Optimization Techniques for Machine Learning

Alassane KONE
(ISE 2019 – Senior Data Scientist)

February 2023

Chapter 4

Data Preparation Optimization

1. Grid Search Technique for Data Preparation

Overview

Data preparation can be challenging.

The approach that is most often prescribed and followed is to analyze the dataset, **review the requirements of the algorithms, and transform the raw data to best meet the expectations of the algorithms.**

This can be effective but is also slow and can require deep expertise with data analysis and machine learning algorithms.

An alternative approach is to treat the preparation of input variables as a hyperparameter of the modeling pipeline and to tune it along with the choice of algorithm and algorithm configurations.

This can be achieved by designing a grid search of data preparation techniques and/or sequences of data preparation techniques in pipelines. This may involve evaluating each on a single chosen machine learning algorithm, or on a suite of machine learning algorithms.

The benefit of this approach is that it always results in suggestions of modeling pipelines that give good relative results. Most importantly, it can unearth the non-obvious and unintuitive solutions to practitioners without the need for deep expertise.

Use Case : Data Prep. Optimization in a Classification Problem

Initialization

```
In this section, we will first select a standard machine learning dataset and establish a baseline in performance on this dataset. This will provide the context for exploring the grid search method of data preparation in the next section.

We will use a classification dataset.

This dataset has 13 input variables that describe the chemical composition of samples of a product and requires that the product be classified as one of three types.

# 1. Import pandas as pd

# 2.  Read the dataset in a dataframe called df - set the header parameters to None

# 3. Retrieve the values of df in a variable called data

# 4. Split the columns into X and y variables (Use sclicing here and note that the last variable is the target (y))

# 5. Summarize the shape of X and y
```

Use Case : Data Prep. Optimization in a Classification Problem

Baseline Model Performance

```
# 6. Set X type to float and ensure that y is well Label Encoded (use the LabelEncoder from sklearn.preprocessing)

# 7. Import cross_val_score and RepeatedStratifiedKFold from sklearn.model_selection

# 8. Create a function called evaluate_model, the function should take as input X, y and a variable called model. Create in the function a variable called cv which is an instance of RepeatedStratifiedKFold with n_splits=10, n_repeats=3, random_state=1 also create a variable called scores which is an instance of cross_val_score with theses inputs : model, X, y, scoring="accuracy", cv=cv, n_jobs=-1). The function should return scores.

# 9. create a variable model which is an instance of LogisticRegression. Set the solver parameter to liblinear.

# 10. Evaluate the previous model using your function evaluate_model. Put the output in variable called scores.

# 11. Write the code below and explain what it does:
print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

Use Case : Data Prep. Optimization in a Classification Problem

Baseline Model Performance

```
# 6. Set X type to float and ensure that y is well Label Encoded (use the LabelEncoder from sklearn.preprocessing)

# 7. Import cross_val_score and RepeatedStratifiedKFold from sklearn.model_selection

# 8. Create a function called evaluate_model, the function should take as input X, y and a variable called model. Create in the function a variable called cv which is an instance of RepeatedStratifiedKFold with n_splits=10, n_repeats=3, random_state=1 also create a variable called scores which is an instance of cross_val_score with theses inputs : model, X, y, scoring="accuracy", cv=cv, n_jobs=-1). The function should return scores.

# 9. create a variable model which is an instance of LogisticRegression. Set the solver parameter to liblinear.

# 10. Evaluate the previous model using your function evaluate_model. Put the output in variable called scores.

# 11. Write the code below and explain what it does:
print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

Use Case : Data Prep. Optimization in a Classification Problem

Grid Search Approach to Data Transformation

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import QuantileTransformer
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.decomposition import PCA
from sklearn.decomposition import TruncatedSVD
from matplotlib import pyplot
```

Use Case : Data Prep. Optimization in a Classification Problem

Grid Search Approach to Data Transformation

