ngh = nn.kneighbors(X, return_distance=False)
        txx=tss.loc[ngh[0],tss.drop(['Decision','cluster'],axis=1).columns]
        result = {}
        for index, row in txx.iterrows():
             vals = (row.to_list()
             if alg == 1:  # c45
                g = chef.load_model("model"+str(j)+".pkl")
                result[index] = chef.predict(g, vals)
            elif alg == 2:  # cart
                result[index] = g.predict([vals])[0]
            elif alg == 3:  # svm
                result[index] = g.predict([vals])[0]
        c = tss.loc[ngh[0], 'Decision']
        for index, row in result.items():
            if str(c[index]) == str(row):
                correct += 1
        accuracy.insert(j, (correct * 100) / len(result))
    for j, a in enumerate(accuracy): #calculating Wj for each classifier
        if sum(accuracy) == 0:
            Wj.insert(j, 0)
        else:
            Wj.insert(j, '{:.2f}'.format(a / sum(accuracy)))
    max_value = max(Wj)
    maxWj = [i for i, x in enumerate(Wj) if x == max_value]
    result = list()
    for indexvalue in maxWj: #classifying sample with classifiers with
        if alg == 1:  # c45                                  highest Wj
            classifiers[indexvalue] = chef.load_model("model" + str(indexvalue)+".pkl")
            result.append(chef.predict(classifiers[indexvalue], X.values.tolist()[0]))
        elif alg == 2: #cart
            result.append(classifiers[indexvalue].predict(X)[0])
        elif alg == 3: #svm
            result.append(classifiers[indexvalue].predict(X)[0])
    try:
        result = mode(result)
    except StatisticsError:  # if equal number of votes
        result = random.choice(result)
```

Figure 5. Prediction of a class label

In the implementation shown in Figure 5, in the first loop (lines 7-29), the accuracies of the classifiers in the neighborhood of sample X are determined. For discovering the nearest neighbors of sample X the $NearestNeighbors$ class from $scikit-learn$ Python library [7] is used. The neighborhood size is set in line 10 (here, to the half of the size of the training set), in line 11 an object of the $NearestNeighbors$ class is created, which is then filled with all the data samples, from which a predetermined number of neighbors is selected (lines 12-14). The classification of the neighbor samples by means of the indicated classification model is carried out in lines 16-24. The accuracy of each classifier is equal to the ratio of the number of samples correctly classified to the total number of the nearest neighbors (line 29). The weight of a classifier is calculated as the ratio of its accuracy to the sum of the accuracies of all the classifiers (line 34). Sample X is given a class label which is chosen by plurality voting from among the results of the classifiers with the greatest weights (lines 35-49).

3. Illustrative example

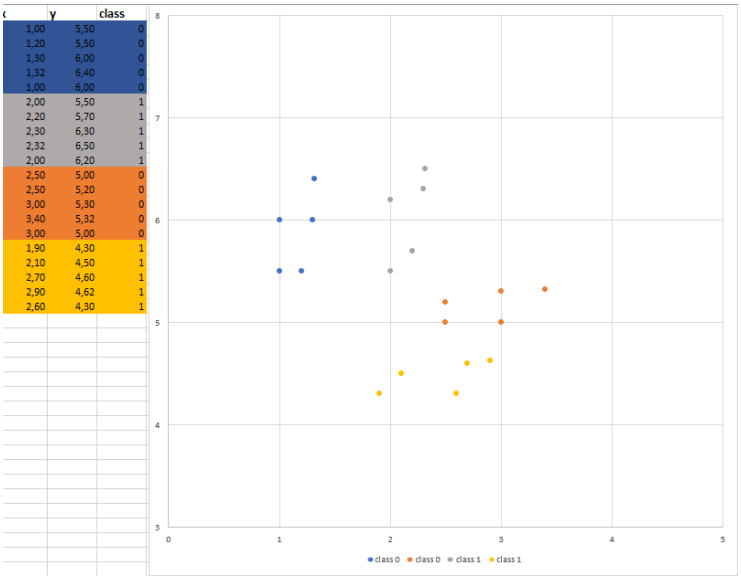

Figure 6. Data set

To test the implementation and visualize the execution of the algorithm, a data set was created which consists of 20 records described by 3 attributes, the last attribute has a binary nature and decides on the class of a sample. The data set is shown in Figure 6.

Before k-means clustering can be applied, the data set is split into two subsets S^+ and S^- according to a class label. The data from each subset are grouped into clusters. Let's consider grouping for the positive subset S^+.

