UCLouvain

ÉCOLE POLYTECHNIQUE DE LOUVAIN

LINGI2262

Last Project : A Machine Learning Competition

May 7, 2021

Farnoodian, Nima - 68372000

Professors : Prof. Pierre Dupont

Academic Year : 2020-2021

1 Abstract

The given dataset required several treatment and processing methods. It has proven to be a non-trivial data set since I tried different scenarios in regards to features selection, data normalization, hyper-parameter tuning, and model selection. Moreover, both training and test dataset include a huge number of missing values, which should be carefully imputed as the imputation technique has a direct influence on the performance of the classification task. It is worth mentioning that, due to the curse of the dimensionality of the given dataset, the freedom to choosing pre-processing techniques such as imputation, Normalization, and to select a model becomes limited, thereby a more challenging Machine Learning Task. Finally, I could process the datasets and reduce the number of features to only 22 significant features. Using the processed datasets, I could train an ensemble-learning stacking-classifier that could provide an expected BCR of 82.90% with a 95% Confidence interval of [77.54, 87.07], calculated using .632+ Bootstrap Method. The performance of the chosen model on a valid dataset (20% of the training dataset) is found to be 77.84% , indicating a relatively low variance.

2 Preprocessing Phase

2.1 Imputation

In both training and test datasets, there are 8076966 and 2696441 missing values, respectively. The average number of missing values of a column (feature) in the training dataset is 6.18 with a standard deviation of 2.46. Considering the number of available samples for training and the average number of missing values, we cannot simply drop the columns or samples, which have missing values. Such an approach almost suggests removing all features. Therefore, replacing missing data with valid and meaningful values comes in handy.

Although there are many imputation techniques in the market, I had to just apply statistical imputation techniques (e.g., mean and mode) due to the high dimensionality. For scalar features, I replaced the missing values with mean values, and for the categorical features, I used the frequency technique for imputation—mode is used for the categorical features. These two statistical imputation approaches were found more effective in training the given data.

2.2 Removing Constant and Quasi-Constant Features

The given dataset consists of many constant and quasi-constant features, which provide no information that allows a machine learning model to discriminate or predict a target. Therefore, removing these features helps speed up training and model selection as many useless features may get removed after processing. Quasi-constant features are those that show the same value for the great majority of the observations of the dataset. To identify Quasi constant features, I used a variance threshold of 0.01. Indeed, I removed all scalar features whose variances are equal to or below 0.01. After removing all these constant and quasi-constant features, the datasets only contain 541682 features six out of which are the categorical features.

2.3 Feature Standardization

To help to train a model like Multi-layer perceptron faster and in a more reliable manner, I normalized all the scalar features between -1 and 1 (subtract the mean and divide by the standard deviation for each feature).

2.4 Categorical Variables

The categorical features in the given dataset represent the viewpoint index of the 6 pictures. To have concrete dataset, I encoded all categorical features using a one-hot (aka 'one-of-K' or 'dummy') encoding scheme provided by Scikit Learn one hot encoder.

2.5 Feature Selection

Although I reduced the number of features to 541682, I tried different feature selection techniques such as K-best feature selection using Mutual Information and Correlation, Boruta Feature selection,

Autoencoder Feature selection with different configurations, and model-based feature selection using Decision Trees, Extra Trees classifier, and Linear Support vector Machine (SVM). The winner among all these techniques was model-based feature selection using a SVM model with the following parameters : $C = .01, penalty = "l1", dual = False, max - iter = 2000$. The winner was selected based on its performance on a baseline model such as Logistic Regression—none of the other approach led to a better performance. It is worth mentioning that the above SVM model was initially found using a grid search, and then the relative features were extracted using *SelectFromModel* method in Scikit Learn. After feature selection, the resulting datasets (training and test) only have 22 relevant features. In order to ensure the relevancy of the selected features, I tested them using Boruta Automated feature selection algorithm in [1]. According to Boruta, every selected feature in the dataset is relevant to the target variable in at least a minimal way. Figure 2.1 illustrates the correlation among the features.

