Tabular Data Science - Final Course Project

Ohad Cohen ID. (315996785), Itay Lotan (ID. 308453935)

Submitted as final project report for the Tabular Data Science course, BIU, 2023

Abstract

This project introduces an automatic, user-friendly API for transforming categorical features for classification models. It allows for the automatic identification of candidate features and the evaluation of various transformation methods. The different combinations of feature transformations are evaluated according to the metric that was defined by the user, and compared to the baseline metric value. The transformations combinations that yielded the best results are displayed to the user clearly and informally, allowing him to choose the combination that makes most sense, using his domain knowledge as well. This process aims to save the user considerable time and effort, and does not require any prior knowledge on the dataset.

1 Problem Description

The proposed algorithm aims to improve the EDA and feature engineering stage of the data model development process. Specifically, it focuses on the challenge of transforming categorical features to improve the performance of classification models. The motivation for these transformations is to eliminate the possibility of the model relying on ordinal relations between feature values, which will not make sense and might cause overfitting (as was described in class). There are several issues with the manual process of applying these categorical transformations:

- This process often requires prior knowledge from the model developer, and if he doesn't know the meaning of a certain feature, he might miss the fact that it makes more sense to treat its data as categorical.
- When working on complicated datasets with a lot of features, this process might be tedious and time consuming.
- The decisions that the model developer makes are often subjective and don't rely on accurate measurements for the impact it'll have on the model's performance.
- The most common method for these transformations is 'One hot encoding'. However, there are a lot of other methods that are less known to the community, but might come off handy for certain problems.

2 Solution Overview

2.1 Solution details

To handle the issues that were described in Section 1.1, we suggest an automatic approach for finding the best combinations of categorical feature transformations on a given dataset. This solution can be broken into these logical steps:

- Input description
  - A dataset (including the predicted feature, a.k.a 'y').
  - The method of categorical transformation (currently supported methods are 'one hot', 'weight of evidence' and 'leave one out').
  - The metric(s) that will be used for evaluating the transformation combinations (currently supported metrics are 'accuracy', 'f1', 'precision' and 'recall').
  - Various optional settings - please see our API Python documentation for more details.
- Find the candidate features for categorical transformations. A feature is considered a candidate if it meets one of these conditions:
  - The absolute number of distinct values it has is lower than a threshold (the threshold is configurable, with the default being 10).
  - The number of distinct values it has is low when compared to the dataset size (the threshold is configurable, with the default being 1%).
- Calculate all the possible combinations of candidate features to transform:
  - This includes the 'empty' combination, which doesn't transform any candidate feature. We will refer to it as the 'baseline' combination from now on.
  - Each candidate can be transformed or not transformed, so the total number of combinations is $2^{number\_of\_candidates}$.
  - Notice that the candidate thresholds that were described above can affect the number of combinations drastically.
- For each of the transformations combination, perform these steps in order:
  - Apply the categorical transformation for all the features of the current combination. Reminder: the method of the transformation was defined in the input.
  - Apply basic pre-processing on the dataset, which includes features normalization and imputation.
  - Split the dataset using K-fold cross validation process (to achieve maximum stability of the evaluation results).

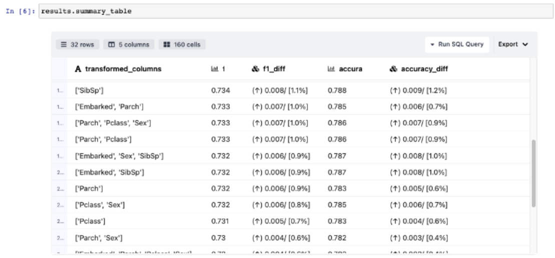
- For every fold (which includes a train and a test set), train a classifier, and evaluate its performance on the test set (the list of evaluation metrics is configurable, with the default being ‘f1’ and ‘accuracy’). The type of classifier and its parameters are also configurable, with the default being sklearn’s logistic regression implementation.

- Return a results object to the user.

