On methods for improving the accuracy of multi-class classification on imbalanced data

Leonid A. Sevastianov[a], Eugene Yu. Shchetinin[b]

[a]Peoples' Friendship University of Russia (RUDN University), 6, Miklukho-Maklaya St., Moscow, 117198, Russia
[b]Financial University under the Government of the Russian Federation, 49, Leningradsky pr., Moscow, 117538, Russia

Abstract

Imbalance of the classes, characterized by a disproportional ratio of observations in each class, is one of the significant problems in machine learning. Class imbalances can be detected in many areas, including medical diagnostics, spam filtering, and fraud detection. Most machine learning algorithms work optimally when the number of samples in each class is approximately the same. This is because most algorithms are designed to maximize accuracy and reduce error. However, under conditions of class imbalance, the model may be overfitted, which leads to incorrect estimates of object classification. Thus, in order to avoid this phenomenon and achieve better results, it is necessary to research methods for working with unbalanced data, as well as develop effective algorithms for classifying them.

In this paper, we study machine learning methods to eliminate class imbalance in data in order to improve accuracy in multi-class classification problems. In this paper, to improve the accuracy of classification, it is proposed to use a combination of classification algorithms and feature selection methods RFE, Random Forest and Boruta with pre-balancing classes by random sampling, SMOTE and ADASYN. Using data on skin diseases as an example, computer experiments have shown that the use of sampling algorithms to eliminate the imbalance of classes, as well as the selection of the most informative features, significantly improves the accuracy of classification results. The Random Forest algorithm was the most effective in terms of classification accuracy when sampling data using the ADASYN algorithm.

Keywords

multiclass classification, imbalanced classes, machine learning, SMOTE, ADASYN, Random Forest

1. Introduction

Classification tasks are among the most popular in data analysis [1]. Supervised machine learning is most often used as the method for determining whether an object belongs to a particular class. The main idea of this approach is to inductively output a function based on marked-up data for training. This means that the success of using a machine learning classification algorithm depends largely on the selection of objects that the algorithm "learns" from. Most of these algorithms require the researcher to include a comparable number of examples for each of the classes, but it is often not possible to make balanced data sets due to a number of factors. Often there are situations when the dataset number of examples of some

Workshop on information technology and scientific computing in the framework of the X International Conference Information and Telecommunication Technologies and Mathematical Modeling of High-Tech Systems (ITTMM-2020), Moscow, Russian, April 13–17, 2020

sevastianov-la@rudn.ru (L. A. Sevastianov); riviera-molto@mail.ru (E. Yu. Shchetinin)
0000-0002-1856-4643 (L. A. Sevastianov); 0000-0003-3651-7629 (E. Yu. Shchetinin)

of the minor class (this class will be called the minority, and the other, prevailing over first — majority class). The key ones are the specificity of the target area (balancing data can lower the indicator of its representativeness) and the different price of errors of the first and second types when classifying. Such trends are clearly visible, for example, in credit scoring, medicine and marketing [2, 3].

This leads to the problem of training the model on imbalanced data (these are data whose distribution is skewed, and the mode and average values are not equal): according to the basic assumptions contained in most algorithms, the goal of training is to maximize the proportion of correct decisions relative to all decisions made, and the data for training and the general population are subject to the same distribution. However, taking into account these assumptions and unbalanced sampling results in the model being unable to classify data better than a trivial model that completely ignores a less represented class and marks all objects for classification as belonging to the majority class.

On the other hand, it is possible to build too much complex model that includes a large set of rules, but will cover a small number of objects. This classifier may be ineffective, which will lead the model to overfitting and incorrect estimates of the forecast. It should be noted that the consequences of erroneous classification may also differ. Moreover, an incorrect classification of examples of a minority class usually costs many times more than an erroneous classification of an object from a majority class. The correct selection of features may be more important than reducing data processing time or improving classification accuracy. for example, in medicine, finding the minimum set of features that is optimal for the classification task may be a prerequisite for making a diagnosis. Thus, to avoid this phenomenon and achieve a good result, it is necessary to research methods for working with imbalanced data.

