Student Research Project
Classification of Coupon Recommendations

(Computer Science Bachelor)

Maximilian Brückner

Summer semester 2021

Supervision by
Prof. Dr. rer. nat. Patrick Baier

Contents

1 Introduction

This project is based on the in-vehicle coupon recommendation data set.[Wan20] It contains information about whether a driver is interested in accepting a coupon for a restaurant or bar while driving. The data was collected via a survey on Amazon Mechanical Turk which presented participants with differing scenarios before asking them to make their decision. This poses a binary classification problem. Through examining different features like time of day, weather, the driver's destination and familial situation we ultimately need to come to a decision which can be either "yes, accept the coupon" or "no".

2 Topic

Firstly, we will be reviewing the data set in an exploratory data analysis. We will be looking at what kind of features are provided highlighting both their correlation to the label and amongst each other. Furthermore, the features will be graded based on their significance for making accurate predictions. We will also touch on the topic of feature engineering and the limitations of the provided data. Finally, our goal is to train varying models for classification as well as evaluate their performance and compare them against each other. In order to achieve best results we will thereby rely on cross-validation when determining hyper parameters. The implementation can be found in this repository. [Br1]

3 Implementation

3.1 Data Review

The data set contains 25 feature columns and one more for the label. This makes it difficult to display it in it's entirety, but here is an excerpt. For more details please refer to the source.[Wan20]

	destination	passenger	weather	temperature	time	coupon	expiration	gender	age	maritalStatus	...	CoffeeHouse	CarryAway
0	No Urgent Place	Alone	Sunny	55	2PM	Restaurant(<20)	1d	Female	21	Unmarried partner	...	never	NaN
1	No Urgent Place	Friend(s)	Sunny	80	10AM	Coffee House	2h	Female	21	Unmarried partner	...	never	NaN
2	No Urgent Place	Friend(s)	Sunny	80	10AM	Carry out & Take away	2h	Female	21	Unmarried partner	...	never	NaN
3	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	2h	Female	21	Unmarried partner	...	never	NaN
4	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	1d	Female	21	Unmarried partner	...	never	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
12679	Home	Partner	Rainy	55	6PM	Carry out & Take away	1d	Male	26	Single	...	never	1~3
12680	Work	Alone	Rainy	55	7AM	Carry out & Take away	1d	Male	26	Single	...	never	1~3
12681	Work	Alone	Snowy	30	7AM	Coffee House	1d	Male	26	Single	...	never	1~3
12682	Work	Alone	Snowy	30	7AM	Bar	1d	Male	26	Single	...	never	1~3
12683	Work	Alone	Sunny	80	7AM	Restaurant(20-50)	2h	Male	26	Single	...	never	1~3

Surprisingly all features are categorical, the only exception being "temperature". It could be considered numerical but there are only three different values in all 12684 data points. Even the drivers age is categorical because all drivers below the age of 21 and those above the age of 50 were grouped together respectively. Furthermore, the information provided is already quite expressive and does not leave any room for interpretation. This poses a significant challenge for Feature Engineering because we cannot derive new meaningful features. If there was a column containing something akin to a list of items in a shopping cart, we could generate a new feature like the number of items or their total value.
As often is the case not all entries are complete. We should therefore count the amount of missing values per feature.

- car 12576
- Bar 107
- CoffeeHouse 217
- CarryAway 151
- RestaurantLessThan20 130
- Restaurant20To50 189

Since for most features the proportion of NAs is small, we can simply fill the gaps using the mode. However, we must not include the "car" column where 99% of all entries are missing. In fact, it would be reasonable to drop the column altogether.
The next step is to prepare the data for model training through one-hot-encoding. Some features are already binary labeled, but most of them still need to be broken up into separate columns. Initially I wrote my own loop to adjust the data set. But you can also use pandas.get_dummies() which functions quite similarly.

```
dfOHE = df
featuresToBeOHE = featuresToBeOHE.drop(labels=['has_children', 'toCoupon_GEQ5min', '
    toCoupon_GEQ15min', 'toCoupon_GEQ25min', 'direction_same', 'direction_opp', 'Y'])
```

```
print(featuresToBeOHE)

for feature in featuresToBeOHE:
    print('Current feature: ' + feature)
    valueArray = df[feature].value_counts(dropna=False).index
    print(dfOHE[feature].value_counts())
    for value in valueArray:
        dfOHE[str(feature) + '_IS_' + str(value)] = df[feature].map(lambda x: 1 if x==
    value else 0)
        print('  Current value: ' + str(value))
        print(dfOHE[str(feature) + '_IS_' + str(value)].value_counts())
    print()

dfOHE = dfOHE.drop(columns=featuresToBeOHE)
```

Finally we perform the train-test-split and scale the features to facilitate convergence for gradient descent.

