index

April 20, 2022

1 Integrating PCA in Pipelines - Lab

1.1 Introduction

In a previous section, you learned about how to use pipelines in scikit-learn to combine several supervised learning algorithms in a manageable pipeline. In this lesson, you will integrate PCA along with classifiers in the pipeline.

1.2 Objectives

In this lab you will:

- Integrate PCA in scikit-learn pipelines

1.3 The Data Science Workflow

You will be following the data science workflow:

1. Initial data inspection, exploratory data analysis, and cleaning
2. Feature engineering and selection
3. Create a baseline model
4. Create a machine learning pipeline and compare results with the baseline model
5. Interpret the model and draw conclusions

1.4 Initial data inspection, exploratory data analysis, and cleaning

You'll use a dataset created by the Otto group, which was also used in a Kaggle competition. The description of the dataset is as follows:

The Otto Group is one of the world's biggest e-commerce companies, with subsidiaries in more than 20 countries, including Crate & Barrel (USA), Otto.de (Germany) and 3 Suisses (France). They are selling millions of products worldwide every day, with several thousand products being added to their product line.

A consistent analysis of the performance of their products is crucial. However, due to their global infrastructure, many identical products get classified differently. Therefore, the quality of product analysis depends heavily on the ability to accurately cluster similar products. The better the classification, the more insights the Otto Group can generate about their product range.

In this lab, you'll use a dataset containing: - A column `id`, which is an anonymous id unique to a product - 93 columns `feat_1`, `feat_2`, …, `feat_93`, which are the various features of a product - a column `target` - the class of a product

The dataset is stored in the `'otto_group.csv'` file. Import this file into a DataFrame called `data`, and then:

- Check for missing values
- Check the distribution of columns
- … and any other things that come to your mind to explore the data

[1]:
```
# Your code here
import pandas as pd
df = pd.read_csv('otto_group.csv')
```

[2]:
```
# Your code here
sum(df.isna().sum())
```

[2]: 0

[3]:
```
# Your code here
hist = df.loc[:,"feat_1":"feat_30"]
hist.hist(figsize = (30,20));
```

[4]:
```
# Your code here
```

[5]:
```
# Your code here
```

[6]:
```
# Your code here
```

If you look at all the histograms, you can tell that a lot of the data are zero-inflated, so most of the variables contain mostly zeros and then some higher values here and there. No normality, but for most machine learning techniques this is not an issue.

[7]:
```
# Your code here
```

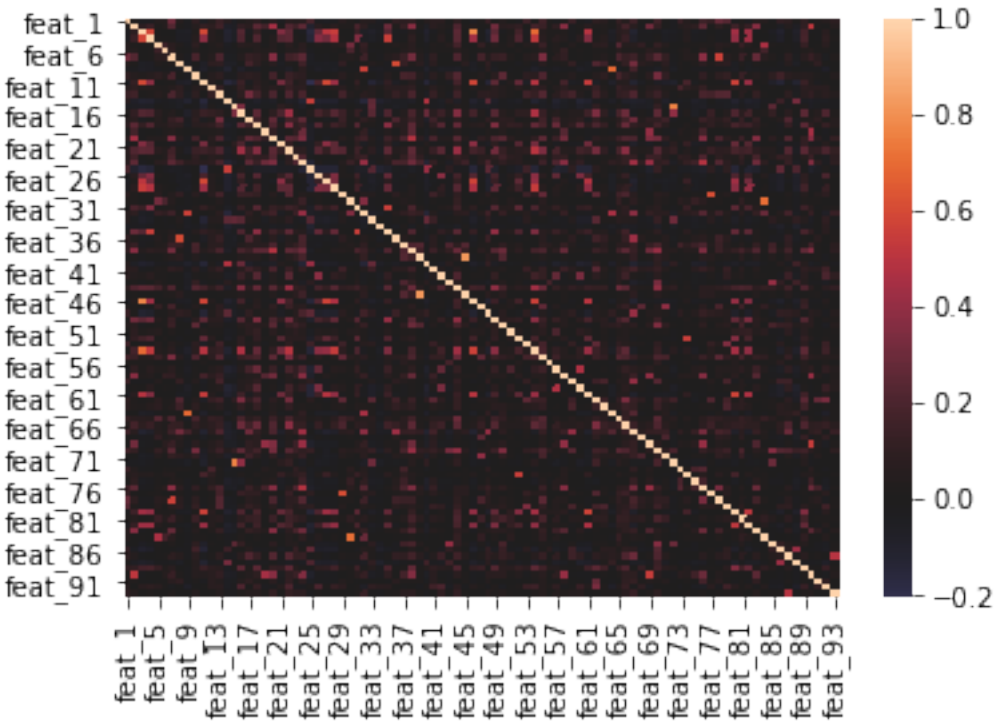
Because there are so many zeroes, most values above zero will seem to be outliers. The safe decision for this data is to not delete any outliers and see what happens. With many 0s, sparse data is available and high values may be super informative. Moreover, without having any intuitive meaning for each of the features, we don't know if a value of ~260 is actually an outlier.

