SCIKIT-LEARN PIPELINES AND CROSS VALIDATION

ML Pipelines using scikit-learn and GridSearchCV

Managing ML workflows with Pipelines and using GridSearch Cross validation techniques for parameter tuning

Nikhil pentapalli
Jun 7, 2020 · 6 min read

Image from Unsplash

> *ML calculations and algorithms generally process enormous information. A pipeline is an approach to chain those information handling ventures as required in an organized manner. Not just that,it likewise helps in making incredible work process and Reproducible code.*

What is a ML Pipeline?

A pipeline is a progression of steps where information is changed. It originates from the "pipe and filter" plan design. In this way, you may have a class for each filter and afterward another class to join those means into the pipeline and make a complete final pipeline.

Methods of a Scikit-Learn Pipeline

Pipelines **must have those two methods**:

- The word “fit” is to learn on the data and acquire its state
- The word “transform” (or “predict”) to actually process the data and generate a prediction.

It’s also possible to call this method or to chain both of them:

- The word “fit_transform” is to fit and then transform the data, but all in one go, which allows for great code optimizations when these two methods must be done simultaneously.

First, we shall define the model pipelines and then we do Grid search cross validation technique to find the optimal model for our problem statement.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import feature_extraction, linear_model,
model_selection, preprocessing
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

Here I am using the bbc news dataset from kaggle for building the pipelines.

```
news=pd.read_csv('bbc-text.csv')
news.head()
```

```
Out[4]:
```

category text

	category	text
0	tech	tv future in the hands of viewers with home th...
1	business	worldcom boss left books alone former worldc...
2	sport	tigers wary of farrell gamble leicester say ...
3	sport	yeading face newcastle in fa cup premiership s...
4	entertainment	ocean s twelve raids box office ocean s twelve...

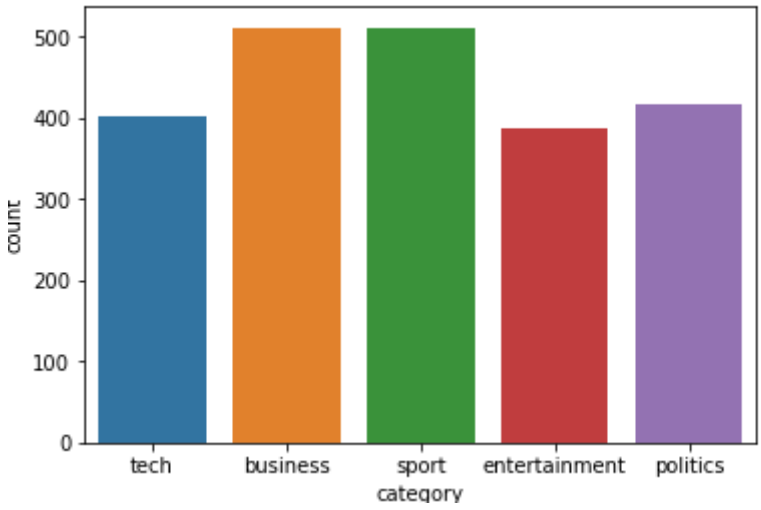
sample data

```
news['category'].value_counts()
```

```
Out[7]: sport            511
        business         510
        politics         417
        tech             401
        entertainment    386
        Name: category, dtype: int64
```

Value Counts of each category

```
sns.countplot(x="category", data=news)
```



Plot of each category

Logistic Regression Classifier

```
x_train,x_test,y_train,y_test = train_test_split(news['text'],
news.category, test_size=0.2, random_state=2020)
```

```
pipe_lr = Pipeline([('vect', CountVectorizer()),
 ('tfidf', TfidfTransformer()),
 ('model', LogisticRegression())])

model = pipe_lr.fit(x_train, y_train)
prediction = model.predict(x_test)
print("\naccuracy: {}%".format(round(accuracy_score(y_test,
prediction)*100,2)))

print('\n',confusion_matrix(y_test, prediction))

print('\n',classification_report(y_test, prediction))
```

```
accuracy: 96.63%

 [[ 97   0   0   1   3]
 [  1  80   1   0   0]
 [  3   0  70   1   1]
 [  0   0   0 105   0]
 [  4   0   0   0  78]]

               precision    recall  f1-score   support

     business       0.92      0.96      0.94       101
entertainment       1.00      0.98      0.99        82
     politics       0.99      0.93      0.96        75
        sport       0.98      1.00      0.99       105
         tech       0.95      0.95      0.95        82

  avg / total       0.97      0.97      0.97       445
```