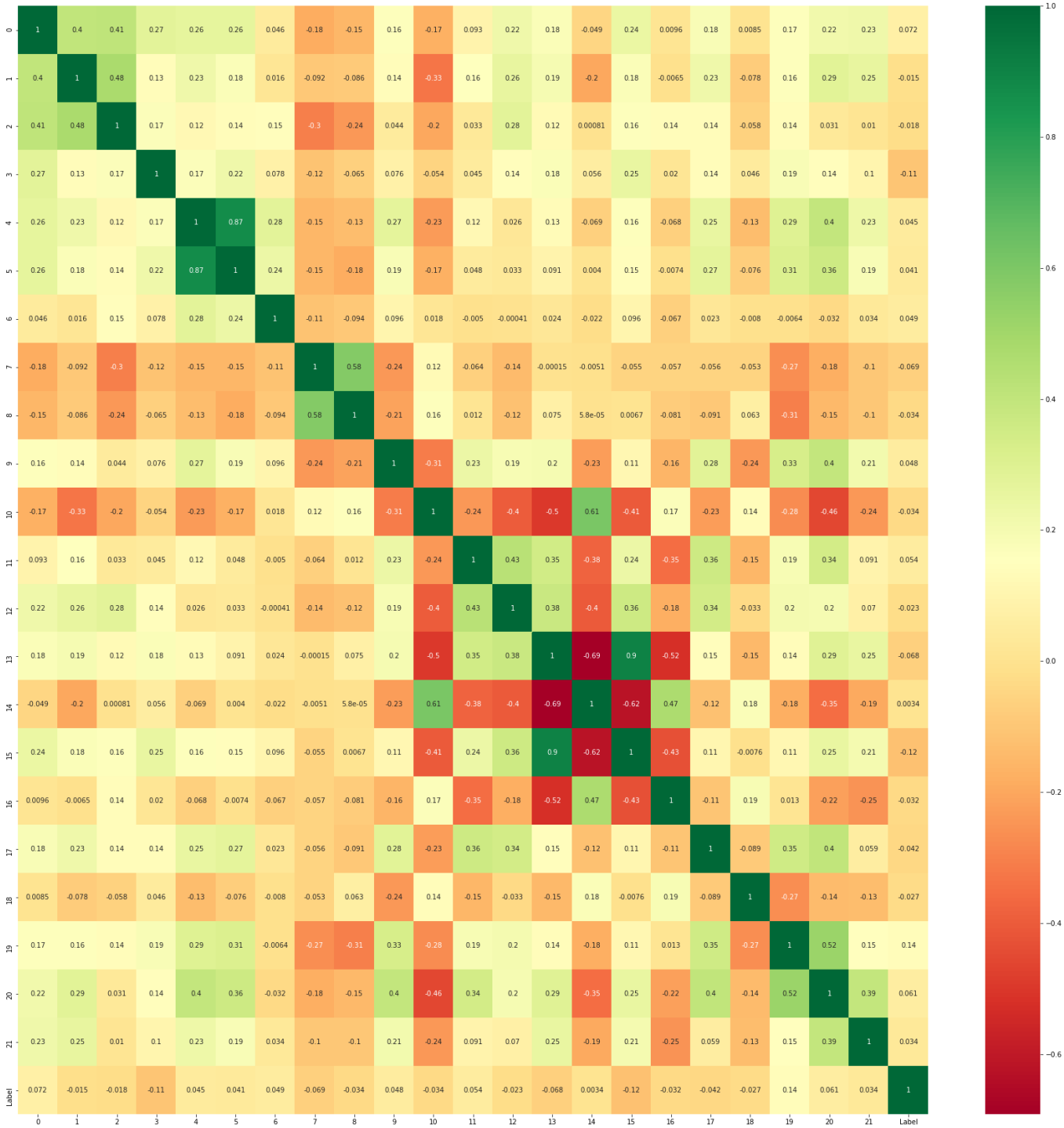

FIGURE 2.1 – Features correlation matrix. The last feature is the target label.