2.2 User interface

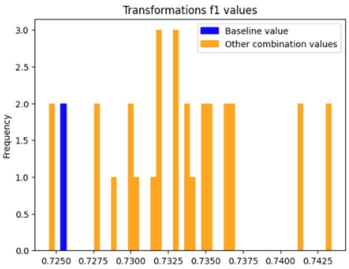
- Because the described algorithm might take time, a progress bar of the remaining evaluated combinations is displayed to the user during its execution.
- As described above, the API that we developed returns a results object at the end of its execution. This object holds the following properties:
  - A list of the candidate features that were detected
  - The evaluation metrics of the baseline model
  - A summary table (dataframe)
    - The table contains each combination and the values of its evaluation metrics.
    - The table also contains a diff column per evaluation metric, which shows its improvement/decrease from the baseline metric value (the diffs are shown both in absolute values and percentages).
    - The table is sorted by the first metric (descending), so the user is able to pick its top rows and consider which combination to choose.
- If the argument *return_rich_table* is set to true, the summary table will be converted from a dataframe to a datapane DataTable, which is very convenient to work with from a Jupyter notebook.
- If the argument *verbose* is set to true, these additions will be made:
  - The summary table will also contain a column for the kfold evaluation std value per metric.
  - In addition to the results object, a histogram of the first evaluation metric values will be plotted.

Here is an example of the results interface (over the Titanic dataset):



```
In [6]: results.summary_table
```

	transformed_columns	1	f1_diff	accura	accuracy_diff
1...	['SibSp']	0.734	(↑) 0.008/ [1.1%]	0.788	(↑) 0.009/ [1.2%]
1...	['Embarked', 'Parch']	0.733	(↑) 0.007/ [1.0%]	0.785	(↑) 0.006/ [0.7%]
1...	['Parch', 'Pclass', 'Sex']	0.733	(↑) 0.007/ [1.0%]	0.786	(↑) 0.007/ [0.9%]
1...	['Parch', 'Pclass']	0.733	(↑) 0.007/ [1.0%]	0.786	(↑) 0.007/ [0.9%]
1...	['Embarked', 'Sex', 'SibSp']	0.732	(↑) 0.006/ [0.9%]	0.787	(↑) 0.008/ [1.0%]
2...	['Embarked', 'SibSp']	0.732	(↑) 0.006/ [0.9%]	0.787	(↑) 0.008/ [1.0%]
2...	['Parch']	0.732	(↑) 0.006/ [0.9%]	0.783	(↑) 0.005/ [0.6%]
2...	['Pclass', 'Sex']	0.732	(↑) 0.006/ [0.8%]	0.785	(↑) 0.006/ [0.7%]
2...	['Pclass']	0.731	(↑) 0.005/ [0.7%]	0.783	(↑) 0.004/ [0.6%]
2...	['Parch', 'Sex']	0.73	(↑) 0.004/ [0.6%]	0.782	(↑) 0.003/ [0.4%]



3 Experimental Evaluation

In order to evaluate our solution, we tested it on 4 classification problems over 4 well known public datasets (as required) - Kaggle's Titanic survival, Music Genre and Insurance costs datasets, and UCI's Bank Marketing dataset. On each of these datasets, we did not apply any modifications (besides minor adjustments to the target values in some cases), and used our API directly on them to find the recommended categorical feature transformations. We used the default configuration setup for these experiments, to keep this report focused. The main evaluation metric that we considered is the f1 score, since it's more challenging to improve and is the community standard in classification problems.