complete report of the model

We imported pipeline from scikit-learn library.In the pipeline we can define the function that has to be performed in a sequence.Here we first split our data into train,test using train_test_split.

Then we defined CountVectorizer, Tf-Idf, Logistic regression in an order in our pipeline.This way it reduces the amount of code and pipelining the model helps in comparing it with different models and getting an optimal model for our choice.

Similarly let's create pipelines for different models with the same data as input and select the best model out of everything.

Creation of different Pipelines:

```
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.externals import joblib
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm

x_train,x_test,y_train,y_test = train_test_split(news['text'], news.category, test_size=0.2, ra


#Now let us contruct some pipelines of our choice.

pipe_lr = Pipeline([('vect', CountVectorizer()),
                    ('tfidf', TfidfTransformer()),
                   ('clf', LogisticRegression(random_state=42))])

pipe_dt = Pipeline([('vect', CountVectorizer()),
                    ('tfidf', TfidfTransformer()),
                    ('model', DecisionTreeClassifier(random_state=42))])

pipe_rf = Pipeline([('vect', CountVectorizer()),
                    ('tfidf', TfidfTransformer()),
               ('clf', RandomForestClassifier(random_state=42))])


pipe_svm = Pipeline([('vect', CountVectorizer()),
                    ('tfidf', TfidfTransformer()),
               ('clf', svm.SVC(random_state=42))])
```

pipeline creation.py hosted with ♡ by GitHub view raw

In the above code i constructed four pipelines for Four different models.

1. Logistic Regression
2. Decision Tree
3. Random Forest
4. Support Vector Machine

You can absolutely change the architecture of the single pipeline. For example, in Logistic Regression if you feel the target values needs to be scaled you can use standard scaler or you can absolutely use some preprocessing techniques.

Let us assume a case where preprocessing is needed and how to handle such case.

```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder

num_transformer =
Pipeline([('imputer',SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())])

cat_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='constant',
fill_value='missing')),
('onehot',OneHotEncoder(handle_unknown='ignore'))])
```

So i have created two separate pipelines to handle different types of data that is num_transformer handles the missing numbers using simple imputer while the standard scaler will transform your data such that its distribution will have a mean value 0 and standard deviation of 1.

In cat_transformer i have used simple imputer and one hot encoder to encode the categories.

Now let us take the columns of the numerical features as well as categorical features and apply the pipeline of preprocessing steps described above.To do that i take the train_data and select the specific columns as shown below.

```
num_features = train_data.select_dtypes(include=['int64']).columns
cat_features = train_data.select_dtypes(include=['object']).columns
```

Once we are ready we can now push everything into our column Transformer which can be imported from sklearn.compose

```
from sklearn.compose import ColumnTransformer
```

```
preprocessor = ColumnTransformer([
        ('num', num_transformer, num_features),
        ('cat', cat_transformer, cat_features)])
```

So Now unlike above i just did all the preprocessing in one step which can be used for multiple models.

```
pipe_rf = Pipeline([('preprocess',preprocessor),
 ('clf', RandomForestClassifier(random_state=42))])
```

Let us now fit the models using GridSearchCV which helps us in model selection by passing many different params for each pipeline and getting the best model as well as best params with which the model was fit using. So let's get started by defining some params for grid search.