Clustering function applies k-means clustering and calculates validity function $VF(K)$ for $K = 2$ and 3. The obtained results are as follows:
for $K = 2$, $intra = 0.68$, $inter = 0.36$ and $VF(2) = 0.24$,
for K = 3, $intra = 0.81$, $inter = 0.40$ and $VF(3) = 0.32$

Because $VF(K)$ has a smaller value for $K = 2$ than for $K = 3$, S^+ is split into two clusters as shown in Figure **7**. The same procedure is carried out for negative subset S^-, for which the best setting is also $K = 2$.

sample	x	y	Class	cluster
5	2	5,5	1	A
6	2,2	5,7	1	A
7	2,3	6,3	1	A
8	2,32	6,5	1	A
9	2	6,2	1	A
15	1,9	4,3	1	B
16	2,1	4,5	1	B
17	2,7	4,6	1	B
18	2,9	4,62	1	B
19	2,6	4,3	1	B

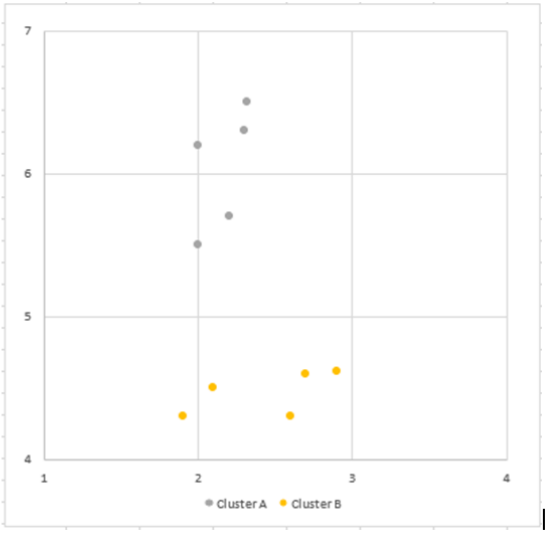

Figure 7. Clusters in set S^+

Training sets, TS_1, TS_2, TS_3, TS_4, being the result of the pairwise combination of clusters from S^+ and S^- are shown in Figure 8. Four classifiers, G_1, G_2, G_3, G_4, were built using these training sets by means of the CART algorithm.

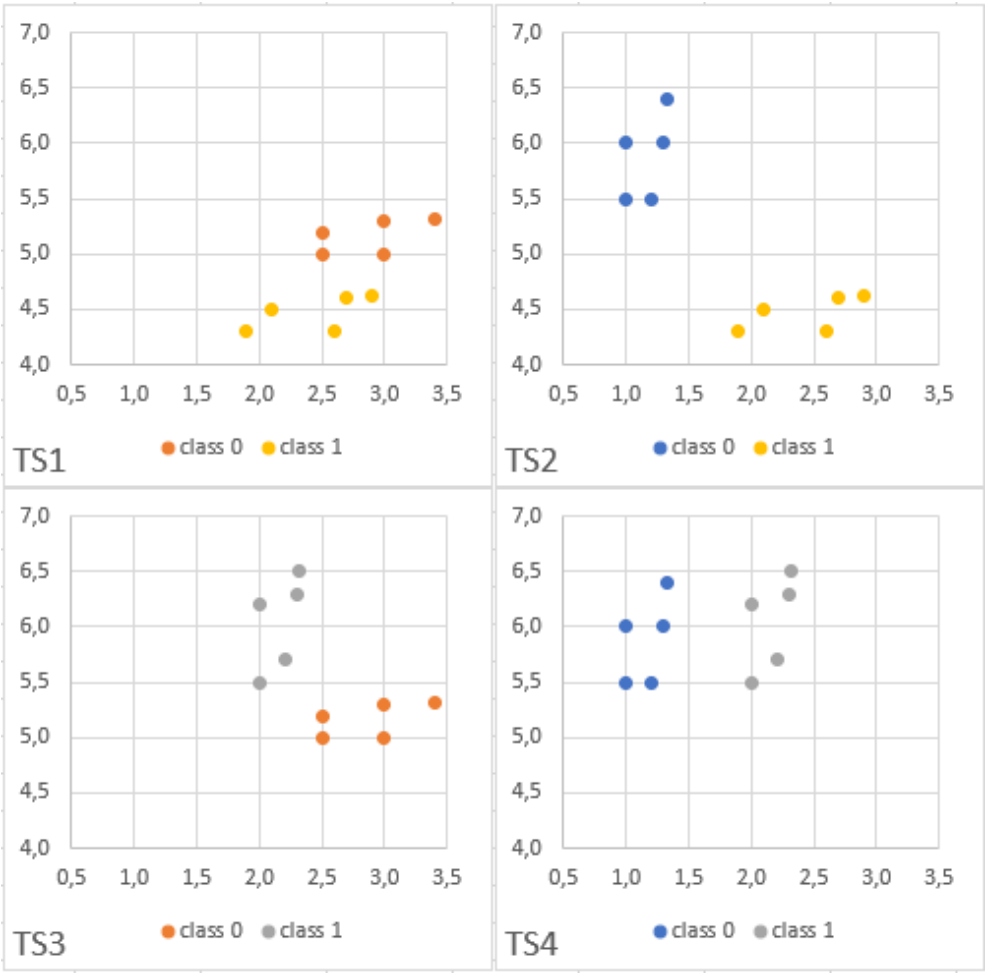

Figure 8. Pairwise combination of clusters

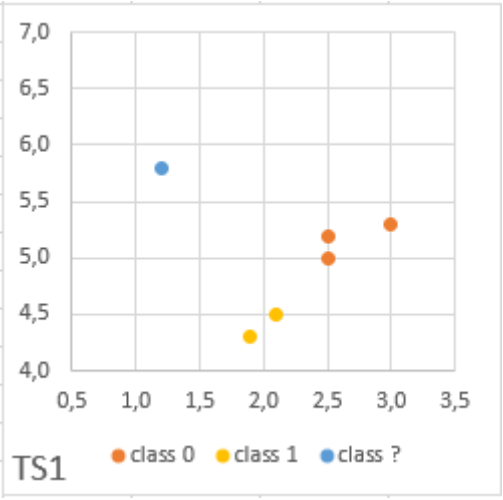

Figure 9. Sample with an unknown class label and its neighbours

Below, we describe the results of the classification of a new sample $X = [1.2, 5.8]$ with an unknown class label. Sample X and its five nearest neighbours are shown in Figure 9. The classification conducted by G_1 yielded the following predictions for the neighbours of X: [0, 0, 1, 1, 0]. The actual class labels are [0, 0, 1, 1, 0]. So, G_1 has 100% accuracy. The other classifiers had the same 100% accuracy. Thus, in the example, all the classifiers have the same weight.

Classifiers G_1, G_2, G_3, G_4 classified sample X as 0, 0, 1, 0 respectively. Therefore, by plurality voting, sample X has been classified as belonging to class 0.

4. Experiment