3.2 Exploratory Data Analysis

Before getting into training any models, we should examine the data more closely. Most importantly we need to ensure our data set is not imbalanced. Luckily, the label distribution appears to be even.

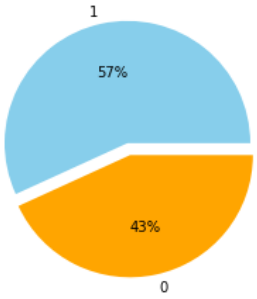
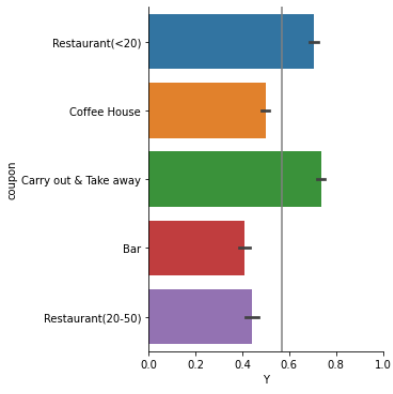


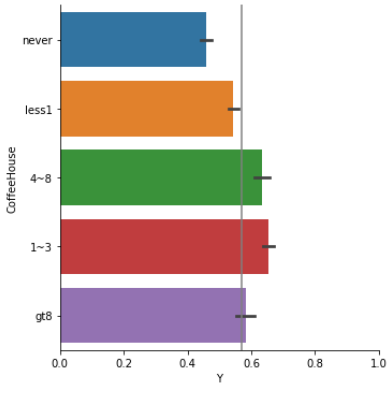
Figure 1: label distribution

3.2.1 Correlation

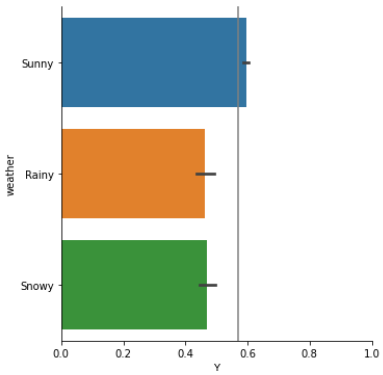
When examining correlations between features, I first attempted to generate a heatmap for all 115 columns. It is hard to read but can be viewed in notebook 02_LogReg_RF_GBT.ipynb.[ohe] There are patterns of boxlike of negative correlation along the diagonal which can be attributed to one-hot-encoding. Evidently, a positive entry for one manifestation demands negative entries for all other values of the previously combined categorical feature. Apart from that I observed a 1 to 1 correlation between people driving to work and the clock showing 7 am. Therefore we can drop one of the two because they effectively encode the same information. There are other unsurprising correlations like sunny weather indicating hot temperature or married drivers being more likely to have kids as passengers. But no major new insights were gained.
Instead we will examine the correlation between the label and selected group of features. Judging from the



(a) coupon type



(b) coffee house visit frequency



(c) weather

diagrams coupons for cheaper restaurants and "carry out & take away" seem to be most popular, possibly because it might have a proportionally greater impact on the total bill in these cases. We can also observe that drivers who will visit a coffee house between 1 and 8 times a month favor to accept. However, if they hardly ever visit such a place to begin with or even when they do so regularly, a coupon is less likely to sway them. And as expected people are less interested in taking a detour during bad weather.
Most notably, none of the features displays a major impact on the label. They are all aligned with the innate

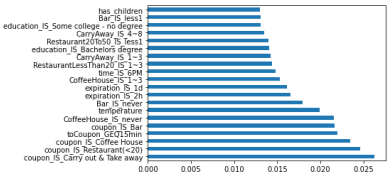
distribution (57% to 43%) indicated by the grey line. This trend can be observed throughout the entire data set with the ones shown above displaying the greatest variation already. Further plots can be found in the notebook 05_NN_with_vectors.ipynb.[NN] This hints at the difficulties we will encounter when training our models. If the data is too noisy, it might be impossible to make accurate predictions.