In this paper, we study methods for overcoming imbalanced classes in order to improve the quality of classification with a higher accuracy than when directly using classification algorithms for imbalanced classes. To improve the accuracy of classification, we propose a scheme that consists of using a combination of classification algorithms and feature selection methods RFE, Random Forest and Boruta with the preliminary use of class balancing by random sampling, SMOTE and ADASYN.

2. Basic algorithms for balancing classes

One approach to solving this problem is to use various sampling strategies, which can be divided into two groups: random and special [3]. In the first case, delete a certain number of examples of the majority class (undersampling), in the second — increase the number of examples of the minority class (oversampling).

2.1. The exclusion of examples of the majority class. Algorithm for random sampling of the majority class (random undersampling)

To do this, we calculate the K – number of majority examples that must be removed to achieve the required ratio of different classes. Then K majority examples are randomly selected and removed. In the case of the studied data, methods for increasing the minority class are natural. Let's move on to the consideration of such strategies.

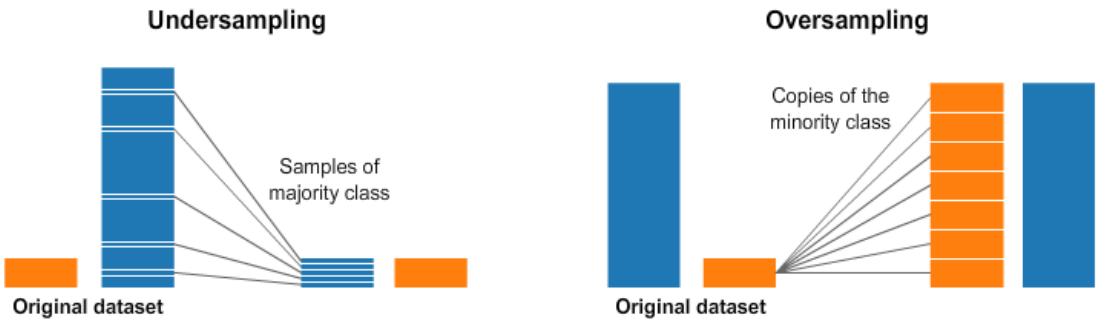


Figure 1: Undersampling and Oversampling balancing classes algorithms

2.2. The increase in the minority class. Duplicate examples of a minority class (oversampling). Random naive sampling

The easiest way to increase the number of examples of a minority class is to randomly select observations from it and add them to the general dataset until a balance is reached between the majority and minority classes. Depending on what class ratio is needed, the number of random records to duplicate is selected. One of the problems with random naive sampling is that it simply duplicates existing data. The advantages of this approach include its simplicity, ease of implementation and the ability to change the balance in any desired direction. The disadvantages should be discussed separately according to which sampling strategy is used: although both of them change the overall size of the data in order to find a balance, their application has different consequences. In the case of undersampling, deleting data may cause the class to lose important information and, as a result, lower its presentation rate.

In turn, the use of oversampling can lead to overfitting [3]. This approach to restoring balance is not always effective, so a special method was proposed to increase the number of examples of a minority class-the SMOTE algorithm (Synthetic Minority Oversampling Technique) [4]. The SMOTE algorithm is based on the idea of generating a certain number of artificial examples that are "similar" to those in the minority class, but do not duplicate them. To create a new record find the difference $d = X_b - X_a$, where X_a, X_b– feature vectors of "neighboring" examples a and b from the minority class. They are found using the nearest neighbors algorithm (KNN). In this case, it is necessary and sufficient for example b to get a set of k neighbors, from which the entry b will be selected later. The remaining steps of the KNN algorithm are not required. Then from d by multiplying each of its elements by a random number in the interval $(0, 1)$ we get $\tilde{d}$. The feature vector of the new example is calculated by adding X_a and $\tilde{d}$. The SMOTE algorithm allows you to set the number of records to be artificially generated. The degree of similarity of examples a and b can be adjusted by changing the value of k (the number of nearest neighbors). See for the illustration SMOTE algorithm on Figure 2.