Linear Regression takes l2 penalty by default.so i would like to experiment with l1 penalty.Similarly for Random forest in the selection criterion i could want to experiment on both 'gini' and 'entropy'. So i have passed both the values in to the clf_criterion

You can also experiment on different kernels of your choice for the specific problem statement and use case.

```
# Set grid search params
param_range = [9, 10]
param_range_fl = [1.0, 0.5]

grid_params_lr = [{'clf__penalty': ['l1', 'l2'],
        'clf__C': param_range_fl,
        'clf__solver': ['liblinear']}]


grid_params_rf = [{'clf__criterion': ['gini', 'entropy'],
        'clf__max_depth': param_range,
        'clf__min_samples_split': param_range[1:]}]

grid_params_svm = [{'clf__kernel': ['linear', 'rbf'],
        'clf__C': param_range}]

# Construct grid searches
jobs = -1
```

```
LR = GridSearchCV(estimator=pipe_lr,
            param_grid=grid_params_lr,
            scoring='accuracy',
            cv=10)



RF = GridSearchCV(estimator=pipe_rf,
            param_grid=grid_params_rf,
            scoring='accuracy',
            cv=10,
            n_jobs=jobs)


SVM = GridSearchCV(estimator=pipe_svm,
            param_grid=grid_params_svm,
            scoring='accuracy',
            cv=10,
            n_jobs=jobs)



# List of pipelines for iterating through each of them
grids = [LR,RF,SVM]

# Creating a dict for our reference
grid_dict = {0: 'Logistic Regression',
        1: 'Random Forest',
        2: 'Support Vector Machine'}

# Fit the grid search objects
print('Performing model optimizations...')
best_acc = 0.0
best_clf = 0
best_gs = ''
for idx, gs in enumerate(grids):
    print('\nEstimator: %s' % grid_dict[idx])
    gs.fit(x_train, y_train)
    print('Best params are : %s' % gs.best_params_)
    # Best training data accuracy
    print('Best training accuracy: %.3f' % gs.best_score_)
    # Predict on test data with best params
    y_pred = gs.predict(x_test)
    # Test data accuracy of model with best params
    print('Test set accuracy score for best params: %.3f ' % accuracy_score(y_test, y_pred))
    # Track best (highest test accuracy) model
    if accuracy_score(y_test, y_pred) > best_acc:
```

```
        best_acc = accuracy_score(y_test, y_pred)
        best_gs = gs
        best_clf = idx
print('\nClassifier with best test set accuracy: %s' % grid_dict[best_clf])

# Save best grid search pipeline to file
dump_file = 'best_grid_search_pipeline.pkl'
joblib.dump(best_gs, dump_file, compress=1)
print('\nSaved %s grid search pipeline to file: %s' % (grid_dict[best_clf], dump_file))
```

OUTPUT:

```
Performing model optimizations...

Estimator: Logistic Regression
Best params: {'clf__C': 1.0, 'clf__penalty': 'l2', 'clf__solver':
'liblinear'}
Best training accuracy: 0.969
Test set accuracy score for best params: 0.966

Estimator: Random Forest
Best params: {'clf__criterion': 'gini', 'clf__max_depth': 10,
'clf__min_samples_split': 10}
Best training accuracy: 0.844
Test set accuracy score for best params: 0.836

Estimator: Support Vector Machine
Best params: {'clf__C': 9, 'clf__kernel': 'linear'}
Best training accuracy: 0.978
Test set accuracy score for best params: 0.971

Classifier with best test set accuracy: Support Vector Machine

Saved Support Vector Machine grid search pipeline to file:
best_grid_search_pipeline.pkl
```

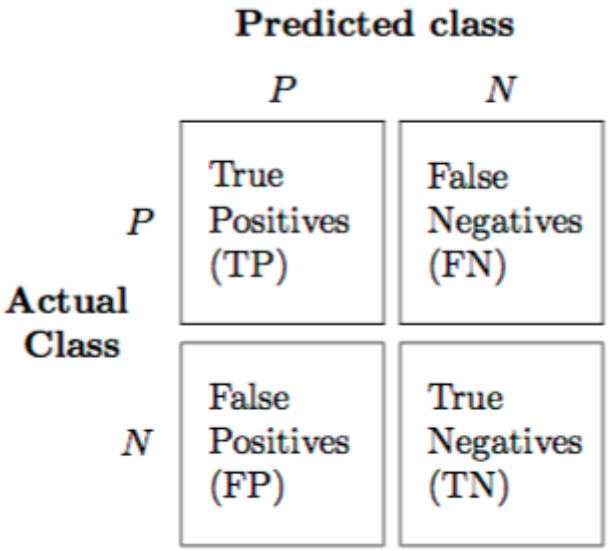
Now as i have compared Logistic Regression, Random Forest and SVM in which i could definitely see that SVM is the best model with an accuracy of 0.978 .we also obtained the best parameters from the Grid Search cross validation.