```
def get_pipelines(model):
  """ ADD YOUR OWN DOCSTRING HERE BASED ON YOUR UNDERSTANDING OF THE FUNCTION """
 pipelines = list()
 # normalize
 p = Pipeline([('s',MinMaxScaler()), ('m',model)])
 pipelines.append(('norm', p))
 # standardize
 p = Pipeline([('s',StandardScaler()), ('m',model)])
 pipelines.append(('std', p))
 # quantile
 p = Pipeline([('s',QuantileTransformer(n_quantiles=100, output_distribution='normal')),
(pipelines.append(('quan', p))
 # discretize
 p = Pipeline([('s',KBinsDiscretizer(n_bins=10, encode='ordinal', strategy='uniform')), ('m',model)])
 pipelines.append(('kbins', p))
 # pca
 p = Pipeline([('s',PCA(n_components=7)), ('m',model)])
 pipelines.append(('pca', p))
 # svd
 p = Pipeline([('s',TruncatedSVD(n_components=7)), ('m',model)])
 pipelines.append(('svd', p))

 return pipelines
```

Use Case : Data Prep. Optimization in a Classification Problem

Grid Search Approach to Data Transformation

```
# 12. Create again the same variable model as in question 9

# 13. Apply the function get_pipelines to model and put the result in a variable called pipelines

# 14. Now we are going to evaluate each pipeline in pipelines. For that write and compile the code
below:
results, names = list(), list()
for name, pipeline in pipelines:
 # evaluate
 scores = evaluate_model(X, y, pipeline)
 # summarize
 print('>%s: %.3f (%.3f)' % (name, mean(scores), std(scores)))
 # store
 results.append(scores)
 names.append(name)
# plot the result
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()

# 15. What can you conclude ?
```

Use Case : Data Prep. Optimization in a Classification Problem

Grid Search Approach to Missing Data Handling

```
# 1. Load the dataset data_0003.csv with header=None and na_values='?'

# 2. Print the first few rows

# 3. Enumerate each column and report the number of rows with missing values for the column.

# 4. Comment your result

# 5. Split the dataframe into input and output variables (READ CARREFULLY THE DESCRIPTION)

# 6. Print total missing values in X

# 7. from sklearn.impute import SimpleImputer

# 8. Create the variable imputer, an instance of SimpleImputer with strategy='mean'.

# 9. Fit the imputer on X

# 10. Create a variable X_trans, a transformation of X using the imputer

# 11. Print total missing values in X_trans.
```

Use Case : Data Prep. Optimization in a Classification Problem

Missing Data Handling: Baseline

```
# 1. Load the dataset data_0003.csv with header=None and na_values='?'

# 2. Print the first few rows

# 3. Enumerate each column and report the number of rows with missing values for the column.

# 4. Comment your result

# 5. Split the dataframe into input and output variables (READ CARREFULLY THE DESCRIPTION)

# 6. Print total missing values in X

# 7. from sklearn.impute import SimpleImputer

# 8. Create the variable imputer, an instance of SimpleImputer with strategy='mean'.

# 9. Fit the imputer on X

# 10. Create a variable X_trans, a transformation of X using the imputer

# 11. Print total missing values in X_trans.
```

Use Case : Data Prep. Optimization in a Classification Problem

Missing Data Handling: Baseline Evaluation

```
from numpy import mean
from numpy import std
from pandas import read_csv
from sklearn.ensemble import RandomForestClassifier
from sklearn.impute import SimpleImputer
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.pipeline import Pipeline

model = RandomForestClassifier()
imputer = SimpleImputer(strategy='mean')
pipeline = Pipeline(steps=[('i', imputer), ('m', model)])
# define model evaluation
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(pipeline, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
print('Mean Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

Use Case : Data Prep. Optimization in a Classification Problem

Missing Data Handling: Grid Search

```
# 12. Create this list strategies = ['mean', 'median', 'most_frequent', 'constant']

# 13. For each strategie in strategies, apply the previous model implementation

# 14. Print the result and the boxplot

# 15. Conclusion ?

# 16. Using the best pipeline, predict the output for this :

row = [2, 1, 530101, 38.50, 66, 28, 3, 3, nan, 2, 5, 4, 4, nan, nan, nan, 3, 5, 45.00, 8.40, nan, nan, 2,
11300, 00000, 00000, 2]
```