SMOTE solves many problems that are inherent to the random sampling method, and actually increases the initial data set in such a way that the model is trained much more efficiently [5]. However, this algorithm has its drawbacks, the main of which is ignoring the majority class. This may result in a highly sparse distribution of objects of a minority class relative to a majority

class, where data sets are “mixed”, i.e. they are arranged in such a way that it is very difficult to separate objects of one class from another.

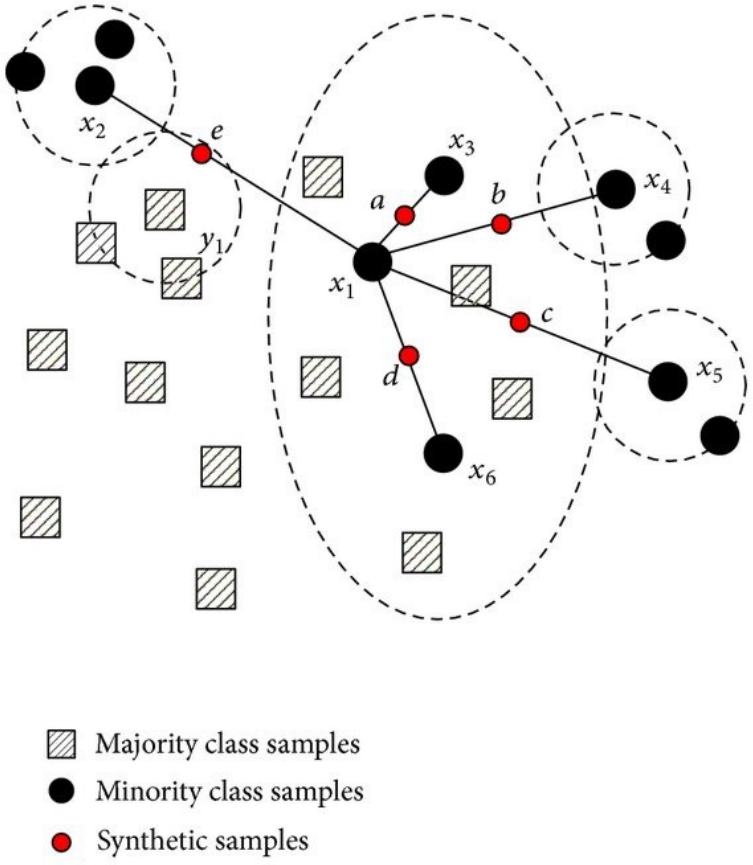


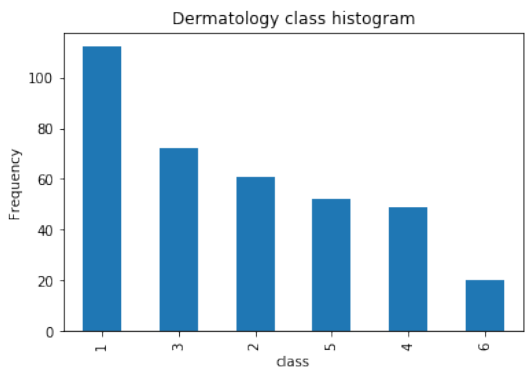
Figure 2: SMOTE balancing classes algorithm

An example of this phenomenon is when an object of a different class is located between an object and its neighbor, based on which a new instance is generated. As a result, the synthetically created object will be closer to the opposite class than to the class of its parents. In addition, the number of instances generated using SMOTE is set in advance, which reduces the ability to change the balance and flexibility of the method.

