# News Classification on AG News Corpus
# Group 14 Project Report

Abbou, Jannik (1598438), Arslantürk, Fatih (1865936), Döhler, Simon Nikolaus (1608502), Eroglu, Zeynep (1834472), König, Malte (1653278), Scheffler, Marc (1817334), and Zimmer, Vincent Oliver (1822731)

Data and Web Science Group
Prof. Dr. Christian Bizer
University of Mannheim, Germany

**Abstract.** A subset of the AG news corpus, which is divided into training and test sets, is used to classify news article descriptions into labels of news categories. Our supervised learning approach for text mining includes several data mining algorithms, which are commonly used for text classification. Text will be preprocessed and transformed with Bag-of-Word approaches, as well as word embedding models. Different parameter settings are tried out during hyperparameter tuning. In evaluation we calculate the performance with the use of various metrics. We achieve very good results with different classifiers. Our highest accuracy (91%) is achieved by an implementation of a Support-Vector Machine built on a TF-IDF model of the data. All our development, code and results can be found on GitHub: `https://github.com/malte-koenig/AG_News_Classifier`.

**Keywords:** Text Classification · Text Mining · Supervised Learning · News Articles · Categories.

## 1 Application Area and Goals

There is a overwhelming amount of news articles and news sources on the internet. Online news papers, blogs and magazines categorize articles by their own rules. Getting a structured overview, fetching news articles from different websites in practical datasets or utilizing a news article search engine could benefit greatly from text mining and the classification of news.

Within the Data Mining I project we create such a news classifier. News classification enables customization and recommendations for news consumption. As for all text mining approaches, we want to achieve best results while keeping processing times low. The AG News Corpus consists of short news article descriptions. The descriptions are sufficient for this task. They summarize the articles enough, so running classification with full articles might be as exact as our approach, but the effort would be much higher. Furthermore, most news articles on the internet provide descriptions, so the classifier can be used in a

broad field of real world data. Our defined goal is that our classifier would eventually be able to handle arbitrary news descriptions to reliably classify them on the basis of predefined labels.

While we train the classification model on a given set of data and labels; training on any different dataset with different labels is possible to achieve structuring of labels in different departments.

## 2 dataset

The AG's news corpus contains more than 1 million categorized news articles that are gathered from more than 2.000 news sources by an academic news search engine called "ComeToMyHead"[1]. The AG dataset is provided by the academic community for research purposes in data mining and any other non-commercial activity.

The dataset we are using is a subset of the original corpus provided by the data mining website "papers with code"[2]. It contains 120.000 training samples and 7.600 testing samples. The dataset is build out of four different categories of articles: "World", "Sports", "Business" and "Sci/Tech". Every category covers 30.000 training samples and 1.900 testing samples. The original dataset contained 14 classes and several other fields, but had a lot of noise and format issues.

Finally our dataset is constructed out of only two columns: The article description and the category label. The four classes are assigned to a specific number from 0-3. The class "World" is assigned to the class id 0, "Sports" is assigned to 1, "Business" is assigned to 2 and "Sci/Tech" is assigned to 3. The dataset can be easily downloaded by using the python datasets library and after preprocessing we replace the id's with their category names. For documentation purposes only, we save the data as csv files, altough we access the data directly from the online source (Hugging Face Hub).

After getting the data, we investigated it further. We found some issues which we address in preprocessing, like HTML-tags or multiple whitespaces within the text. We are also interested in the average length of the article descriptions, because if the classifier will be used in a real world scenario, that is an important fact to know. Figure 1 shows the distribution of articles based on their length for each of the classes. The length of a article is defined by the number of words. We can see that most descriptions have a range of about 15-40 words across all categories. While the histograms for the other classes are very similar, the one for the category "World" is a little broader. In the end, a successful classifier built on this data would only need a few sentences about the article in order to make a decision about its category.