3 Model

For training the data, I picked ensemble learning techniques such as Voting Classifier and Stacked Classifier as I found single classifiers less reliable. The aim of using the ensemble learning techniques was to reduce the variance as the single classifiers like Decision Tree, Random Forest, etc., were prone to overfitting on the prepossessed dataset — the bias on the valid dataset was too high compared to the training set. To build the ensemble classifiers and compare them, I first splited the training dataset into two training (80%) and valid datasets (20%). The chosen single classifiers are Multi-layer Perceptron (MLP), Support Vector Machine (SVC), and Logistic Regression (LR). I fed the Voting and stacked classifiers with the best possible trained model of each single model. The approach that I used to build Voting Classifier and Stacked Classifier is as follow. For each single classifier, I initially tried to find the best configuration using the grid search. Take for example, Best-MLP, Best-SVC, and Best-LG as the best found models for MLP, SVC, and LG, respectively. For MLP that involves huge stochasticity in its weights, I repeated training Best-MLP several times and picked the best one as my MLP model. Indeed, my MLP is the best of Best-MLP. In what follows, you will see the best configuration found for each model :

```
Best_MLP=MLPClassifier(activation='tanh', alpha=0.0001, batch_size='auto',
    beta_1=0.9,beta_2=0.999, early_stopping=False, epsilon=1e-08,
    hidden_layer_sizes=(10, 20, 4), learning_rate='adaptive',
    learning_rate_init=0.001, max_fun=15000, max_iter=100,
    momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
    power_t=0.5, random_state=None, shuffle=True, solver='adam',
    tol=0.0001, validation_fraction=0.1, verbose=False,
    warm_start=False)

Best_SVC=SVC(C=10, break_ties=False, cache_size=200, class_weight=None,
    coef0=0.0, decision_function_shape='ovr', degree=3, gamma=0.001,
    kernel='rbf',max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)

Best_LG=LogisticRegression(C=0.01, class_weight=None, dual=False,
    fit_intercept=True,
    intercept_scaling=1, l1_ratio=None, max_iter=100,
    multi_class='auto', n_jobs=None, penalty='l2',
    random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
    warm_start=False)
```

Using the above models, I created Voting Classifier with hard voting. The corresponding weights for the classifiers are 0.5 for Best_MLP, 0.25 for Best_SVC, and 0.25 for Best_LG. Similarly, I created my Stacked Classifier with the following architecture : (First Classifier : Best_MLP, Second Classifier : Best_SVC, meta classifier : Best_LG). For the stacked classifier, no weight is considered.

3.1 Evaluation

To compare and pick the best model, I used .632+ Bootstrap method in [2] as it is more robust to overfitting. To have reliable estimates, the .632+ Bootstrap draws 200 bootstrap samples as being sufficient. The score for comparing the models is Balanced Classification Rate (BCR) as suggested. The expected BCR is the average of the scores given by the .632+ Bootstrap. Not only were the models compared according to the .632+ Bootstrap, they were also tested based on their performance on the valid test. The table below shows the results.

	Expected BCR	95% Confidence interval	BCR on Valid Test
Voting Classifier	82.59%	[76.66%, 87.43%]	75.59%
Stacked Classifier	82.90%	[77.54%, 87.07%]	77.84 %

As shown in the table, almost both models have similar performance —though Stacked Classifier is a bit better. I have chosen the stacked classifier as my final classifier for predicting the test dataset since its confidence interval and achieved BCR on the valid datasets are preferable, and it does not depend upon any weight. Indeed, due to weights in the Voting Classifier, the Voting Classifier needs more parameter tuning.

Bibliographie

[1] *Boruta all-relevant feature selection method.* https ://github.com/scikit-learn-contrib/boruta_py

[2] $bootstrap_point632_score.http : //rasbt.github.io/mlxtend/user\_guide/evaluate/bootstrap\_point632\_score/$