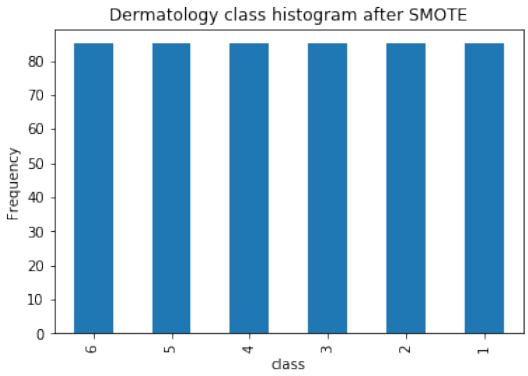
It is important to note the significant limitations of SMOTE algorithm [6]. Since it works by interpolating between rare examples, it can only generate examples inside the body of available examples — never outside. Formally, SMOTE can only fill in the convex hull of existing minority examples, but not create new external areas for them. The main advantage of SMOTE over traditional random naive over-sampling is that when creating synthetic observations instead of reusing existing observations, this classifier is less likely to be overfitted. At the same time, it is always necessary to make sure that the observations created by SMOTE are realistic.

Table 1
Results of balancing classes with SMOTE algorithm

Imbalanced classes



Classes after SMOTE



2.3. Adaptive synthetic sampling algorithm and its generalizations

This method is based on synthetic sampling algorithms, the main ones being Borderline-SMOTE and Adaptive Synthetic Sampling (ADASYN) [7, 8]. Borderline-SMOTE imposes restrictions on the selection of objects of the minority class that new instances are generated from. This happens as follows: for each object of a minority class, a set of k nearest neighbors is determined, then it is calculated how many instances of this set belong to the majority class (this number is taken as m).

After this, we select those objects of the minority class for which the inequality $k/2 \leqslant m < k$ is true. The resulting set represents instances of the minority class located on the distribution boundary, and they are the ones that are more likely to be incorrectly classified than the others. It should be noted why the inequality that determines the selection of objects excludes cases in which all k neighbors belong to the majority class: this is due to the fact that such instances are located in the "mixing" zone of two classes, and only objects that distort the model learning process can be generated on their basis. In this regard, they are declared as noise and are ignored by the algorithm.

The ADASYN algorithm, in turn, is based on a systematic method that allows adaptive generation of different amounts of data in accordance with their distributions [7]. Input data for the algorithm – training data set: D_r with m samples with $\{x_i, y_i\}$, $i = 1, \ldots, m$, where x_i– is the n– dimensional vector in the feature space, y_i– labels of corresponding class. Let's the m_r and m_x are the number of samples of minority and majority classes, respectively, such that $m_r \ll m_x$ and $m_x + m_r = m$. The algorithm's pseudocode looks like this:

1. Calculate the proportion of classes $d = m_r/m_x$;

2. If $d < d_x$ (where d_x is the specified threshold for the maximum allowable class imbalance):

a) Find the number of synthetically generated samples of the minor class $G = (m_x - m_r) \times \beta$, where β is the parameter used to determine the desired balance level ($\beta = 1$) indicates full class balance.

b) for each $x_i \in minority class$ find the K-nearest neighbors using the Euclidean distance and

calculate $r_i = \triangle_i / K$;

c) normalize $r_x = r_i / \sum_i r_i$ so that r_x becomes the distibution density;

d) calculate $g_i = r_x \times G$ a synthetic sample formed for each image from the minority class, where G is the total number of examples of synthetic data;

e) for each example of data from a minority class x_i create the examples of synthetic g_i data in accordance with the following steps:

In a cycle from 1 to i :

(i) randomly select one example of minority data, x_u from K nearest neighbors for x_i data;

(ii) create an example of synthetic data: $g_i = x_i + (x_u - x_i) \times \lambda$, where $(x_u - x_i)$ is n-dimensional vector of Euclidean space, λ — random number, $\lambda \in [0, 1]$.

The main difference between SMOTE and ADASYN is how to create synthetic sample samples for the minority class. ADASYN uses the r_x density function to determine the number of synthetic samples that will be created for a specific point, whereas SMOTE has a single weight for all minority points.

3. Research data: description and characteristics

In this paper, a set of data on skin diseases was used for testing and comparative analysis of the methods described above to eliminate the class imbalance. Diagnosis of erythematous squamous cell diseases is a serious problem in dermatology, and modern principles of diagnosis and treatment are based on the earliest detection of the disease. All of them have common clinical features with very small differences. Another difficulty for diagnosis is that the disease may show signs of another disease at the initial stage and may have characteristic signs in subsequent stages.