## 3 Preprocessing Steps

For the preprocessing we primarily use the python nltk and re library. We define the function preprocessing(text) which we execute for every article description

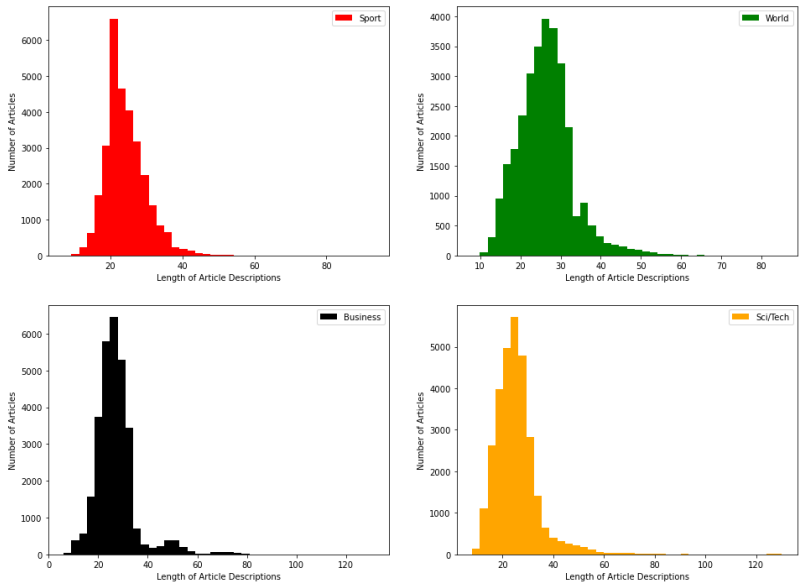


**Fig. 1.** Histogram over the word count of the article descriptions for each category.

we have in our data. Here we first set the text to lowercase and strip it of possible whitespaces at the beginning and end of the description. Subsequently punctuation, possible HTML-code, special characters and digits are removed. Also multiple whitespaces are replaced with single whitespaces. After the first cleaning steps, we remove common English stop words as they do not provide any real value for our feature generation. Next, the goal is to bring all the words to a base form in order increase their frequency and improve the output of feature generation. We choose to rather use lemmatization over stemming as it preserves the context of the word in the sentences. For the lemmatization we calculate the POS tag for every word in our clean text. With a supporting function to get the right tag they are then lemmatized using the WordNetLemmatizer from the nltk library. Afterwards, we execute the preprocessing separately on the training data and on the test data, we save the results into separate csv-files. The process of preprocessing takes up some time as the lemmatization is computationally expensive and because it has to be executed for every article description.

## 4 Feature Generation and Selection

For the feature generation we settle for the Bag-of-Words approaches of TF-IDF and Terms Frequency. Additionally we implement four different state-of-the-art methods for creating word embeddings out of textual data: Word2Vec CBOW, Word2Vec SkipGram, fastText and GloVe. All the algorithms are executed or

trained on the whole training dataset. For the Bag-of-Words approaches we choose to limit the output to the top 10.000 features ordered by term frequency in order to get a smaller matrix and save computation time for later stages of the project. We found little to no decline in evaluation results with classifiers built on these approaches when limiting the number of features to the specified amount.

The Word2Vec and fastText models are trained using the python gensim library whereas the GloVe model is trained using the original implementation in the language C (REF). Afterwards the resulting GloVe vectors and the vocabulary are read into our code. For the word embedding models we set the parameters window = 8, vector_size = 100 and min_count = 5 which are relatively standard values. Here we try to have a vector size which is more on the lower side in order to again save computation time later. The word embedding models encode each of the distinct words in the data as a vector. Since we need an encoding on a document level, we define functions which take a document as input and then get the vector for each existing word in the document from the respective word embedding model and constitute their average representing the document vector. This way we can later also obtain vectors for the test data.

As we can see in Figure 2 the classifiers are initially trained on the vectorizers with the described standard parameters. After finding the best model combinations during classificaton (described in section 5) we perform a hyperparameter tuning of the resulting vectorizer where we try different combinations of hyperparamters to find the local optimum for this combination. We understand that the ideal solution would include this step into the hyperparameter tuning of the classifiers, but due to performance issues we decided on this approach.