3.2.2 Feature Importance

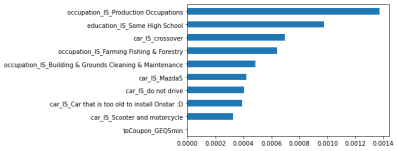
To grade the relevance of each feature we can employ the techniques illustrated in this article.[Sha18] Univariate selection relies on statistical tests similar to mean square error calculation. In this case the test was based on chi-squared. Feature importance can also be derived using a more practical approach. When training tree based classifiers, the model automatically scores features by their potency to create splits of low entropy. Here the ExtraTreesClassifier was used which is similar to Random Forest but with an even greater emphasis on reducing overfitting by randomly selecting by which feature to perform the next split.[Bha18] Top scorers

Feature	Score
temperature	211.388293
coupon_IS_Carry out & Take away	203.581136
coupon_IS_Restaurant(<20)	165.959110
coupon_IS_Bar	162.123273
CoffeeHouse_IS_never	120.945825
passenger_IS_Friend(s)	108.712940
CoffeeHouse_IS_1~3	94.383038
toCoupon_GEQ25min	90.365798
destination_IS_No Urgent Place	86.528807
expiration_IS_2h	85.798154

(a) Chi2 scores



(b) Top 20 Features



(c) Bottom 10 Features

are "coupon" type and "temperature". There is also a sizeable amount of extremely low scoring features which could potentially be dropped. This matches the previous observation of most features following the label distribution. Therefore, I created 3 separate copies of the data set including only the top 25, 50 and 75 features.

3.3 Model training

3.3.1 Basic models

The early phase of this project focussed on training three basic classification models, namely *Logistic Regression (LogR)*, *Random Forest(RF)* and *Gradient Boosting Tree (GBT)*. Their underlying mechanics will not be explained here as it is assumed you have familiarized yourself with them in your introductory course to machine learning. The majority of the training time was spent on optimizing hyper parameters. GridSearch is a potent tool facilitate experimentation. For *Logistic Regression* we need only determine the type of regularization to employ in order to avoid overfitting. As well as the "C" factor by which it should factor into the loss function.

```
parameter_candidates = [{'C': [0.1,0.2,0.4,0.6,1,2,5,7,10,15,30,100], 'penalty': ["l1",
    "l2"]}]
log = LogisticRegression(max_iter=1000,random_state=0)

clf_all = GridSearchCV(estimator=log, param_grid=parameter_candidates, n_jobs=-1)
clf_top25 = GridSearchCV(estimator=log, param_grid=parameter_candidates, n_jobs=-1)
clf_top50 = GridSearchCV(estimator=log, param_grid=parameter_candidates, n_jobs=-1)
clf_top75 = GridSearchCV(estimator=log, param_grid=parameter_candidates, n_jobs=-1)

clf_all = clf_all.fit(X_train_scaled, y_train)
clf_top25 = clf_top25.fit(X_train_top25_scaled, y_train)
clf_top50 = clf_top50.fit(X_train_top50_scaled, y_train)
clf_top75 = clf_top75.fit(X_train_top75_scaled, y_train)
print('--- Training on all Features ---')
print('Best penalty:', clf_all.best_estimator_.penalty)
print('Best C:', clf_all.best_estimator_.C)

print('--- Training on top 25 Features ---')
print('Best penalty:', clf_top25.best_estimator_.penalty)
print('Best C:', clf_top25.best_estimator_.C)
```

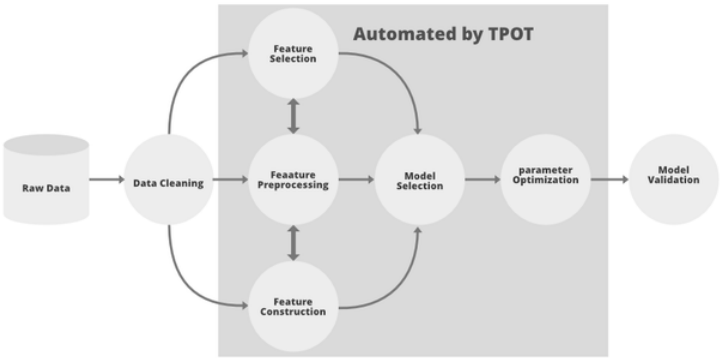
```
print('--- Training on top 50 Features ---')
print('Best penalty:', clf_top50.best_estimator_.penalty)
print('Best C:', clf_top50.best_estimator_.C)

print('--- Training on top 75 Features ---')
print('Best penalty:', clf_top75.best_estimator_.penalty)
print('Best C:', clf_top75.best_estimator_.C)
```

```
--- Training on all Features ---
Best penalty: l2
Best C: 7
--- Training on top 25 Features ---
Best penalty: l2
Best C: 0.1
--- Training on top 50 Features ---
Best penalty: l2
Best C: 0.2
--- Training on top 75 Features ---
Best penalty: l2
Best C: 0.4
```

Results show that with increasing amount of features regularization becomes more important. Given the opportunity our model would go to extreme lengths to memorize the training data by heart. Since the data set only contains about 12 thousand entries, this can happen if we make the model too powerful. But it would result in high variance and failure to detect broader patterns in the data. That is why we should penalize larger weights through regularization or focus on a core set of features.
For the two tree based models training time ramps up quickly as the amount of possible combinations increases. It proved to be impractical to provide long lists of parameter candidates. Instead, I decided to run multiple attempts with smaller lists. Initially, providing a wide range of candidates while narrowing out which intervals to focus on. The parameters in question were number of trees in the forest and their respective maximum depth. The optimal values ranged between 250 to 280 trees with a depth of 12 to 17. The models training on the full data set preferred higher values. Training the Gradient Boosting Tree is what took up most of the computation time. Unlike in a Random Forest the GBT cannot train multiple trees in parallel because they all build on their predecessor. For this reason I did not make the effort to train GBTs for the three streamlined versions of the data set containing less features. Yet, it is probably that the results would have matched the trend of the other two models.

