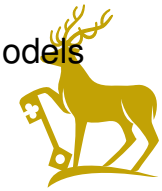
MATM063: Principles of Data Science, Python LAB

Worksheet 4 (Week 4)

MATM063: Preparing data for ML & Regression models

Werner Bauer

Department of Mathematics



UNIVERSITY OF SURREY

Key Learning: Cleaning and scaling data, converting categories, preprocessing pipeline, regression models

1 USUAL WORK FLOW IN DATA SCIENCE (PART 2)

Here, in short, part 2 of the steps to be done to prepare your data for machine learning algorithms.

1. Split training data into predictors and labels
2. Clean data: remove missing values or fill in values (e.g. SimpleImputer)
3. Convert categorical to numerical values (e.g. OneHotEncoder)
4. Scale features (e.g. MinMaxScaler, StandardScaler)
5. Combine these steps into a transformer pipeline
6. Train and evaluate your ML model

2 REGRESSION MODELS

For the provided dataset, we want to use a regression model. Besides those regression models discussed in the lectures, another one is included that you will need further below in Q2.

- Linear Regression:

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression() # create instance
lin_reg.fit(housing_prepared, house_labels) # train the model
```

- Decision Tree Regressor:

```
from sklearn.tree import DecisionTreeRegressor
tree_reg = DecisionTreeRegressor()
tree_reg.fit(housing_prepared, housing_labels)
```

- In Q2, you should train a Random Forest Regressor:

```
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor()
forest_reg.fit(housing_prepared, housing_labels)
```

3 EVALUATION OF THE MODELS

Here are two ways of evaluating the performance of regression model:

- Root Mean Square Error (RMSE):

```
from sklearn.metrics import mean_squared_error
housing_predictions = lin_reg.predict(housing_prepared)
lin_rmse = np.sqrt(mean_squared_error(housing_labels,housing_predictions))
print(lin_rmse)
```

- k-fold Cross Validation (CV):

```
from sklearn.model_selection import cross_val_score

lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,
```

```
                            scoring="neg_mean_squared_error", cv=10)
lin_rmse_scores = np.sqrt(-lin_scores)

# Use output function:
def display_scores(scores):
    print ("Scores:", scores)
    print ("Mean:", scores.mean())
    print ("Standard deviation:", scores.std())
display_scores(lin_rmse_scores)
```

4 CUSTOMER TRANSFORMER TO ADD COMBINED ATTRIBUTES

The following customer transformer is used below in the transformer pipeline.

```
rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6
class CombinedAttributesAdder (BaseEstimator, TransformerMixin):
    def __init__ (self, add_bedrooms_per_room=True):
        self.add_bedrooms_per_room =
        add_bedrooms_per_room
    def fit(self, X, y=None):
        return self # nothing else to do
    def transform(self, X):
        rooms_per_household = X[:, rooms_ix] /
                              X[:, households_ix]
        population_per_household = X[:,population_ix]/
                                   X[:,households_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] /
                                X[:, rooms_ix]
            return np.c_[X, rooms_per_household,
                         population_per_household,
                         bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household,
                         population_per_household]
```

Listing 1. combined attribute adder

5 DATA PREPROCESSING CLASSES AND PIPELINES

Here is a summary of the packages we can use to do data preprocessing.

- Data cleaning:

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='median') # create instance and define strategy
X = imputer.fit_transform(housing_num)#numpy array
housing_tr = pd.DataFrame(X, columns=housing_num.columns, index=housing_num.index) #DataFrame
```

- Converting text (categories) to numbers:

```
# define DataFrame with categories
housing_cat = housing[['ocean_proximity']]

from sklearn.preprocessing import OrdinalEncoder
ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)

# Output categories with:
cat_encoder.categories_
# Output:
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'], dtype=object)]

# simiarly for OneHotEncoder:
from sklearn.preprocessing import OneHotEncoder
cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
```

- Feature scaling:

```
# normalising data:
from sklearn.preprocessing import MinMaxScaler
minmax_scale = MinMaxScaler()
housing_minmax =  minmax_scale.fit_transform(housing_num)

# standardising data:
from sklearn.preprocessing import StandardScaler
std_scale = StandardScaler()
housing_std = std_scale.fit_transform(housing_num)
```

- Setting up a transformation pipeline:

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

# define numerical and categorical attribute lists
num_attributs = list(housing_num)
cat_attributs = ['ocean_proximity']

num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('attribs_adder',CombinedAttributesAdder(add_bedrooms_per_room=True)),
    ('std_scaler', StandardScaler()), ])

full_pipeline = ColumnTransformer([('num', num_pipeline, num_attributs),
    ('cat', OneHotEncoder(), cat_attributs), ])
```

- Automatisation: easily transform data by applying `full_pipeline` directly to original data set `housing`.