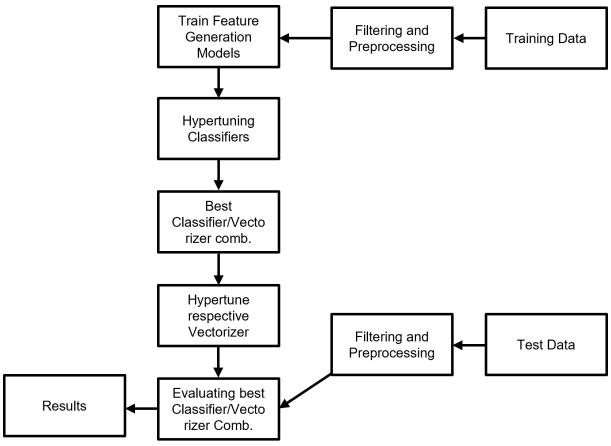


**Fig. 2.** hyperparameter tuning and Evaluation Approach.

# 5 Classification

## 5.1 General Approach

The classification is the core element of our work where preprocessed data gets assigned to the labels from our news dataset. As we use supervised learning algorithms like Support-Vector Machines, K-Nearest Neighbor, Gaussian NB, Random-Forest Classifier, Multilayer Perceptron Classifier and Decision Trees, it is necessary, that we can rely on the data being split in a training dataset and test dataset. Therefore, our classifier can learn from already labeled training data which gives examples on how labeling on the test dataset should look like.

The other dependency for classification in context of text mining, is proper preprocessing. The function param_search() takes a a vector_matrix, which has been calculated in preprocessing, and an estimator dictionary, in which the estimator itself and the hyperparameter searchspaces are packed. It finds the best parameter combination for the input estimator via cross-validated grid search. The grid search itself is again validated via cross-validation and the resulting mean accuracy. The mean accuracy of the best parameter-combination is returned by the function together with the (respectively fitted) model. We use accuracy as our metric to compare the performance of different combinations because our dataset is balanced. Nevertheless our solution could be improved by incorporating other metrics in the process.

The two cross validations in param_search() are a nested cross-validation: the cross-validation used by the gridSearchCV() function is nested in the outer cross-validation to validate and evaluate the best performing parameter combination. The outer cross validation had to be done by hand, because the sklearn.model_selection.cross_val_score() function is only able to compute the mean accuracy of the best performing hyperparameter-combination, but cannot return the fitted model itself. StratifiedKFold() is used as cross-validation-function to keep the class-balance during cross-validation. This is important, because otherwise the model may over-/underfit on classes of random imbalanced folds. Nested cross-validation is used to avoid overfitting in the gridSearch process on the therefore used data.[11] [12]

In the function hyper_tune() the parameter settings are defined and vectorizers from preprocessing are loaded and organized in a matrix. Param_search() is called by the hyper_tune() function until every combination of estimators and vectorizers have been tried out. The best performing model and vectorizer gets returned. We initially tried to use the whole train dataset of 120.000 article descriptions for training our classifiers but quickly realized that computational-wise it is to expensive. That's why we chose to use only 20% of the training data.

## 5.2 Parameter Settings

**K-Nearest-Neighbour** For K-Nearest-Neighbour nearest neighbours parameter (k) plays the most important role in the classification process. In our project, the tuning times of the k-values 10, 12 and 14 were examined.

**Decision Tree** The Decision Tree has to be tuned regarding the splitting and stopping criterion. Commonly used and our chosen methods to determine the impurity of the splits are GINI and information gain. In this article, the CART tuning algorithm shows that allowing trees to grow improves the model performance and optimizes the result[5]. We decide to use more depth values and selected maximum depth of 40, 60, 80 or 100 nodes, which were set as [1,30] in the above mentioned article. Especially for splitting with information gain, the maximum depth is an required argument to ensure that the amount of partitions stay reasonable. As our data is slightly large, we decide on selecting a minimum number of 20,30,40 samples in an internal node to split.

**Gaussian NB** In Gaussian NB, the model performance is determined by the dataset's accordance with Gaussian distributions[6]. Therefore, we don't use any hyperparamaters for Gaussian NB.

**Support Vector Machines** In SVC model, the similarity of two data points is determined by the kernel function. There are many kernel types in scikit-learn such as 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'[7]. Polynomial kernel can have some numerical difficulties such it's value can be infinite[8]. Besides, with 'poly' kernel, we need to consider the degree value as in the hyperparameter tuning. We decided to use 'rbf' and 'sigmoid' kernels. 'C' value determines how our model is tolerant with misclassified data points. We decide to set 'C' value 1 as default. 'gamma' value determines the decision boundaries that define, how well the model covers the different data spreads. We decide to use 'gamma' value as 'scale' and 'auto'.