The study data was created by Nielsen in 1998 and contains 366 observations forming 6 classes that can be characterized by 34 features [9]. The classes are: psoriasis (class 1): — 112 cases; seborrheic dermatitis (class 2): — 72 cases; lichen planus (class 3): — 61 cases; pink lichen (class 4): — 49 cases; chronic dermatitis (class 5): — 52 cases; red hair lichen (class 6): — 20 cases. A full description of the data is given in [10].

4. Computer experiments

Data studies were performed using the following algorithm:

1. Data pre-processing: filling the gaps in the data and the coding of signs.
2. Balancing classes using the sampling algorithms described above.
3. Selecting attributes based on their importance.
4. Classification using logistic regression and the support vector method.
5. Assessment of classification quality.

In this paper, the selection of features based on their importance and informativeness was carried out by the following methods: a) recursive exclusion of RFE features [5]; b) decision trees RF [11]; c) Boruta [12].

The Random Forest algorithm is an ensemble of numerous classification algorithms (decision trees). Each of these classifiers is built on a random subset of objects and a random subset of features. Let the training sample consist of N examples, the dimension of the feature space is equal to M, and an additional parameter m is set. All trees are built independently of each other using the following procedure:

1. Generate a random sub-sample with a repeat of size n from the training sample.
2. Let's build a decision tree that classifies the examples of this sub-sample, and during the creation of the next node of the tree, we will select the feature based on which the partition is made, not from all M features, but only from m randomly selected ones.
3. The tree is built until the subsample is completely exhausted and does not undergo the procedure of cutting off branches.

Object classification is carried out by voting: each tree of the ensemble refers the object to be classified to one of the classes, and the class that the largest number of trees voted for wins. To use Random Forest in the task of evaluating the importance of features, it is necessary to train the algorithm on the sample and calculate the out-of-bag error for each example of the training sample [11].

Let X_n be a bootstrapped sample of the b_n tree. Bootstrapping is the selection of l objects from the selection with a return, as a result of which some objects are selected several times, and some – never. Placing multiple copies of the same object in a bootstrapped selection corresponds to setting the weight for this object, the corresponding term will be included in the functionality several times, and therefore the error penalty will be greater on it. Let $L(y, z)$ be the loss function, and y_i be the response on the i-th object of the training sample, then the out-of-bag error is calculated using the following formula:

$$OOB = \sum_{a}^{b} L\left(y_i, \frac{\sum_{n=1}^{N}\left[x_i \ni X_n^l\right] b_n(x_i)}{sum_{n=1}^{N}\left[x_i \ni X_n^l\right]}\right).$$

Then, for each object, this error is averaged across the entire random forest. To evaluate the feature importance, its values are mixed for all objects in the training sample, and the out-of-bag error is counted again. The importance of the features is estimated by averaging the difference in out-of-bag errors across all trees before and after mixing the values. The values of such errors are normalized to the standard deviation.

Boruta is a heuristic algorithm for selecting significant features based on the use of Random Forest [12]. At each iteration, features that have a Z-measure less than the maximum Z-measure among the added features are removed. To get the Z-measure of a feature, you need to calculate the feature's importance obtained using the built-in algorithm in Random Forest, and divide it by the standard deviation of the feature importance. The added features are obtained as follows: the features that are present in the selection are copied, and then each new feature is filled in by shuffling its values. In order to get statistically significant results, this procedure is repeated several times, and variables are generated independently at each iteration.

Let's write down the Boruta algorithm step by step:

1. Add copies of all attributes to the data. In the future, copies will be called hidden signs.

2. Randomly shuffle each hidden attribute.
3. Run Random Forest and get the Z-measure of all attributes.
4. Find the maximum I-measure of all I-measures for hidden features.
5. Delete features that have a Z-measure smaller than the one found in the previous step.
6. Remove all hidden attributes.
7. Repeat all the steps until the Z-measure of all features is greater than the maximum z-measure of hidden features.

