greatlearning

Feature Engineering

Learning Objectives

- Introduction to Feature engineering
- Hands on exercise on Feature engineering
- Part 2 of feature engineering and model tuning
- How to tune the models or improve performance
- Concept of upsampling and downsampling
- Hands on exercise showing tuning of a model

Introduction to Feature engineering

- **Feature engineering** is the process of using domain knowledge of the data to create features that make machine learning algorithms work.

- It is fundamental to the application of machine learning, and is both difficult and expensive.

- One reason why feature engineering is so important is that defining and/or learning higher level domain specific feature is actually one way to deep learning.

Hands on exercise on Feature engineering

Some important functions

1. Importing packages

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

From sklearn import matrics

from sklearn.preprocessing import Imputer

from sklearn.tree import DecisionTreeClassifier

2. Split dataset into inputs and outputs

```
values = pima_df.values
X = values [:,0:8]
Y = values [:,8]
test_size = 0.30 # taking 70:30 training and test set
```

3. Fill missing values with mean column values

```
imputer = Imputer(missing_values = ‘NaN’, strategy=‘median’, axis=0)
transformed_x= imputer.fit_transform(X)
```

Part 2 of feature engineering and model tuning

- Feature engineering is a set of those activities which performs a discipline we follow, to ensure that you finally feed the right data to your algorithms, in which the target variable is very strongly influenced by the independent variables.

- The independent variables do not have any significant outliers, all the missing values have been taken care of.

- As a part of data preparation, we have 2 techniques such as Upsampling and Downsampling

Hands on exercise feature engineering part 2

greatlearning

Analysing car mpg data set using clustering and PCA

Some important functions

1. Importing libraries

```
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import pandas as pd
import matplotlib.pyplot as plt
```

2. Drop unique columns

```
Numeric_cols = mpg_df.drop('car_name', axis=1)
```

3. Pair plot analysis to visually check number of likely clusters

mpg_df_attr = mg_df.iloc[; ,0:9]

sns.pairplot(mpg_df_attr, diag_kind = 'kde') #to plot density curve instead of histogram

4. Splitting the data

array = mpg_df_attr_z.values

X= array [: , 1:5]

Y= array [: , 0]

X_train, x_test, y_train, y_test = train_test_split (x, y, test_size=0.30, random_state=1)

Iteration 2

5. Drop acc column and follow the same steps for splitting

mpg_df_attr_z.pop ('acc')

array = mpg_df_attr_z.values

And same splitting steps

6. Create a regularized RIDGE model and note the coefficients

ridge= Ridge(alpha = 0.3)

ridge.fit(x_train, y_train)

print("Ridge model:" , (ridge.coef_))

Ridge model: [[2.47057456 2.4449775 ...]]

7. Create a regularized LASSO model and note the coefficients

```
lasso= Lasso(alpha = 0.2)
lasso.fit(x_train, y_train)
print("Lasso model:" , (lasso.coef_))
Lasso model: [[0.  0.  -0.3456   -4.09876 ---]]
```

8. Generate polynomial models reflecting the non-linear interaction between some dimensions

```
from sklearn.precprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree = 2, interaction_only = True)
X_poly = poly.fit_transform(X_scaled)
X_train, x_test, y_train, y_test = train_test_split (x_poly, y, test_size=0.30, random_state = 1)
X_train.shape
```

How to tune the models or improve performance

- To increase performance of any model, increase model capacity.

- The tuning process is more empirical than theoretical. We add layers and nodes gradually with the intention to overfit the model since we can tone it down with regularizations.

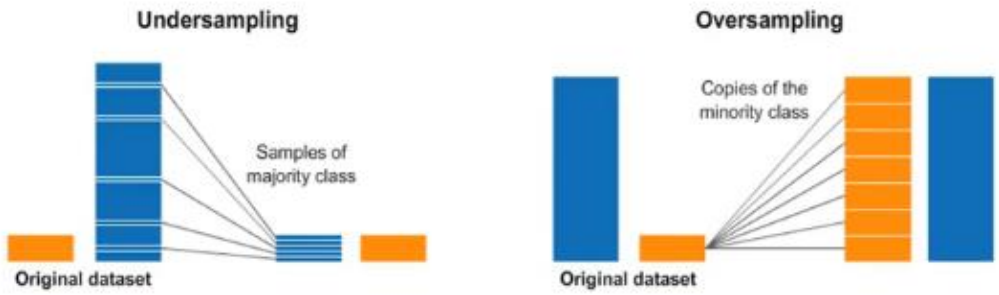
- We repeat the iterations until the accuracy improvement is diminishing and no longer justify the drop in the training and computation performance

Concept of upsampling and downsampling

greatlearning

- We can handle the imbalanced dataset cases to minimize the Type II errors by balancing the class representations.

- To balance the classes we can –
  - Decrease the frequency of the majority class
  - Increase the frequency of the minority class



Imblearn techniques

- Python imbalanced-learn module – provides more sophisticated resampling techniques.
- In over-sampling, instead of creating exact copies of the minority class records, we can introduce small variations into those copies creating more diverse synthetic samples.
- SMOTE consists of synthesizing elements for the minority class, based on those that already exist, works randomly picking a point from the majority class and computing the K-nearest neighbours.

Hands on exercise showing tuning of a model

1. Importing libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.style
plt.style.use('classic')
import seaborn as sns
```

2. Copy all the predictor variables into X dataframe.

```
X= mpg_df.drop('mpg', axis=1)
X= x.drop({'origin_america', 'origin_asia', 'origin_europe'}, axis=1)
```

3. Prepare the model after splitting

regression_model = LinearRegression()

regression_model.fit(x_train, y_train)

4. Explore the coefficients for each of the independent attributes

for idx, col_name in enumerate(x_train.columns):

print("The coefficient for {} is {}". Format(col_name, regression_model.coef_[col_names]))

5. Down sampling the larger class i.e. non diabetics

non_diab_indices = pima_df[pima_df['class']== 0].index

no_diab = len(pima_df['class'] == 0])

print(no_diab)

pima_df_down_sample = pima_df.loc [down_sample_indices] #extract all those recommendations

pima_df_down_sample.shape

pima_df_down_sample.groupby([“class”]).count # Look at the class distribution

6. Fit the model on 30% through Logistic Regression

model = LogisticRegression()

model.fit(x_train, y_train)

y_predict = model.predict(x_test)

model_score = model.score(x_test, y_test)

print(model_score)

print(metrics.confusion_matrix(y_test, y_predict))

greatlearning

HappyLearning

Questions?