This section presents the results of a computational experiment which was conducted to evaluate the performance of the presented ensemble classification method. In the experiment, the following data sets from the UCI Machine Learning Repository were used:

- Banknote: a collection describing a banknote authentication problem [8];
- German: a data set widely used by researchers conducting tests related to credit scoring [9];

- Haberman: a data set containing cases from a study conducted on the survival of patients who had undergone surgery for breast cancer [10].

The presented method was run on each data set 10 times for each considered classification algorithm: C4.5, CART and SVM. In each run, a data set was divided randomly into a training subset and a test subset in proportions 0.8/0.2.

In the experiment, we considered 5 different values for the neighborhood size of a classified sample, which is denoted by M. M was set at: 1, 2, 5, 10 and a half of the training set size.

The results of a computational experiment, i.e. the average classification accuracy, are presented in Table 1.

Table 1. Classification accuracy [%]

Data set	***M***	**Classifier based on:**		
		C4.5	**CART**	**SVM**
Banknote	1	61.89	81.64	90.18
	2	66.18	82.98	91.02
	5	66.91	81.16	89.75
	10	66.61	82.69	90.15
	$(Data\ set\ size)/2$	67.21	81.78	90.40
German	1	64.05	67.35	67.40
	2	66.05	65.95	69.50
	5	58.95	64.80	66.05
	10	54.15	66.65	62.15
	$(Data\ set\ size)/2$	64.10	65.40	53.95
Haberman	1	60.32	68.71	72.58
	2	57.42	73.55	72.42
	5	53.87	69.19	70.00
	10	55.81	67.90	72.,10
	$(Data\ set\ size)/2$	62.74	70.65	72.90

The results of the experiment show that the accuracy of the presented method strongly depends on the choice of an algorithm constructing classifiers. Also, clear differences in the accuracy can be observed between the runs on different data sets.