5. Results and discussion

To solve the problem of multiclass classification on unbalanced data, machine learning algorithms were chosen: logistic regression and the method of support vectors with a linear kernel (Linear SVM). All calculations were implemented in PYTHON, their results, data, and program codes are placed in the repository of the authors of this article [10] and some algorithms in [13, 14, 15]. Some fragments are presented in Computer Code paragraph. Three metrics were used to compare classification results: accuracy, recall, and F1-measure. The results of the research are presented in Table 2, Table 3.

Table 2
Results of classification using the support vector method

Sampling of the imbalanced data	Feature selection methods	number of chosen features	Accuracy	F1-Score	Recall
Imbalanced data	all features	630	0,9324	0,9337	0,9324
	RFE	65	0,9595	0,9598	0,9595
	Random Forest	32	0,9595	0,9590	0,9595
	Boruta	207	0,9324	0,9330	0,9324
Random sample	all features	630	0,9324	0,9337	0,9324
	RFE	44	0,9359	0,9468	0,9459
	Random Forest	44	0,9465	0,9466	0,9465
	Boruta	284	0,9595	0,9598	0,9595
SMOTE	all features	630	0,9324	0,9337	0,9324
	RFE	68	0,9595	0,9730	0,9730
	Random Forest	42	0,9595	0,9072	0,9054
	Boruta	257	0,9459	0,9337	0,9324
ADASYN	all features	630	0,9324	0,9337	0,9324
	RFE	44	0,9459	0,9459	0,9459
	Random Forest	40	0,9845	0,9330	0,9324
	Boruta	276	0,9459	0,9602	0,9595

Table 3
Results of classification using the logistic regression

Sampling of the imbalanced data	Feature selection methods	number of chosen features	Accuracy	F1-Score	Recall
Imbalanced data	All features	630	0,9330	0,9234	0,9231
	RFE	19	0,9459	0,9459	0,9459
	Random Forest	32	0,9595	0,9590	0,9595
	Boruta	200	0.9595	0,9590	0,9595
Random Sample	All features	630	0,9630	0,9634	0,9630
	RFE	48	0,9665	0,9866	0,9865
	Random Forest	44	0,9730	0,9730	0,9730
	Boruta	290	0,9730	0,9765	0,9730
SMOTE	All features	630	0,9730	0,9734	0,9730
	RFE	20	0,9459	0,9459	0,9459
	Random Forest	41	0,9324	0,9330	0,9324
	Boruta	264	0,9595	0,9590	0,9595
ADASYN	All features	630	0,9530	0,9534	0,9530
	RFE	67	0,9595	0,9602	0,9595
	Random Forest	42	0,9895	0,9859	0,9893
	Boruta	245	0,9595	0,9590	0,9595

First column of Table 2 lists the sampling methods used. The second column shows the methods used for selecting features, and the third column shows the number of selected features. The remaining columns show the values of quality metrics obtained as a result of applying the support vector algorithm (SVM) to the converted data. The Table 3 is constructed similarly, containing the results of classification using logistic regression.

From the analysis of the obtained results, that are shown in Table 2, and Table 3, it can be seen that in all cases, the use of sampling methods allowed for higher classification accuracy than on unbalanced data. Within the framework of the scheme described in this paper, the best classification accuracy was achieved by applying the ADASYN class balancing algorithm and then selecting features using the Random Forest algorithm. For comparison, in the works of other researchers who conducted similar studies, for example, [5, 11], the classification accuracy reached only 93%.

6. Conclusion

In this paper, we propose a scheme for improving the accuracy of classification on unbalanced data using algorithms for class balancing and feature selection, such as RFE, Boruta, Random Forest, and others. The results of computational experiments have shown the effectiveness

of its application to solve this problem. In particular, the ADASYN algorithm has improved classification accuracy by up to 98% compared to other algorithms. In conclusion, it is worth noting that the problem discussed in this paper is still relevant, and existing methods can be improved. In recent time there are some new trends in data mining so called dee learning, developing the deep neural networks as a tool for solving various classification problems. So, we hope to apply them in our future researches of imbalanced classes classification.

Acknowledgments

The work is partially supported by RFBR grants No 18-07-00567.

