Make Model Training and Testing Easier with MultiTrain

LIBRARIES MACHINE LEARNING PYTHON

This article was published as a part of the Data Science Blogathon.

Introduction

For data scientists and machine learning engineers, developing and testing machine learning models may take a lot of time. For instance, you would need to write a few lines of code, wait for each model to run, and then go on to the following five models to train and test. This may be rather tedious. However, when I encountered a similar issue, I became so frustrated that I began to devise a way to make things simpler for myself. After four months of hard work – coding, and bug-fixing, I'm happy to share my solution with you.

MultiTrain is a Python module I created that allows you to train many machine learning models on the same dataset to analyze performance and choose the best models. The content of this article will show you how to utilize MultiTrain for a basic regression problem. Please visit here to learn how to use it for classification problems.

Now, code alongside me as we simplify the model training and testing of machine learning models with MultiTrain.

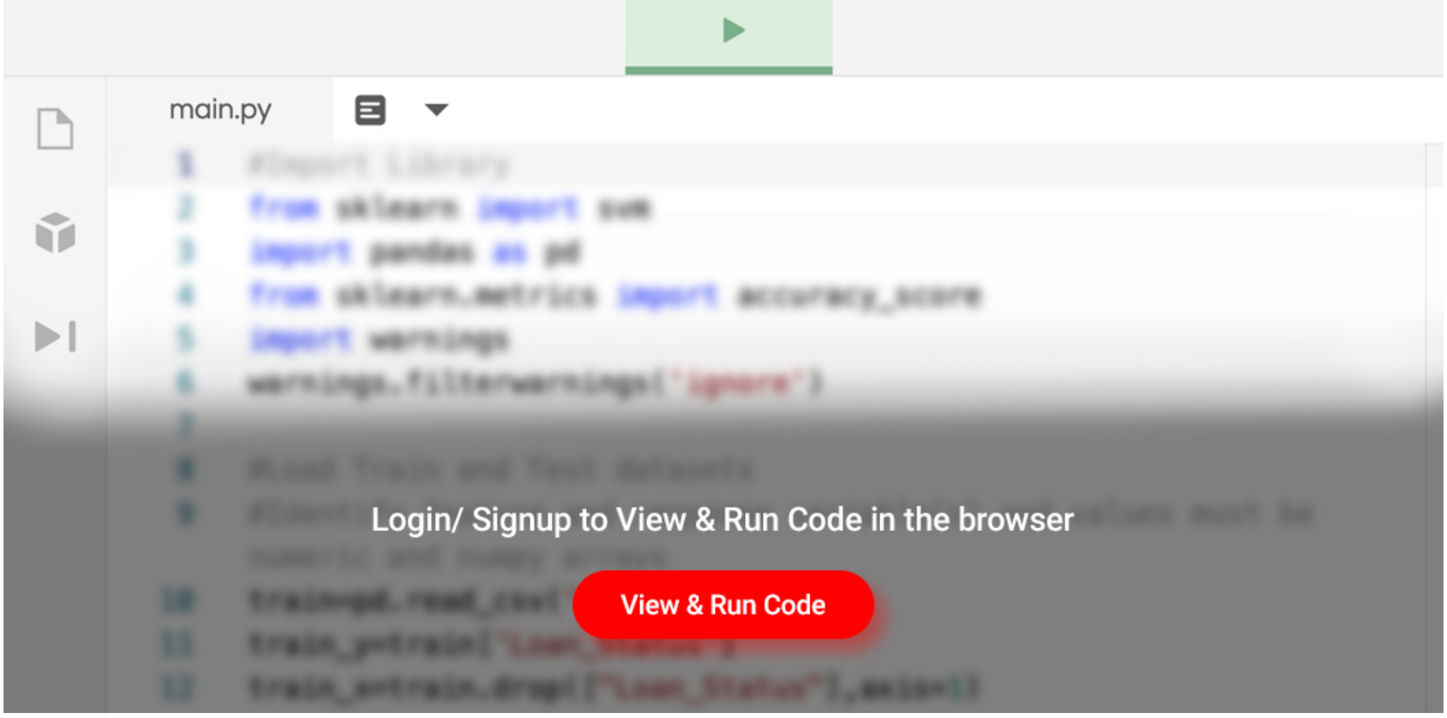
Importing Libraries and Dataset

Today, we will be testing which machine learning model would work best for the **productivity prediction of garment employees.** The dataset is available here on Kaggle for download.

To start working on our dataset, we need to import some libraries.

To import our dataset, you must ensure it is in the same path as your ipynb file, or else you will have to set a file path.