3.1 Experiment 1 - The Titanic Survival Dataset

The Titanic dataset is a classical dataset that is commonly used in the data science field. Here are the results that were produced (the top 3 combinations for each transformation type):

Transformation type	**transformed_columns**	**f1**	**f1_diff**	**f1_std**	**accuracy**	**accuracy_diff**	**accuracy_std**
Baseline	[]	0.712	(-) 0 [0%]	0.023	0.768	(-) 0 [0%]	0.012
One hot encoding	['Embarked', 'Pclass', 'SibSp']	0.739	(↑) 0.027/ [3.7%]	0.021	0.792	(↑) 0.025/ [3.2%]	0.015
One hot encoding	['Embarked', 'Pclass', 'Sex', 'SibSp']	0.739	(↑) 0.027/ [3.7%]	0.022	0.792	(↑) 0.025/ [3.2%]	0.015
One hot encoding	['Pclass', 'Sex', 'SibSp']	0.737	(↑) 0.025/ [3.5%]	0.022	0.791	(↑) 0.024/ [3.1%]	0.014
Weight of evidence	['Embarked', 'Parch', 'Pclass', 'Sex', 'SibSp']	0.732	(↑) 0.02/ [2.8%]	0.027	0.781	(↑) 0.013/ [1.8%]	0.02
Weight of evidence	['Embarked', 'Parch', 'Pclass', 'SibSp']	0.732	(↑) 0.02/ [2.8%]	0.027	0.781	(↑) 0.013/ [1.8%]	0.02
Weight of evidence	['Parch', 'Pclass', 'Sex', 'SibSp']	0.73	(↑) 0.018/ [2.5%]	0.025	0.779	(↑) 0.011/ [1.5%]	0.019
Leave one out	['Parch', 'Pclass', 'Sex', 'SibSp']	0.73	(↑) 0.018/ [2.6%]	0.024	0.78	(↑) 0.012/ [1.6%]	0.016
Leave one out	['Parch', 'Pclass', 'SibSp']	0.729	(↑) 0.016/ [2.3%]	0.023	0.779	(↑) 0.011/ [1.5%]	0.015
Leave one out	['Embarked', 'Parch', 'Pclass', 'Sex', 'SibSp']	0.728	(↑) 0.016/ [2.2%]	0.024	0.778	(↑) 0.01/ [1.3%]	0.017

* Candidate features for categorical transformation: 'Pclass', 'Sex', 'SibSp', 'Embarked', 'Parch'.

We were pleased to see that the algorithm did find combinations of transformations that had a positive impact on both the f1 score and the accuracy, in all 3 methods of categorical transformations. We can see that for this basic dataset, the classic one hot encoding method was the most efficient and yielded the best results. It may have to do with the relative small size of this dataset, so the two more advanced transformations may caused overfitting which hurt the performance.

3.2 Experiment 2 - The Music Genre Dataset

The Music Genre dataset is a collection of audio features extracted from various songs of different music genres. We included this dataset to test the performance of our algorithm on multi class problems. Here are the results of the top 3 combinations of the supported methods methods (the weight of evidence method isn't implemented for the multi class case):

* Candidate features for categorical transformation: 'mode', 'popularity', 'key', 'obtained_date'.

Unfortunately, In this case we did not find any transformation combination that yielded an improvement over the baseline results. This can have multiple explanations. First of all, the performance of the baseline setting is already very high. This may imply that there is not much room left for improvement, and if there is it can be achieved in other ways. Additionally, there might be some adjustments that we should have in order to fully support the multiple classes.

Transformation type	transformed_columns	f1	f1_diff	f1_std	accuracy	accuracy_diff	accuracy_std
Baseline	[]	0.958	(-) 0 [0%]	0.001	0.958	(-) 0 [0%]	0.001
One hot encoding	['obtained_date']	0.958	(-) 0 [0%]	0.001	0.958	(-) 0 [0%]	0.001
One hot encoding	[]	0.958	(-) 0 [0%]	0.001	0.958	(-) 0 [0%]	0.001
One hot encoding	['key', 'mode']	0.958	(-) 0 [0%]	0.001	0.958	(-) 0 [0%]	0.001
Leave one out	['key', 'mode', 'obtained_date']	0.958	(-) 0 [0%]	0.001	0.958	(-) 0 [0%]	0.001
Leave one out	[]	0.958	(-) 0 [0%]	0.001	0.958	(-) 0 [0%]	0.001
Leave one out	['mode']	0.958	(-) 0 [0%]	0.001	0.958	(-) 0 [0%]	0.001

For example, we may have better understood the gap if we would break this task into 10 'one vs all' sub tasks, and analyse the algorithm's output on each sub task.