So i have taken accuracy as a scoring parameter. It is completely based on the problem statement.You can take scoring parameter as recall or f1-score or precision which are completely derived from confusion matrix

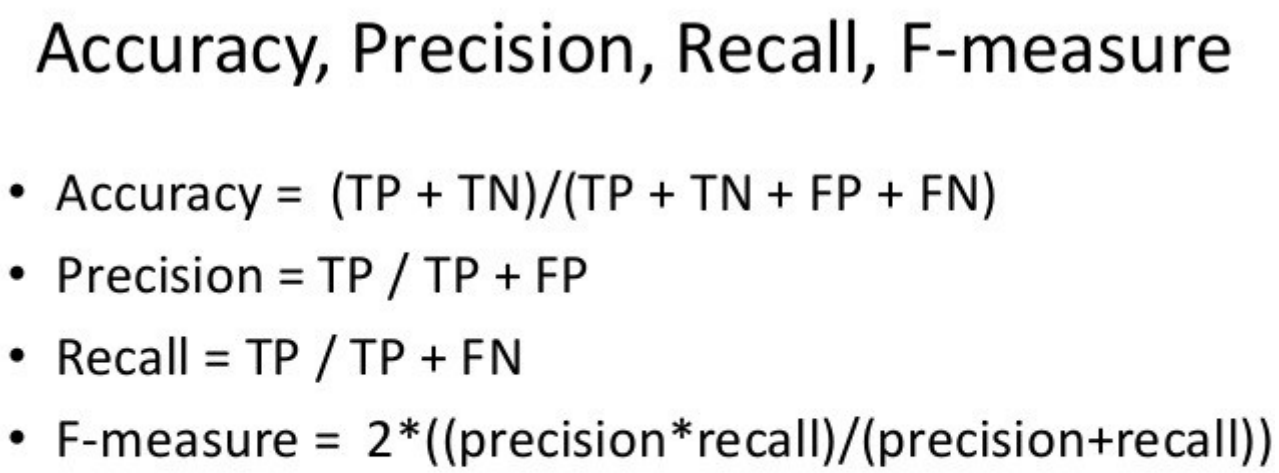
Finally, Grid search builds a model for every combination of hyper parameters specified and evaluates each model. Another efficient technique for hyper parameter

tuning is the Randomized search — where random combinations of the hyper parameters are used to find the best solution.

The Confusion matrix and the scoring parameters can be understood from the below images.



Confusion Matrix(image by author)



Scoring Parameters(image by author)

You can define your own custom transformers which should definitely contain fit and transform methods in it.if you had noticed i had left the column transformer for you to try on your own.

Voila, there you go. Now you can try building your own machine learning pipeline and do not forget to try a custom transformer. Also change the parameters in the grid search and run experiments. The best way to learn anything is by experimenting with it.

I hope this article empowers your knowledge. Keep supporting and Happy Learning.

Nikhil Pentapalli - Data Scientist || Machine Learning Engineer - Jio Platforms Limited | LinkedIn

Experience in Real world AI product building. ->Experienced in transferring real problem into requirements and solution...

www.linkedin.com

Sign up for Analytics Vidhya News Bytes

By Analytics Vidhya

Latest news from Analytics Vidhya on our Hackathons and some of our best articles! Take a look.

Your email

Get this newsletter

By signing up, you will create a Medium account if you don't already have one. Review our Privacy Policy for more information about our privacy practices.

Gridsearchcv Scikit Learn Machine Learning Data Science Model Tuning

About Help Legal

Get the Medium app

Download on the App Store GET IT ON Google Play