References

[1] J. Patterson, A. Gibson, Deep Learning: A Practitioner's Approach, O'Reilly Media, 2017.
[2] H. He, E. Garcia, Learning from imbalanced data, IEEE Transactions on Knowledge and Data Engineering 21 (2009) 1263–1284. doi:10.1109/TKDE.2008.239.
[3] N. Japkowicz, S. Stephen, The class imbalance problem: A systematic study, Intelligent Data Analysis 6 (2002) 429–449. doi:10.3233/IDA-2002-6504.
[4] N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer, Smote: Synthetic minority over-sampling technique, Journal of Artificial Intelligence Research 16 (2002) 321–357. URL: http://dx.doi.org/10.1613/jair.953. doi:10.1613/jair.953.
[5] X. Lin, F. Yang, P. Yin, H. Kong, W. Xing, N. Wu, L. Jia, Q. Wang, G. Xu, A support vector machine-recursive feature elimination feature selection method based on artificial contrast variables and mutual information, Journal of Chromatography B: Analytical Technologies in the Biomedical and Life Sciences 910 (2012) 149–155. doi:10.1016/j.jchromb.2012.05.020.
[6] L. Abdi, S. Hashemi 28 (2016) 238–251.
[7] H. He, Y. Bai, E. A. Garcia, S. Li, Adasyn: Adaptive synthetic sampling approach for imbalanced learning, in: 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), 2008, pp. 1322–1328.
[8] H. Han, W.-Y. Wang, B.-H. Mao, Borderline-smote: A new over-sampling method in imbalanced data sets learning, volume 3644, 2005, pp. 878–887. doi:10.1007/11538059_91.
[9] P. M. Murphy, D. W. Aha, Uci repository of machine learning databases. – irvine: University of california, department of information and computer science, 1998. URL: https://www.ics.uci.edu/mlearn/MLRepository.html.
[10] Dermatology-article, 2020. URL: https://github.com/riviera2015/Dermatology-article.
[11] E. Tuv, A. Borisov, G. Runger, K. Torkkola, Feature selection with ensembles, artificial variables, and redundancy elimination, Journal of Machine Learning Research 10 (2009) 1341–1366.
[12] M. Kursa, W. Rudnicki, Feature selection with boruta package, Journal of Statistical Software 36 (2010) 1–13. doi:10.18637/jss.v036.i11.
[13] P. Lyubin, E. Shchetinin, Fast two-dimensional smoothing with discrete cosine transform, volume 678, 2016, pp. 646–656. doi:10.1007/978-3-319-51917-3_55.

[14] E. Y. Shchetinin, Cluster-based energy consumption forecasting in smart grids, in: V. M. Vishnevskiy, D. V. Kozyrev (Eds.), Distributed Computer and Communication Networks, volume 919, Springer International Publishing, Cham, 2018, pp. 445–456.

[15] L. A. Sevastianov, E. Y. Shchetinin, On methods for improving the accuracy of multiclass classification on imbalanced data, Informatics and Applications 14 (2020) 63–70. doi:10.14357/19922264200109.