**Random Forest Classifier** As a default value, maximum features parameter is set to 'sqrt' in scikit-learn and 'auto' option will be removed in 1.3. version. We settle for the value 'sqrt'. N estimators value represents the number of trees in the forest. The optimal performance of the model shows the biggest improvement after growing 100 trees. Therefore, we decide to select this value preferred to the default value and set it with the values 100, 200, 300. [9]. Sample size and node size are not used in our model, which have been found that they have minor impact on model performance[10]. Since training larger trees which results in stable predicitons costs larger time effort, we set the value of the number of trees lower. We use the same maximum depth of 40,60,80 or 100 nodes in decision tree classifier.

**Multi-Layer Perceptron Classifier** For the MLP classifier we decide to tune the parameters activation (Activation function for hidden layer), solver (solver for weight optimization) and hidden_layer_sizes (number of neurons and hidden layers). For the first two parameters we include the default value as well as another two common options to try out. The scikit-learn documentation states that the adam solver performs better on larger datasets whereas the lbfgs

solver works better with smaller sets. For the last parameter we tried following combinations: one/two hidden layer(s) with 50/100 nodes respectively.

## 6 Evaluation and Results

### 6.1 Evaluation Metrics

In order not to run the complete model selection on one machine, we decided to split the evaluation by classifier on different notebooks and run them separately. After that we pick the best performing classifier.

Evaluation metrics are being used in order to see how the trained model performs on unseen records. In our project, we use these metrics to see how good our model classifies the news categories correctly. In AG News dataset, the news categories are distributed equally in both train and test data. Since the dataset is not balanced, accuracy metric would be used to estimate the model's prediction performance. The model performance is evaluated according to these metrics explained below:

The classification report shows precision, recall, and F-1 Score metrics for each label.

Precision represents the number of correctly classified positive examples (TP) divided by the number of examples that are classified as positive (TP+FP). Recall measures the sensitivity of a classifier which shows the number of correctly classified positive examples (TP) divided by the total number of actual positive examples (TP+FN) in the test set. F-1 score combines both precision and recall metrics into one measure and calculates the harmonic mean of precision and recall in order to see the effects of both metrics. Higher results in F-1 score are preferable for good classification[13].

A Confusion Matrix is a good metric in the demonstration of how well the model classifies each label in the dataset. In a Confusion Matrix, which labels are assigned to which labels with its number of observation values makes it clear that the model makes an error at predicting in which labels. Each row represents true label and each column represents predicted label[14].

The ROC Curve is a graphical demonstration of true positive rate (sensitivity,recall) on y-axis against false positive rate (1-specificity) on x-axis. The area under the ROC Curve also determines the model performance and higher AUC means better classification. ROC Curve suits more when the classes are balanced. Contrarily with ROC Curves, the precision-recall curves do not take true negatives into account. The desired model is with higher precision and higher recall, concluding the correct predictions of all labels[15].

### 6.2 Results

In the AG News Classifier project, we search through the results of different machine learning algorithms with different generated features. The goal is to compare and see how the best model can be achieved in which combinations.

**Table 1.** The fitted model results of classifiers on training data

| Classifier | Hyperparameter | Vectorizer | Accuracy |
|---|---|---|---|
| Gaussian NB | - | w2v_skipg | 0.87 |
| KNN | n_neighbors=12 | w2v_skipg | 0.92 |
| Decision Tree | criterion=entropy, max_depth=60, min_samples_split=40 | w2v_cbow | 0.90 |
| Random Forest | max_depth=60, max_features= sqrt, n_estimators=300 | w2v_cbow | 0.99 |
| SVM | gamma=scale, kernel=rbf | tfidf_vectors | 0.98 |
| MLP classifier | activation=relu, hidden_layer_sizes=(50,), solver=adam | tfidf_vectors | 1.0 |

The best combinations together with the hyperparameters can be seen in Table 1. These results are from the fitted model of training set.

As it can be seen in Table 1, the highest accuracy score is obtained by the combination of MLP Classifier and tf-idf vectors on train data.

Subsequently, finding best models with vectorizer, we also applied hyperparameter-tuning on vectorizer's parameters in order to see the achievability of how much the model performance can go further in accuracy score. The most essential point is generalization of our model on unseen data. Eventually, the found best models and hyperparameter combinations are used in the predictions of test data. The model predictive capability is measured with the mentioned metrics above. The classifiers' performance on test data are shown in Table 2.