```
housing_prepared = full_pipeline.fit_transform(housing)
```

6 DEFINING CELLS IN SPYDER

Sometimes it is useful not to execute the entire Python script. This could be the case when the training of a model takes longer (e.g. you will see that the Random Forest Regressor takes some time to be trained). Then, you can split your code into cells that can be executed separately (e.g. with CTRL-Return shortcut you can execute a selected cell). To begin a new cell in your code, simply use the following command:

```
#%% HERE YOU CAN ADD SOME COMMENT

import ...
...
```

Listing 2. organizing code parts into cells in Spyder

7 EXERCISES

Question 1:

Here, you should combine the material we discussed in the lectures into **one running Python script** that reads in the housing file, creates a test and training set, transforms the data such that it can be used in a machine learning algorithm, and then trains a linear regression model.

Please proceed as follows to create such script:

1. Start you script with the part that reads in the housing data file from the internet `https://raw.githubusercontent.com/ageron/handson-ml2/master/datasets/housing/housing.tgz`.
2. Load the housing.csv file using the `fetch_housing_data()` function.
3. Split your data into test and training data by using Stratified Shuffle Split (20% test data) while assuring that `median_income` is proportionally represented in the test data.
4. Separate the target `median_house_value` from the predictors (similarly to `housing` and `housing_labels` as done in the lectures). Then, define numerical attributes and categorical attributes (in the lectures we called them `num_attributs` and `cat_attributs`).
5. Write a full pipeline that fits and transforms `housing` to `housing_prepared` that can be used to train ML models. Make sure that you can change the option `add_bedrooms_per_room` from True to False as we will need this at the next step.
6. Define and train a linear regression model (as done in the lectures). Output both, the model's RMSE for the training dataset and also the cross validation scores (10-folds). Compare two realization of these models, a) one where you set `add_bedrooms_per_room = True` (you should have very similar results as in the lecture notes); and b) one with `add_bedrooms_per_room = False`

7. Test also the performance of these two models (for the default customer transformer setting) when comparing min-max scaling vs standardisation.
8. Finally test (for the default customer transformer setting and standardisation) what happens when you use the Ordinal Encoder rather then the OneHotEncoder.

Question 2:

This question requires you that you have successfully finished Q1.

1. Train a Random Forest Regressor on the prepared housing dataset (Steps 1-5 of Q1).
2. Make a 'no-CV' predition (no cross-validation) with this model for the entire training set and output the RMSE.
3. Now, make a prediction using a 5-fold cross-validation and output: Scores, Mean, Standard deviation.
4. Compare the RMSE of the no-CV with the CV predictions, and deduce from these values if your model has underfit or overfit the training data?

Question 3:

This question is related to the code you did produce for Question 1 and Question 2.

1. Update the customer transformer (Listing 1) such that it has two hyper-parameters: `add_bedrooms_per_room = True` as well as `add_rooms_per_household = True`.
2. Update the pipeline from Q1 such that it uses this new transformer.
3. Try out different combinations of adding or omitting these combined attributes to explore if their presents might improve the performance of the linear regression model.

Question 4:

For this question, you have to write a Python script that makes predictions for the target value `median_income` (Note: in Q1 the target value is `median_house_values`). The values in `median_income` are in units of 10.000 $. (Hint: rather than writing the whole script from scratch, you could also copy the one from Q1 and change the corresponding entries).

1. Load the housing file.
2. Split your data into test and training data using Stratified Shuffle Split (15% test data) while assuring that `median_house_values` is proportionally represented in the test data.
3. Use the training set to proceed and separate `median_income` from the predictors.
4. Write a transformer pipeline (as in Q1) that summarizes all the necessary data preprocessing steps into one function (use default settings)!
5. Train a Linear Regression, a Decision Tree Regressor, and a Random Forest Regressor, evaluate their RMSEs on the training set and when applying cross-validation. Are the models capable of making reasonable predictions of `median_income`?