3.3 Experiment 3 - The Insurance Costs Dataset

This dataset contains the details of insurance members, and their charges amount over a shared period of time. The original task of this dataset was to predict the charges of new members using a regression model, but we converted it to a classification task by defining a charges threshold. Here are the results that were produced (the top 3 combinations for each transformation type):

Transformation type	transformed_columns	f1	f1_diff	f1_std	accuracy	accuracy_diff	accuracy_std
Baseline	[]	0.681	(-) 0 [0%]	0.043	0.882	(-) 0 [0%]	0.02
One hot encoding	['children']	0.686	(↑) 0.005/ [0.7%]	0.044	0.886	(↑) 0.004/ [0.4%]	0.017
One hot encoding	['children', 'region']	0.686	(↑) 0.005/ [0.7%]	0.044	0.886	(↑) 0.004/ [0.4%]	0.017
One hot encoding	['children', 'sex']	0.686	(↑) 0.005/ [0.7%]	0.044	0.886	(↑) 0.004/ [0.4%]	0.017
Weight of evidence	[]	0.681	(-) 0 [0%]	0.043	0.882	(-) 0 [0%]	0.02
Weight of evidence	['sex']	0.681	(-) 0 [0%]	0.043	0.882	(-) 0 [0%]	0.02
Weight of evidence	['smoker']	0.681	(-) 0 [0%]	0.043	0.882	(-) 0 [0%]	0.02
Leave one out	[]	0.681	(-) 0 [0%]	0.043	0.882	(-) 0 [0%]	0.02
Leave one out	['smoker']	0.681	(-) 0 [0%]	0.043	0.882	(-) 0 [0%]	0.02
Leave one out	['region']	0.68	(↓) -0.001 [-0.2%]	0.047	0.881	(↓) -0.001 [-0.1%]	0.022

* Candidate features for categorical transformation: 'region', 'sex', 'smoker', 'children' In this case either, the algorithm did not find any transformation combination that yielded an improvement over the baseline results. It may imply that the approach that we took is a bit too naive to handle every dataset successfully. It will be interesting to debug and try to understand why it can't find any improvement in this case. Maybe it can come up handy for this problem in a more advanced stage, after a few pre-processing steps are already applied on the dataset.

3.4 Experiment 4 - The Bank Marketing dataset

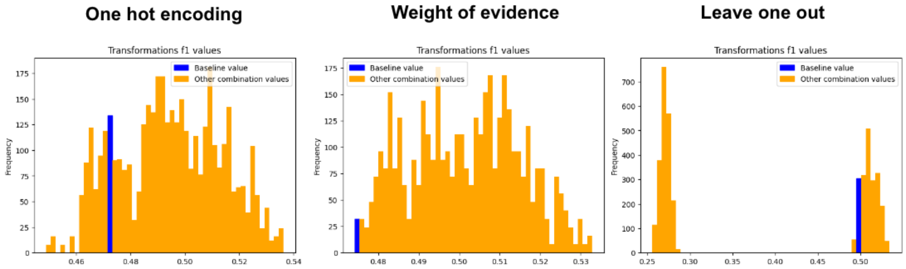
The goal of this classification task is to predict whether a potential client will subscribe to the term deposit product offered by the bank. We used the prepared smaller version of the dataset, which consists of 4,000 rows and 17 columns. Here are the results that were produced (the top 3 combinations for each transformation type):

* Candidate features for categorical transformation: 'marital', 'poutcome', 'housing', 'contact', 'campaign', 'job', 'default', 'month', 'previous', 'loan', 'education', 'day'. Notice the relatively large number of candidate features, which are resolved in 4,096 combinations to search through (execution time of 1̃.5 hours per transformation method).