[8]:
```
# Your code here
```

1.5 Feature engineering and selection with PCA

Have a look at the correlation structure of your features using a heatmap.

[9]:
```
# Your code here
import seaborn as sns
heat = df.loc[:,"feat_1":"feat_93"]
sns.heatmap(heat.corr(), center = 0);
```



Use PCA to select a number of features in a way that you still keep 80% of your explained variance.

[10]:
```
# Your code here
from sklearn.decomposition import PCA

pca_20 = PCA(n_components = 20)
pca_30 = PCA(n_components = 30)
pca_40 = PCA(n_components = 40)

y = df["target"]
x = df.drop(columns = ["target", "id"])

p_20 = pca_20.fit_transform(x)
p_30 = pca_30.fit_transform(x)
p_40 = pca_40.fit_transform(x)

print(np.sum(pca_20.explained_variance_ratio_))
print(np.sum(pca_30.explained_variance_ratio_))
print(np.sum(pca_40.explained_variance_ratio_))
```

```
0.7275677925975583
0.8241297169985204
0.8886540441402313
```

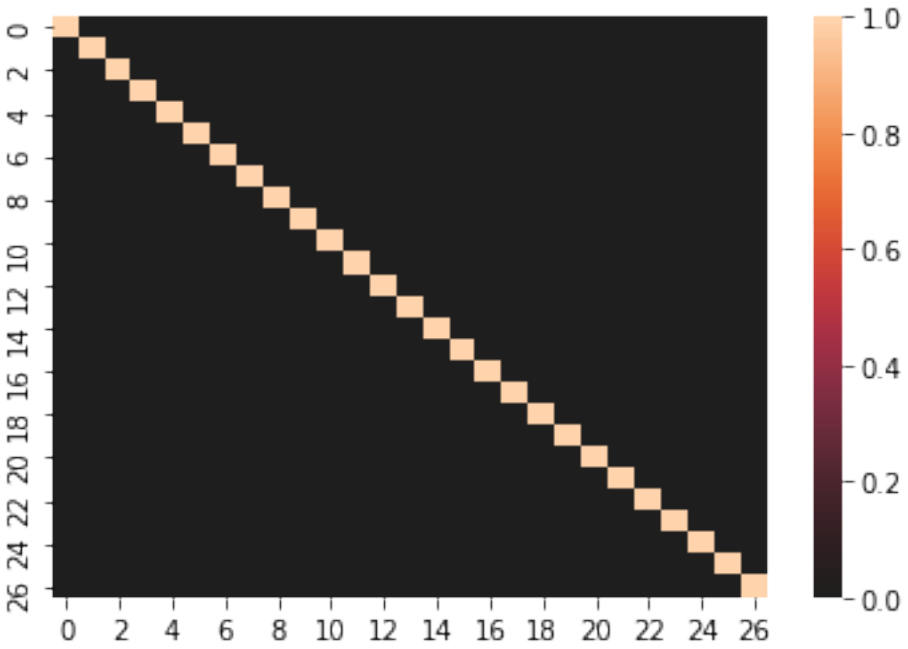
[11]:
```
# Your code here
pca_27 = PCA(n_components = 27)
p_27 = pca_27.fit_transform(x)
print(np.sum(pca_27.explained_variance_ratio_))
```

```
0.8004485447413247
```

[12]:
```
len(list(pca_27.feature_names_in_))
```

[12]: 93

[13]:
```
pca_df = pd.DataFrame(p_27)
sns.heatmap(pca_df.corr(), center = 0);
```



1.6 Create a train-test split with a test size of 40%

This is a relatively big training set, so you can assign 40% to the test set. Set the `random_state` to 42.