**Table 2.** The prediction results of classifiers on test data

| Classifier | Vectorizer | Accuracy |
|---|---|---|
| Gaussian NB | w2v_skipg (min_count=5, vector_size=100, window=10) | 0.87 |
| KNN | w2v_skipg (min_count=5, vector_size=200, window=10) | 0.90 |
| Decision Tree | w2v_cbow (min_count=5, vector_size=50, window=5) | 0.81 |
| Random Forest | w2v_cbow (min_count=8, vector_size=200, window=10) | 0.89 |
| SVM | tfidf_vectors (max_features=30000, min_df=1, ngram_range=(1, 2)) | 0.91 |
| MLP classifier | tfidf_vectors (max_features=80000, min_df=1, ngram_range=(1, 2)) | 0.90 |

If the results are evaluated in Table 2, Decision Tree classifier is not able to label classes correctly on unseen data, showing a decrease from 90% to 81%. In particular, true positives in Business and Sci/tech classes are not labeled correctly as good as from other classifiers. Gaussian NB model gets the same accuracy result, but it's performance on Business and Sci/tech classes is also rather low compared to Sports and World classes. kNN showed 2% slight decrease in accuracy score and Business and Sci/tech classes precision and recall scores are lower than other classes. Besides, the precision score of Sci/tech is much lower than its recall score, false positive counts are larger than false negative counts. kNN is better at predicting true positives correctly in the actual positives of Sci/tech. In world class, false negative counts are larger than false positive counts in kNN, meaning that the model can not label actual positives correctly. Moreover, RFC showed 10% decrease in accuracy score due to low prediction

capability in Business,Sci/tech and World classes. MLP is the best performer in train data, but it's prediction capability underperforms the capability of SVM. The Business and Sci/tech classes have the lowest F-1 scores in all classifiers, standing for the least predictable classes.

**Table 3.** The prediction results of SVM on test data

| Classes | Precision | Recall | F-1 score |
|---|---|---|---|
| Business | 0.88 | 0.87 | 0.87 |
| Sci/Tech | 0.88 | 0.89 | 0.88 |
| Sports | 0.94 | 0.98 | 0.96 |
| World | 0.93 | 0.90 | 0.91 |

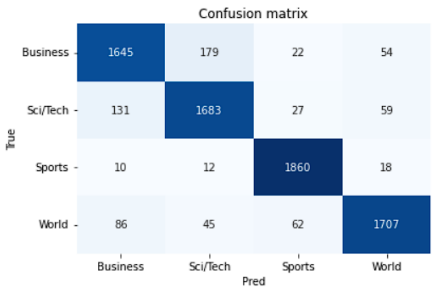


**Fig. 3.** Confusion Matrix for the SVM Classifier.

The most generalizable model is SVM, getting the highest accuracy score on test data. In addition to other classifiers', low performance on Business and Sci/tech classes, SVM is also not good at predicting these classes, as compared to other two classes. False positive counts are much higher than false negative counts in Sports class. The other classes were predicted wrong as Sports.

## 7 Conclusion and Further Developments

In the end we achieve an accuracy score of 91% with an implementation of the SVM classifier built on a TF-IDF vectorizer after trying out the combinations of many different algorithms and different vectorizers. Therefore our classifier would be very practical and useful in the real world. When we searched through other related works with the AG News dataset, the best result is achieved by a paper using a state-of-the-art XLnet language model and a neural network approach achieving an error rate of 4.45 [16]. Compared to our models this is

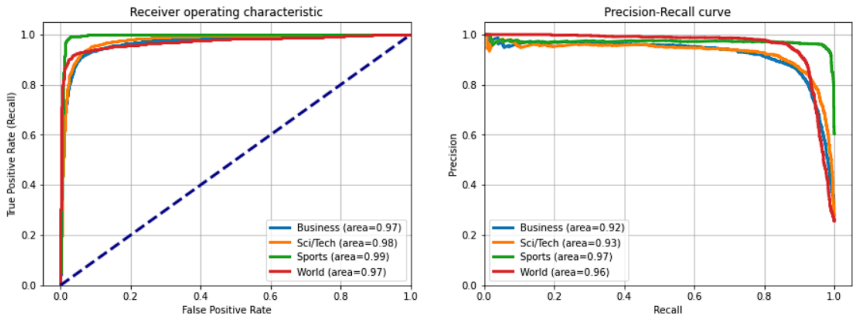


**Fig. 4.** ROC and Precision-Recall Curve for the SVM Classifier.

a much more complex and computationally expensive model. Still we achieve comparable and good results.