Transformation type	**transformed_columns**	**f1**	**f1_diff**	**f1_std**	**accuracy**	**accuracy_diff**	**accuracy_std**
Baseline	[]	0.499-0.472	-	-	-	-	-
One hot encoding	['contact', 'day', 'default', 'education', 'job', 'marital', 'month', 'poutcome']	0.536	(↑) 0.064/ [13.6%]	0.012	0.841	(↑) 0.04/ [4.9%]	0.007
One hot encoding	['contact', 'day', 'education', 'job', 'loan', 'marital', 'month', 'poutcome']	0.536	(↑) 0.064/ [13.6%]	0.012	0.841	(↑) 0.04/ [4.9%]	0.007
One hot encoding	['contact', 'day', 'default', 'education', 'housing', 'job', 'marital', 'month', 'poutcome']	0.536	(↑) 0.064/ [13.6%]	0.012	0.841	(↑) 0.04/ [4.9%]	0.007
Weight of evidence	['contact', 'day', 'education', 'job', 'loan', 'marital', 'month', 'poutcome']	0.533	(↑) 0.058/ [12.1%]	0.015	0.84	(↑) 0.031/ [3.8%]	0.007
Weight of evidence	['contact', 'day', 'default', 'education', 'housing', 'job', 'marital', 'month', 'poutcome']	0.533	(↑) 0.058/ [12.1%]	0.015	0.84	(↑) 0.031/ [3.8%]	0.007
Weight of evidence	['contact', 'day', 'default', 'education', 'job', 'marital', 'month', 'poutcome']	0.533	(↑) 0.058/ [12.1%]	0.015	0.84	(↑) 0.031/ [3.8%]	0.007
Leave one out	['day', 'education', 'loan', 'month', 'poutcome', 'previous']	0.534	(↑) 0.035/ [7.0%]	0.018	0.841	(↑) 0.019/ [2.4%]	0.006
Leave one out	['day', 'loan', 'month', 'poutcome']	0.533	(↑) 0.034/ [6.8%]	0.014	0.841	(↑) 0.019/ [2.3%]	0.007
Leave one out	['day', 'month', 'poutcome']	0.533	(↑) 0.034/ [6.8%]	0.014	0.841	(↑) 0.019/ [2.4%]	0.007

In this case, we do see a significant improvement in the model performance (up to 13.6% improvement in the f1 score). This was encouraging to see, especially after the results of the previous 2 experiments. In addition, this task is more challenging than the previous ones, so the fact that our algorithm was effective on it made us believe it'll be better fitted for real world problems. On a less positive note, we noticed that this experiment was much less stable than the previous ones. Even the baseline setup yielded very different results in different runs (the f1 score ranged between 0.472 to 0.499, which is 5% gap). This occurred even though we used kfold cross validation, and set the same global random seed. This is probably due to the complexity of this task. So, the f1 13.6% improvement is probably too optimistic, and should be normalised to 10% (which is still an impressive improvement).

Because this experiment was especially interesting, we decided to show the histograms of the f1 scores of each method as well:



One hot encoding

Weight of evidence

Leave one out

It's interesting to note that while the 'one hot encoding' transformation had mixed effect on the f1 scores, the 'weight of evidence' transformations had almost only positive impact on the scores, and the 'leave one out' transformations had mixed effects, but the worse the most of negative impacts were much stronger then in any other method.

4 Related Work

We drew inspiration from several lessons from this curse. Firstly, from the introduction notebook that was shown in the first lesson, where the importance of the categorical features transformation were discussed. Secondly, for the idea of displaying the metric scores plots for some of our experiments in a histogram (that was mentioned in the EDA slides). In addition, it helped us to choose the Logistic Regression model for our experiments, as a model that will provide reliable results for the classification tasks that we focused on (which was discussed in the Linear Regression slides).

Our starting point comes from [1], an article that discusses various techniques for encoding categorical features into numeric features in a machine learning model. We based our idea on the importance of these categorical transformations that was described in this article and in class. The author starts by explaining the importance of encoding categorical features and the limitations of the traditional one-hot encoding method. The article then dives into 17 techniques, including ordinal encoding, polynomial encoding, binary encoding, etc. We decided to use these three transformations - One hot encoding, Weight of evidence and Leave one out for our project, because they best suited for the classification that we wanted to focus on.