[14]:
```
# Your code here
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.4,
                                                    random_state = 42)
```

[15]:
```
# Your code here
```

1.7 Create a baseline model

Create your baseline model *in a pipeline setting*. In the pipeline:

- Your first step will be to scale your features down to the number of features that ensure you keep just 80% of your explained variance (which we saw before)
- Your second step will be to build a basic logistic regression model

Make sure to fit the model using the training set and test the result by obtaining the accuracy using the test set. Set the `random_state` to 123.

[17]:
```
# Your code here
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
```

```
pipe_lr = Pipeline([("pca", PCA(n_components = 27, random_state = 123)),
                    ("lr", LogisticRegression(random_state = 123))])

pipe_lr.fit(X_train,y_train)
print(pipe_lr.score(X_test,y_test))
```

```
0.7279411764705882
```

```
/opt/anaconda3/envs/learn-env/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[18]:
```
# Your code here
from sklearn.metrics import accuracy_score

print(accuracy_score(y_test, pipe_lr.predict(X_test)))
```

```
0.7279411764705882
```

[19]:
```
# Your code here
```

1.8 Create a pipeline consisting of a linear SVM, a simple decision tree, and a simple random forest classifier

Repeat the above, but now create three different pipelines: - One for a standard linear SVM - One for a default decision tree - One for a random forest classifier

[20]:
```
# Your code here
#   This cell may take several minutes to run

from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline

pipe_tree = Pipeline([("pca", PCA(n_components = 27, random_state = 123)),
                    ("tree", DecisionTreeClassifier(random_state = 123))])

pipe_tree.fit(X_train,y_train)
print(pipe_tree.score(X_test,y_test))
print(accuracy_score(y_test, pipe_tree.predict(X_test)))
```

```
0.671178086619263
0.671178086619263
```

[21]:
```
# Your code here
#   This cell may take several minutes to run
from sklearn.ensemble import RandomForestClassifier

pipe_rf = Pipeline([("pca", PCA(n_components = 27, random_state = 123)),
                ("rf", DecisionTreeClassifier(random_state = 123))])

pipe_rf.fit(X_train,y_train)
print(pipe_rf.score(X_test,y_test))
print(accuracy_score(y_test, pipe_rf.predict(X_test)))
```

```
0.671178086619263
0.671178086619263
```

[22]:
```
# Your code here
#   This cell may take several minutes to run
from xgboost import XGBClassifier

pipe_xgb = Pipeline([("pca", PCA(n_components = 27, random_state = 123)),
                ("xgb", XGBClassifier(random_state = 123))])

pipe_xgb.fit(X_train,y_train)
print(pipe_xgb.score(X_test,y_test))
print(accuracy_score(y_test, pipe_xgb.predict(X_test)))
```

```
/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/xgboost/compat.py:93:
FutureWarning: pandas.Int64Index is deprecated and will be removed from pandas
in a future version. Use pandas.Index with the appropriate dtype instead.
  from pandas import MultiIndex, Int64Index

0.7717760180995475
0.7717760180995475
```

[ ]:
```
from sklearn import svm

pipe_svm = Pipeline([("pca", PCA(n_components = 27, random_state = 123)),
                ("svm", svm.SVC(random_state = 123))])

pipe_svm.fit(X_train,y_train)
print(pipe_svm.score(X_test,y_test))
print(accuracy_score(y_test, pipe_svm.predict(X_test)))
```

[ ]:
```
## G
# #   This cell may take several minutes to run
# from sklearn import svm
```

```
# from sklearn.pipeline import Pipeline
# from sklearn.ensemble import RandomForestClassifier
# from sklearn import tree

# ## KEEP IT FOR NOW
# # Construct some pipelines
# pipe_svm = Pipeline([('pca', PCA(n_components=27)),
#                      ('clf', svm.SVC(random_state=123))])

# pipe_tree = Pipeline([('pca', PCA(n_components=27)),
#                       ('clf', tree.DecisionTreeClassifier(random_state=123))])

# pipe_rf = Pipeline([('pca', PCA(n_components=27)),
#                     ('clf', RandomForestClassifier(random_state=123))])

# # List of pipelines and pipeline names
# pipelines = [pipe_svm, pipe_tree, pipe_rf]
# pipeline_names = ['Support Vector Machine','Decision Tree','Random Forest']