We observe that, the SVM based classifier almost always outperforms the other ones for all three data sets. When $M = 2$, it is the best classifier for the Banknote and German data sets and the second best for the Haberman data set; in this case, the CART based classifier is slightly more accurate than the SVM based one. The C4.5 based classifier always exhibits the worst performance.

The greatest differences in the classifier performance are observed for the Banknote data set, where the SVM based classifier shows the advantage of about 8% and 25% over the CART and C4.5 based classifiers, respectively.

The parameter M only slightly affects the results, though for the German data set the lower values of M, like 1 or 2, seem to lead to better results than greater values of M.

Figure 10 presents the comparison of the accuracy of the classifiers averaged over the considered values of parameter M. We can see that the classification was the easiest for the Banknote data set; in this data set each classifier produces better results than in the remaining datasets.

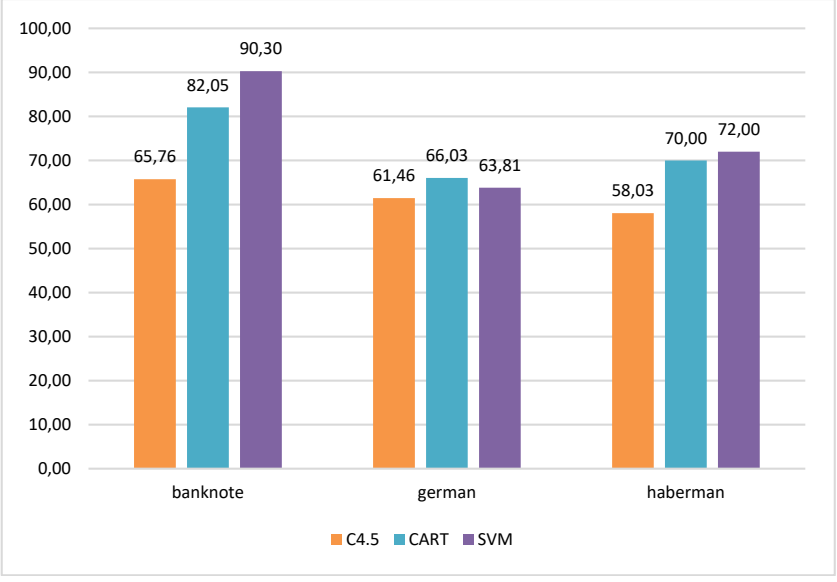

Figure 10. Accuracy comparison of the C4.5, CART and SVM based classifiers

5. Summary

Machine learning is an amazing field of science which we can benefit from in many ways, from banknote authentication in ATM, through credit risk calculation to cancer detection based on previous cases. It is obvious that this field of science will grow and will be put to research progressively more often and more intensively since our fast changing digital world is producing enormous amounts of data which can be used to deduce and present insightful knowledge, build smart systems and improve our life quality.

In this paper, the reader was introduced to the topic of ensemble classification. The ensemble classification method based on clustering has been presented along with its implementation in the Python programming language. The illustrative example has been provided to better understand how the method proceeds. Also, some computational results have been reported.

References

[1] D. Faggella, *What is Machine Learning*, https://emerj.com/ai-glossary-terms/what-is-machine-learning/ [20 August 2019].

[2] D.T. Larose, *Discovering knowledge in data*, New Jersey: John Wiley & Sons, Inc., 2005.

[3] H. Xiao, Z. Xiao, Y. Wang, *Ensemble classification based on supervised clustering for credit scoring*, Applied Soft Computing 43, p. 73–86, 2016.

[4] R. Gandhi, *Support Vector Machine — Introduction to Machine Learning Algorithms*, https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47 [20 August 2019].

[5] S.I. Serengil, *Chefboost*, https://github.com/serengil/chefboost [20 August 2019].

[6] *Anaconda*, https://www.anaconda.com/ [20 August 2019].

[7] *Scikit-learn*, https://scikit-learn.org/ [20 August 2019].

[8] V. Lohweg, *Banknote Authentication Data Set*, University of Applied Sciences, Ostwestfalen-Lippe, http://archive.ics.uci.edu/ml/datasets/banknote+authentication [20 August 2019].

[9] H. Hofmann, *German Credit Data Set*, University of Hamburg, https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data) [20 August 2019].

[10] T.-S. Lim, *Haberman's Survival Data Set*, University of Chicago's Billings Hospital, http://archive.ics.uci.edu/ml/datasets/Haberman%27s+Survival [20 August 2019].