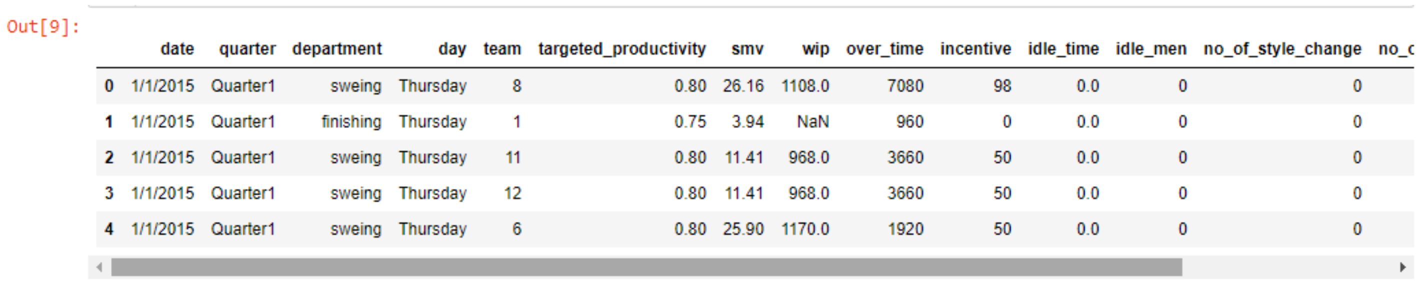
Python Code:



```
#Import Library
from sklearn import svm
import pandas as pd
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings('ignore')

#Load Train and Test datasets
#Identify [illegible] values must be
numeric and numpy arrays
train=pd.read_csv([illegible]
train_y=train['Loan_Status']
train_x=train.drop(["Loan_Status"],axis=1)
```

Now that we have imported our dataset into our Jupyter notebook, we want to see the first five rows of the dataset. You need to use a single line of code for that.



Out[9]:

	date	quarter	department	day	team	targeted_productivity	smv	wip	over_time	incentive	idle_time	idle_men	no_of_style_change	no_c
0	1/1/2015	Quarter1	sweing	Thursday	8	0.80	26.16	1108.0	7080	98	0.0	0	0	
1	1/1/2015	Quarter1	finishing	Thursday	1	0.75	3.94	NaN	960	0	0.0	0	0	
2	1/1/2015	Quarter1	sweing	Thursday	11	0.80	11.41	968.0	3660	50	0.0	0	0	
3	1/1/2015	Quarter1	sweing	Thursday	12	0.80	11.41	968.0	3660	50	0.0	0	0	
4	1/1/2015	Quarter1	sweing	Thursday	6	0.80	25.90	1170.0	1920	50	0.0	0	0	

Note: Not all columns are shown here; only a few we would be working with are present in this snapshot except the output column.