A. Program Code: Deep CNN model

```
#=====================================================================
# All featuresbala
#=====================================================================
import seaborn as sns
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score, GridSearchCV

# Fit Logistic Features to all features
svc = LogisticRegression()
svc.fit(X_train, y_train)

# Test accuracy
acc = accuracy_score(y_test, svc.predict(X_test))
print('Test Accuracy {}'.format(acc))

# Plot confusion matrix
cm = confusion_matrix(y_test, svc.predict(X_test))
sns.heatmap(cm, fmt='d', cmap='GnBu', cbar=False, annot=True)
lr_pred = svc.predict(X_test)

#=====================================================================
# Recursive Feature Selection
#=====================================================================
from sklearn.feature_selection import RFECV
# RFE
rfe = RFECV(estimator=LogisticRegression(), cv=4, scoring='accuracy')
rfe = rfe.fit(X_train, y_train)

# Select variables and calulate test accuracy
cols = X_train.columns[rfe.support_]
acc = accuracy_score(y_test, rfe.estimator_.predict(X_test[cols]))
print('Number of features selected: {}'.format(rfe.n_features_))
print('Test Accuracy {}'.format(acc))
```

```
# Plot number of features vs CV scores
plt.figure()
plt.xlabel('k')
plt.ylabel('CV accuracy')
plt.plot(np.arange(1, rfe.grid_scores_.size+1), rfe.grid_scores_)
plt.show()
lr_pred = rfe.estimator_.predict(X_test[cols])
print('f1_score (macro)',f1_score(y_test, lr_pred, average='macro'))
print('f1_score (micro)',f1_score(y_test, lr_pred, average='micro'))
print('f1_score (weighted)',f1_score(y_test, lr_pred, average='weighted'))
print('recall_score (macro)',recall_score(y_test, lr_pred, average='macro'))
print('recall_score (micro)',recall_score(y_test, lr_pred, average='micro'))
print('recall_score (weighted)',recall_score(y_test, lr_pred, average='weighted'))

#=================================================================
# Feature importances
#=================================================================
from sklearn.ensemble import RandomForestClassifier
# Feature importance values from Random Forests
rf = RandomForestClassifier(n_jobs=-1, random_state=SEED)
rf.fit(X_train, y_train)
feat_imp = rf.feature_importances_

# Select features and fit Logistic Regression
cols = X_train.columns[feat_imp >= 0.01]
est_imp = LogisticRegression()
est_imp.fit(X_train[cols], y_train)

# Test accuracy
acc = accuracy_score(y_test, est_imp.predict(X_test[cols]))
print('Number of features selected: {}'.format(len(cols)))
print('Test Accuracy {}'.format(acc))
lr_pred = est_imp.predict(X_test[cols])
print('f1_score (macro)',f1_score(y_test, lr_pred, average='macro'))
print('f1_score (micro)',f1_score(y_test, lr_pred, average='micro'))
print('f1_score (weighted)',f1_score(y_test, lr_pred,
average='weighted'))
print('recall_score (macro)',recall_score(y_test, lr_pred,
average='macro'))
print('recall_score (micro)',recall_score(y_test, lr_pred,
average='micro'))
print('recall_score (weighted)',recall_score(y_test, lr_pred,
average='weighted'))
```

```
#=================================================================
# Boruta
#=================================================================
from boruta import BorutaPy
# Random Forests for Boruta
rf_boruta = RandomForestClassifier(n_jobs=-1, random_state=SEED)
# Perform Boruta
boruta = BorutaPy(rf_boruta, n_estimators='auto', verbose=2)
boruta.fit(X_train.values, y_train.values.ravel())

# Select features and fit Logistic Regression
cols = X_train.columns[boruta.support_]
est_boruta = LogisticRegression()
est_boruta.fit(X_train[cols], y_train)

# Test accuracy
acc = accuracy_score(y_test, est_boruta.predict(X_test[cols]))
print('Number of features selected: {}'.format(len(cols)))
print('Test Accuracy {}'.format(acc))
lr_pred = est_boruta.predict(X_test[cols])
print('f1_score (macro)',f1_score(y_test, lr_pred, average='macro'))
print('f1_score (micro)',f1_score(y_test, lr_pred, average='micro'))
print('f1_score (weighted)',f1_score(y_test, lr_pred,
average='weighted'))
print('recall_score (macro)',recall_score(y_test, lr_pred,
average='macro'))
print('recall_score (micro)',recall_score(y_test, lr_pred,
average='micro'))
print('recall_score (weighted)',recall_score(y_test, lr_pred,
average='weighted'))

from imblearn.over_sampling import RandomOverSampler
# random oversampling
ros = RandomOverSampler(random_state=0)
X_resampled, y_resampled = ros.fit_resample(X_train, y_train)
# using Counter to display results of naive oversampling
from collections import Counter
print(sorted(Counter(y_resampled).items()))

from imblearn.over_sampling import SMOTE
# applying SMOTE to our data and checking the class counts
X_resampled1, y_resampled1 = SMOTE().fit_resample(X_train, y_train)
print(sorted(Counter(y_resampled1).items()))
```