In [2], a compact one-hot encoding method for transforming categorical features in fraud detection applications in a distributed environment is presented. The proposed method addresses the issue of high-dimensional feature space and sparsity of one-hot encoding by reducing the number of dimensions and compressing the encoded features using a compact binary representation. The authors evaluate the proposed method on two real-world datasets, and the results show that it outperforms traditional one-hot encoding. This paper encouraged us to try our algorithm on tasks with a relatively larger number of features that we initially planned to (more then 5-8 features).

Another source of related work is proposed in [3], an automated feature engineering and selection tool named "AutoFeat" that can be used for various machine learning applications. The tool is built on the idea of combining simple mathematical operations to create complex features. It automatically identifies potential relationships between input features mainly by the correlation between them and to the target feature, and generates new features through a combination of mathematical operations. Furthermore, it also includes a large number of other capabilities, like feature selection and imputation technical. In contrast to our algorithm, AutoFeat only does end-to-end pre-processing of the dataset, and does not train actual models during its execution. Our approach for automatic categorical feature transformations is slightly different; we do empirical tests and evaluate the different transformation combinations according to actual trained models and the metric that the user is interested in.

5 Conclusion

5.1 Findings and lessons learned

To summarize, two out of the four experiments that we conducted showed successful results, while the other 2 had no impact (positive or negative) in comparison to the performance of the baseline setup. These results were a bit surprising for us at first, because we learned about the importance that the categorical transformations might have on classification tasks. However, we understand that these transformations by themselves might not always improve the model performance directly, and in some cases we'll be able to see their impact only with the addition of other pre-processing steps. So, it probably was naive to believe that we'll see an improvement in all cases, as was indicated by the results of the experiments.

During the work on this project, we had an interesting and challenging experience. One of the main challenges that we had was to finalize our idea and come up with a feasible development and experiments plan (and approve it). A lot of effort has been already invested in the development of automated ML tools by the community, so in order to provide value in our project we need to try a lot of things. We learned that planning our experiments is key to our project's success. Some of the classification tasks that we wanted to include in our experiments were too ambitious, and when we tried them in practice they took too much time. We also learned the importance of creating a simple API and a clear user interface, both for our experiments and for the usability of our project for other data scientists as well.

5.2 Future steps

In the original proposal, we suggested several advanced ideas that were not included in the final project, because it was very challenging on its own. We regard to these ideas as possible future stets, which are listed below:

- Because the space of combinations of categorical feature transformations is with exponential relation to the number of candidate features, a simple extensive search might not be feasible in a lot of cases. For that reason, a smarter approach for hyper parameters search can be utilized, using existing frameworks, for example Optuna.
- Because of the computational challenge that was described in the previous bullet, we were unable to include all the different categorical transformation methods in the same combinations search. This isn't ideal, because in the same dataset, different features will probably have different methods that will better suit them. If implementing the idea from the previous bullet will be successful, it'll probably allow extending our algorithm to support these kinds of transformations combinations as well.
- For when the user chooses certain transformations combinations, we suggest providing a useful API that will generate a code block that will apply the selected transformations automatically, so the user will be able to copy-paste it.

References

[1] S. Mazzanti, "Beyond one-hot. 17 ways of transforming categorical features into numeric features." https://towardsdatascience.com/beyond-one-hot-17-ways-of-transforming-categorical-features-into-numeric-features-57f54f199ea4, 2020.

[2] I. Ul Haq, I. Gondal, P. Vamplew, and S. Brown, "Categorical features transformation with compact one-hot encoder for fraud detection in distributed environment," in *Data Mining: 16th Australasian Conference, AusDM 2018, Bahrurst, NSW, Australia, November 28–30, 2018, Revised Selected Papers 16.* Springer, 2019, pp. 69–80.

[3] F. Horn, R. Pack, and M. Rieger, "The autofeat python library for automated feature engineering and selection," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases.* Springer, 2019, pp. 111–120.