Supervised Learning Report

Maja Świerk and Jakub Konieczny

Abstract

The goal of this assignment is to predict the level of damage to buildings caused by the 2015 Gorkha earthquake in Nepal, based on aspects of building location and construction. For this cause, we used several ML methods: KNN, Decision Tree, Random Forest, XGBoost, and CatBoost. What is more, we used AutoML as a benchmark for the algorithms of our choice. We've achieved the best results using CatBoost algorithm, in which the f1 score amounts to 0.7299.

Keywords: sklearn, Gorkha earthquake, ML, AI

Contents

1 Introduction **2**

2 Results **2**

3 Exploratory Data Analysis **2**

4 Outlier detection **3**

5 Feature Selection **3**
5.1 PCA 4
5.2 Feature engineering 4

6 Methods used **4**
6.1 Naive Bayes 4
6.2 KNN 4
6.3 Decision Tree 5
6.4 Random forest 5
6.5 XGBoost 5
6.6 CatBoost 5

7 AutoML **6**

8 Extensions **6**
8.1 Outliers detection 6
8.2 Feature selection 6
8.3 Model selection 6
8.4 Searching for hyperparameters 6

9 Conclusions **6**

1 Introduction

This report analyzes the process of building the machine learning model, which can predict with the highest possible score the level of damage to buildings caused by the 2015 Gorkha earthquake in Nepal. Predictions are made based on the buildings' location and construction. The data used for this study was collected through surveys by Kathmandu Living Labs and the Central Bureau of Statistics, which works under the National Planning Commission Secretariat of Nepal. That data set is one of the biggest post-disaster data sets ever collected. We predict the *damagegrade* variable, representing the level (1,2 or 3) of damage by which the building was affected. For predicting we've used several Machine Learning algorithms: KNN, Decision Tree, Random Forest and XGBoost. As a benchmark, we have decided to use AutoML.

2 Results

With the CatBoost algorithm, using following parameters:

- n_estimators = 1000
- learning_rate = 0.4
- depth = 6

we have managed to achieve the best score which amounts to 0.7299.

3 Exploratory Data Analysis

For the Exploratory Data Analysis we used a tool called pandas profiling. We saw that there are no null values in whole data set, moreover in general quality of the data is good. We discovered an interesting fact that there is a high correlation between roof_type and damage_grade, which might be helpful in our predictions. There were some columns with "zeros" and "correlation" alerts, but we decided those are not problems in our case. We also saw that there are some duplicated rows, however we believe that they are representing real state of the world and we should keep them as they are.

4 Outlier detection

As our data set contains categorical features, the first thing we had to do, was encoding them as an integer array. For that, we used the OrdinalEncoder function. The results are columns with integers instead of characters.

We tried to use the DBSCAN for outlier detection, however we did not manage to calculate the best parameters, due to the size of the dataset. We decided to try another technique for anomaly detection.

After that, we used the Isolation Forest algorithm for finding the anomaly score of each sample. With contamination set to *auto*, the algorithm found about 20 000 outliers, which noticeably worsen our data. By trial and error we've discovered, that contamination set to 0.4 gives the best results.

```
from sklearn.preprocessing import OrdinalEncoder
from sklearn.ensemble import IsolationForest

oe = OrdinalEncoder()

clf = IsolationForest(contamination=0.04)
clf.fit(oe.fit_transform(X_train))

pred = clf.predict(oe.fit_transform(X_train))
metrics_df=pd.DataFrame(pred,columns=['anomaly'])
```

5 Feature Selection

For feature selection we combined 2 methods *SelectFromModel* with random forest as an estimator and *SelectKBest()*. For the select k best we've tried to use different techniques such as Chi-Squared Statistic and Mutual Information Statistic, but they gave worse results than using SelectKBest() without specifying the technique. There is a snippet from the code:

```
oe = OrdinalEncoder()
sel = SelectFromModel(RandomForestClassifier(), max_features=15).fit(oe.fi

selected_feat= X_train.columns[(sel.get_support())]
selected_feat

sb = SelectKBest().fit(oe.fit_transform(X_train), y_train)
feat = X_train.columns[(sb.get_support())]
selected = list(selected_feat) + list(feat)
selected = list(set(selected))
```

5.1 PCA

What's more, we tried to use PCA instead of feature selection, but its outcome was not satisfactory. The number of dimensions was still high, the highest variance ratio was 0.09, and the rest was around 0.01. Finally, models fitted with PCA data were not performing well.