Besides, there are other articles about news classification with traditional algortihms, though every article uses different approaches in each step (preprocessing, feature selection, algorithms, hyperparameter settings), different trained classes, different amount of data in the analysis and different text length while training the model, we decide that these articles would not be appropriate to compare.

Possibly we could have achieved different results, if we would have been able to compute more data, which was not possible due to limited computing capacity. Moreover the used test set is not necessarily comparable to real-world data as it is very balanced. Another decision driven by limited computing capacity is that fine tuning on hyperparameters has been progressed only for the combination of best vectorizer and classifier. Other combinations are only tried with standard, small parameter spaces, which may be responsible for worse performance. The downside of our best vectorizer tf-idf would be the lack of semantic analysis or consideration of word position and co-occurrences. For Support-Vector Machines huge data sizes are more challenging to classify. Overlapping classes decrease the accuracy, table 4 shows confusion for Sci/Tech and Business news. Also a potential risk is underfitting when applying SVM, but no evidence have been found which indicates that.

Further work could include using more data for training and testing of the model to get better results. The model can be improved by applying multi-modal classification which uses the heterogeneous information such text, image, video, voice. AG News dataset has only text information. If the images can be collected via URLs from the dataset, multi-modal might be applicable on AG News and thus the model can achieve better accuracy results.

## References


1. Gulli, A., AG's corpus of news articles, (n.d.). URL: http://groups.di.unipi.it/ gulli/AG_corpus_of_news_articles.html
2. https://paperswithcode.com/dataset/ag-news
3. Gavin C. Cawley, Nicola L. C. Talbot, On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation, (2010). URL: https://www.jmlr.org/papers/v11/cawley10a.html
4. Ghawi, R. & Pfeffer, J. Efficient Hyperparameter Tuning with Grid Search for Text Categorization using kNN Approach with BM25 Similarity. *Open Computer Science*. **9** pp. 160 - 180 (2019)
5. Mantovani, R., Horvath, T., Cerri, R., Barbon Junior, S., Vanschoren, J. & Carvalho, A. An empirical study on hyperparameter tuning of decision trees. (2018,12)
6. Yang, L. & Shami, A. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*. **415** pp. 295-316 (2020,11), https://doi.org/10.1016
7. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12:2825–2830. URL: https://scikit-learn.org/stable/modules/svm.html
8. Mantovani, R., Rossi, A., Vanschoren, J., Bischl, B. & Carvalho, A. Effectiveness of Random Search in SVM hyperparameter tuning. *2015 International Joint Conference On Neural Networks (IJCNN)*. pp. 1-8 (2015)
9. Lunetta, K., Hayward, B., Segal, J. & Eerdewegh, P. Screening large-scale association study data: exploiting interactions using Random Forest.. *BMC Genetics*. **5** pp. 32 (2004,2)
10. Probst, P., Wright, M. & Boulesteix, A. Hyperparameters and tuning strategies for random forest. *WIREs Data Mining And Knowledge Discovery*. **9** (2019,1), https://doi.org/10.1002
11. Jason Brownlee, Nested Cross-Validation for Machine Learning with Python, (2020). URL: Nested Cross-Validation for Machine Learning with Python
12. Gavin C. Cawley, Nicola L. C. Talbot, On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation, (2010). URL: https://machinelearningmastery.com/nested-cross-validation-for-machine-learning-with-python/
13. ,Jason Brownlee, Tour of Evaluation Metrics for Imbalanced Classification (2020). URL: https://machinelearningmastery.com/tour-of-evaluation-metrics-for-imbalanced-classification/
14. ,Jason Brownlee,What is a Confusion Matrix in Machine Learning (2016). URL: https://machinelearningmastery.com/confusion-matrix-machine-learning/
15. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12:2825–2830. URL: https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall
16. Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. & Le, Q. XLNet: Generalized Autoregressive Pretraining for Language Understanding. (arXiv,2019), https://arxiv.org/abs/1906.08237