3.3.2 AutoML

AutoML describes tools which are meant to facilitate feature engineering and model training both saving time in research and development as well as making machine learning more accessible in general. After training my own models this lent itself perfectly as comparison. Using the *TPOT (Tree-Based Pipeline Optimization)* python package you need only clean the data (fill NAs) and forward it in a suitable format (one-hot-encoding). The pipeline will automatically try to find the best classifier by trial and error. It employs genetic programming inspired by the idea of natural selection.[Sax21] It recommended using a combination of RandomForestClassifier for estimations on feature synthesis and ExtraTreesClassifier for prediction with a forest size of 100 trees each.



3.3.3 Neural net

In the final phase of the project I worked with the PyTorch and TensorFlow libraries to set up a *neural net* for classification. After working through the tutorials[PyT21] I went on to transform the one-hot-encoded data into tensors which can be passed to the 114 input neurons, one for each feature. Between layers a non-linear activation is performed using ReLu. Later on Batch Normalization was added, too. In regards to the net's layout I found little to go on. After trying many different variations of expansion and reduction in neuron count and layer depth, I came to the conclusion that performance is best on a simple layout with at most 2 or 3 hidden layers. In order to further simplify the neural net, I used label encoding and categorical embeddings[S21] to pass up on one-hot-encoding reduce the required amount of input neurons back to 25. By adding a dropout factor some elements of the input tensor will be randomly set to 0 during training. This is has a similar effect to the regularization in *Logistic Regression*.
The biggest hindrance was probably the lack of data. Depending on the dropout models would either converge quickly, memorizing the entire training data and producing significant overfit, or they would oscillate and make no progress to convergence regardless of training time.

```
class Model(nn.Module):

    def __init__(self, embedding_size, num_numerical_cols, output_size, layers, p=0.1):
        super().__init__()
        self.all_embeddings = nn.ModuleList([nn.Embedding(ni, nf) for ni, nf in
    embedding_size])
        self.embedding_dropout = nn.Dropout(p)
        self.batch_norm_num = nn.BatchNorm1d(num_numerical_cols)

        all_layers = []
        num_categorical_cols = sum((nf for ni, nf in embedding_size))
        input_size = num_categorical_cols + num_numerical_cols

        for i in layers:
            all_layers.append(nn.Linear(input_size, i))
            all_layers.append(nn.ReLU(inplace=True))
            all_layers.append(nn.BatchNorm1d(i))
            all_layers.append(nn.Dropout(p))
            input_size = i

        all_layers.append(nn.Linear(layers[-1], output_size))

        self.layers = nn.Sequential(*all_layers)

    def forward(self, x_categorical, x_numerical):
        embeddings = []
        for i,e in enumerate(self.all_embeddings):
            embeddings.append(e(x_categorical[:,i]))
        x = torch.cat(embeddings, 1)
        x = self.embedding_dropout(x)

        x_numerical = self.batch_norm_num(x_numerical)
        x = torch.cat([x, x_numerical], 1)
        x = self.layers(x)
        return x
```

4 Evaluation

5 Summary

References

[Bha18] Naman Bhandari. https://medium.com/@namanbhandari/extratreesclassifier-8e7fc0502c7, 2018.

[Br1] Maximilian Brückner. https://github.com/Lumi1070/ML-Projektarbeit/, 2021.

[NN] https://github.com/Lumi1070/ML-Projektarbeit/blob/master/$05_N N_w ith_v ectors.ipynb$.

[ohe] https://github.com/Lumi1070/ML-Projektarbeit/blob/master/$02_L ogReg_R F_G BT.ipynb$.

[PyT21] PyTorch. https://pytorch.org/tutorials/, 2021.

[S21] Aakanksha N S. https://towardsdatascience.com/deep-learning-for-tabular-data-using-pytorch-1807f2858320, 2021.

[Sax21] Pawan Saxena. https://www.geeksforgeeks.org/tpot-automl/, 2021.

[Sha18] Rahil Shaikh. https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e, 2018.

[Wan20] Tong Wang. https://archive.ics.uci.edu/ml/datasets/in-vehicle+coupon+recommendation, 2020.