5.2 Feature engineering

We tried scaling data, however, depending on the model, it was not improving the outcome and even in some cases it was making it worse. An example of this is a KNN algorithm which was losing about 0.04 points in f1 score. That is surprising because KNN is an algorithm that is sensitive in terms of standardization.

6 Methods used

We decided to use 6 different classification algorithms. Three of them were basic classification techniques: Naive Bayes, K Nearest Neighbours and Decision Tree and three ensembles and boosting algorithms: Random forest, XGBoost and CatBoost.

6.1 Naive Bayes

From the beginning results from this algorithm were very poor (f1 score around 0.3). Because of that, we did not make any further optimizations of NB classification.

6.2 KNN

KNN was getting surprisingly good results, in some cases it was outperforming the ensembles. The f1 score results with default arguments were around 0.70. For optimization, we decided to use Grid Search with the following specification:

```
parameters = {'n_neighbors': [3, 5, 9],
              'leaf_size': [10, 30, 70],
              'n_jobs': [-1],
              'algorithm': ['ball_tree', 'kd_tree']}

gcf = GridSearchCV(knn, parameters, verbose=3, cv=3)
gcf.fit(X_train, y_train)
```

However, our optimization had only a little impact on the results.

6.3 Decision Tree

Decision Tree was also giving quite good results, nevertheless, they were not as good as one from the KNN, so we abandoned its development.

6.4 Random forest

Random forest was giving almost as good results as XGBoost and CatBoost, but still they were lower (0.71 vs 0.72). Therefore we decided not to include it in our later predictions.

6.5 XGBoost

Although XGBoost gave good results (f1 score around 0.72), it was executing so slowly, that further changes made for improving the results were taking too long. We've eventually decided to no longer work on this algorithm, as others are faster and give just as good results.

6.6 CatBoost

CatBoost is the other well-performing algorithm (f1 score around 0.73). Being at the same time quite fast in returning the results, it makes it the best algorithm we've used up to this point.

```
from catboost import CatBoostClassifier

cbc = CatBoostClassifier()

cbc.fit(X_train, y_train, verbose=False)
y_cbc = cbc.predict(X_test)
```

For the parametrization firstly we used the Randomized Search with hyperparameters from different ranges, so that we more or less know, which of them to choose. Later on, we applied Grid Search with data around the best hyperparameters obtained from the previous search. Doing it without performing a Randomized Search before would take too much time.

```
parameters = {'depth': [ 6, 8],
              'learning_rate': [ 0.4, 0.8],
              'n_estimators': [170, 300, 600, 1000]
              }

gcf = GridSearchCV(cbc, parameters,  verbose=3, cv = 3)
gcf.fit(X_train, y_train)
```

The best parameters chosen by the algorithm are:

- depth = 6
- learning_rate = 0.4
- n_estimators = 1000

With that algorithm we achieved our best result of 0.7299 in f1 score in the data-driven competition.

7 AutoML

Moreover, we applied an AutoML library called PyCaret. It was done out of curiosity and also as a benchmark for our results. We run it without any preprocessing. We achieved f1 score of 0.7254, which is lower than the result from CatBoost.

8 Extensions

The program may be improved in many fields.

8.1 Outliers detection

For the outliers detection there may be used other method and also it could be done more systematically. Moreover, it should be provided with a check if the detected anomalies must be removed or not.

8.2 Feature selection

Another variation of our program could be selecting other features or applying another type of encoding for categorical variables.

What is more, the next way of improving results could be done by applying different kinds of engineering between variables.

8.3 Model selection

In terms of Model Selection there is a low chance of any algorithm doing better than those which we have. Although we could try to apply Artificial Neutral Network.

8.4 Searching for hyperparameters

The last place for development is searching for other hyperparameters. There is a possibility of searching with different hyperparameters and with different values.

9 Conclusions

After implementing various algorithms, the one that performed the best was CatBoost. Its f1 score amounts to 0.7299. Others were either too slow or their score was much lower.
We believe that further working on feature selection and exploring other hyperparameters with different values could improve the results even more.

References

[1] Machine Learning Mastery: https://machinelearningmastery.com/
[2] Scikit learn docs: https://scikit-learn.org/stable/user_guide.html
[3] Towards Data Science: https://towardsdatascience.com/
[4] PyCaret docs: https://pycaret.gitbook.io/docs/
[5] Kaggle: https://www.kaggle.com/learn