# # Loop to fit each of the three pipelines
# for pipe in pipelines:
#     print(pipe)
#     pipe.fit(X_train, y_train)

# # Compare accuracies
# for index, val in enumerate(pipelines):
#     print('%s pipeline test accuracy: %.3f' % (pipeline_names[index],
# val.score(X_test, y_test)))
```

1.9 Pipeline with grid search

Construct two pipelines with grid search: - one for random forests - try to have around 40 different models - one for the AdaBoost algorithm

1.9.1 Random Forest pipeline with grid search

[ ]:
```
# Your code here
# imports
from sklearn.model_selection import GridSearchCV

pipe_rf_II = Pipeline(
[("pca", PCA(n_components = 27)),
 ("RF", RandomForestClassifier(random_state = 123))
])

params = [{'RF__n_estimators': [120],
   'RF__criterion': ['entropy', 'gini'],
```

```
   'RF__max_depth': [4, 5, 6],
   'RF__min_samples_leaf':[0.05 ,0.1, 0.2],
   'RF__min_samples_split':[0.05 ,0.1, 0.2]
  }]
```

[ ]:
```
# Your code here
#   This cell may take a long time to run!
grid_RF = GridSearchCV(estimator  = pipe_rf_II,
                       param_grid = params,
                       scoring = "accuracy",
                       cv=3, verbose=2, return_train_score = True)

grid_RF.fit(X_train, y_train)
```

Use your grid search object along with `.cv_results` to get the full result overview

[ ]:
```
# Your code here
print(grid_RF.best_params_)
print()
print(accuracy_score(y_test, grid_RF.predict(X_test)))
```

[ ]:
```
pd.DataFrame(grid_RF.cv_results_)
```

1.9.2 AdaBoost

[ ]:
```
# Your code here
#   This cell may take several minutes to run
from sklearn.ensemble import AdaBoostClassifier

pipe_ada = Pipeline(
[("pca", PCA(n_components = 27)),
 ("ada", AdaBoostClassifier(random_state = 123))
])

params_ada = [{
    "ada__n_estimators" : [30, 50, 70],
    "ada__learning_rate": [1.0, 0.5, 0.1]
}]

grid_ada = GridSearchCV(estimator  = pipe_ada,
                        param_grid = params_ada,
                        scoring = "accuracy",
                        cv=3, verbose=0, return_train_score = True)

grid_ada.fit(X_train, y_train)

# Your code here
```

```
print(grid_ada.best_params_)
print()
print(accuracy_score(y_test, grid_ada.predict(X_test)))
```

Use your grid search object along with `.cv_results` to get the full result overview:

[ ]:
```
# Your code here
pd.DataFrame(grid_ada.cv_results_)
```

1.9.3 Level-up (Optional): SVM pipeline with grid search

As extra level-up work, construct a pipeline with grid search for support vector machines. * Make sure your grid isn't too big. You'll see it takes quite a while to fit SVMs with non-linear kernel functions!

[ ]:
```
# Your code here
#   This cell may take a very long time to run!

pipe_ada = Pipeline(
[("pca", PCA(n_components = 27)),
 ("svm", svm.SVC(random_state = 123))
])

params_svm = [
    {
     'svm__C': [0.1, 1, 10] ,
     'svm__kernel': ['linear']
    },
  {
      'svm__C': [1, 10],
      'svm__gamma': [0.001, 0.01],
      'svm__kernel': ['rbf']
  }
]

grid_svm = GridSearchCV(estimator  = pipe_svm,
                        param_grid = params_svm,
                        scoring = "accuracy",
                        cv=3, verbose=1, return_train_score = True)

grid_svm.fit(X_train, y_train)

# Your code here
print(grid_svm.best_params_)
print()
print(accuracy_score(y_test, grid_svm.predict(X_test)))
```

Use your grid search object along with `.cv_results` to get the full result overview:

[ ]:
```
# Your code here
pd.DataFrame(grid_svm.cv_results_)
```

1.10 Note

Note that this solution is only one of many options. The results in the Random Forest and AdaBoost models show that there is a lot of improvement possible by tuning the hyperparameters further, so make sure to explore this yourself!

1.11 Summary

Great! You've gotten a lot of practice in using PCA in pipelines. What algorithm would you choose and why?