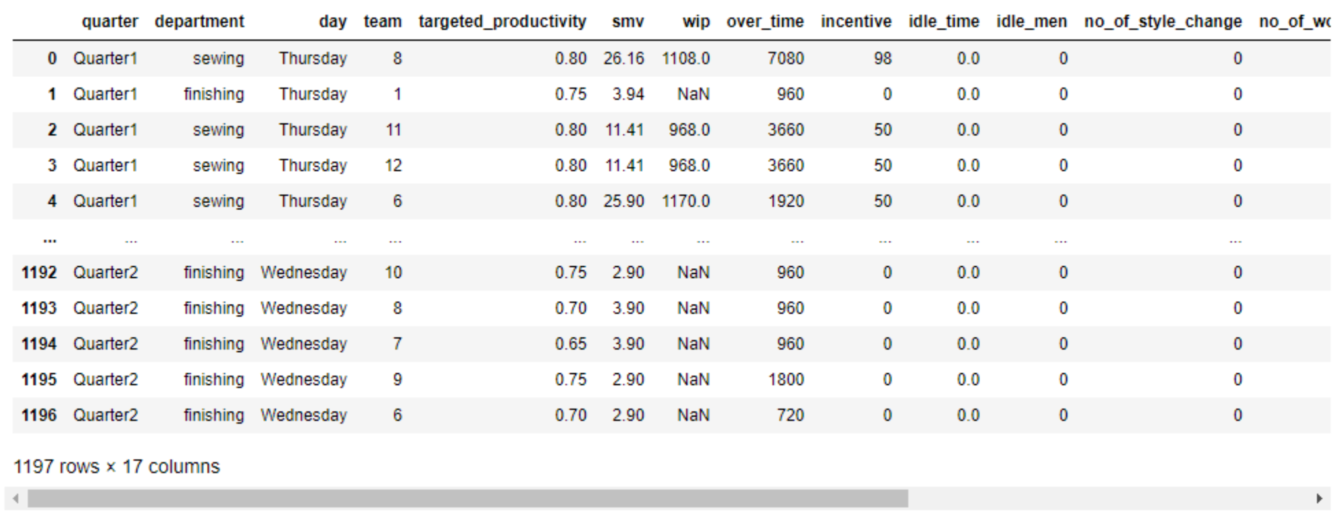
Data Preprocessing

We wouldn't be employing any primary data preprocessing techniques or EDA as our focus is on how to train and test lots of models at once using the MultiTrain library. I strongly encourage you to perform some major preprocessing techniques on your dataset, as dirty data can affect your model's predictions. It also affects the performance of machine learning algorithms.

While you check the first five rows, you should find a column named "department", in which *sewing* is spelt as sweing. We can fix this spelling mistake with this line of code.

```
df["department"] = df["department"].str.replace('sweing', 'sewing')
```

Out[23]:



	quarter	department	day	team	targeted_productivity	smv	wip	over_time	incentive	idle_time	idle_men	no_of_style_change	no_of_wo
0	Quarter1	sewing	Thursday	8	0.80	26.16	1108.0	7080	98	0.0	0	0	
1	Quarter1	finishing	Thursday	1	0.75	3.94	NaN	960	0	0.0	0	0	
2	Quarter1	sewing	Thursday	11	0.80	11.41	968.0	3660	50	0.0	0	0	
3	Quarter1	sewing	Thursday	12	0.80	11.41	968.0	3660	50	0.0	0	0	
4	Quarter1	sewing	Thursday	6	0.80	25.90	1170.0	1920	50	0.0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
1192	Quarter2	finishing	Wednesday	10	0.75	2.90	NaN	960	0	0.0	0	0	
1193	Quarter2	finishing	Wednesday	8	0.70	3.90	NaN	960	0	0.0	0	0	
1194	Quarter2	finishing	Wednesday	7	0.65	3.90	NaN	960	0	0.0	0	0	
1195	Quarter2	finishing	Wednesday	9	0.75	2.90	NaN	1800	0	0.0	0	0	
1196	Quarter2	finishing	Wednesday	6	0.70	2.90	NaN	720	0	0.0	0	0	

1197 rows × 17 columns

We can see in this snapshot above that the spelling mistake is now corrected.

When you run the following lines of code, you will discover that the "department" column has some duplicate values that we have to take care of, and we will also need to fix that before we can start predicting.

```
print(f'Unique Values in Department before cleaning: {df.department.unique()}') Output: Unique Values in Department before cleaning: ['sewing' 'finishing ' 'finishing']
```

To fix this problem:

```
df['department'] = df.department.str.strip() print(f'Unique Values in Department after cleaning: {df.department.unique()}') Output: Unique Values in Department before cleaning: ['sewing', 'finishing']
```

Let's replace all missing values in our dataset with an integer value of 0

```
for i in df.columns:     if df[i].isnull().sum() != 0:         df[i] = df[i].fillna(0)
```

Most scikit-learn algorithms can't handle categorical data, so we need to convert the columns with categorical data to numerical with label encoding. There are several other methods of encoding categorical data, e.g. ordinal encoding, target encoding, binary encoding, frequency encoding, mean encoding, and others. The listed methods have advantages and disadvantages, and it's advisable to test as many as possible to see which works best for your dataset in terms of performance.

As I mentioned previously, we would use a label encoder for this tutorial to encode our categorical columns. Firstly, we have to get a list of our categorical columns. We can do that using the following lines of code.

```
cat_col = [] num_col = [] for i in df.columns:     if df[i].dtypes == object:         cat_col.append(i)
else:         num_col.append(i) #remove the target columns num_col.remove('actual_productivity')
```

Now that we have gotten a list of our categorical columns inside the cat_col variable. We can now apply our label encoder to it to encode the categorical data into numerical data.

```
label = LabelEncoder()     for i in cat_col:         df[i] = label.fit_transform(df[i])
```

All missing values formerly indicated by NaN in column ‘wip’ has now changed to 0 and the three categorical columns – quarter, department, and day have all been label encoded.

You may still need to fix outliers in the dataset and do some feature engineering on your own.

Model Training

Before we can begin model training, we will need to split our dataset into its training features and labels.

```
features = df.drop('actual_productivity', axis=1) labels = df['actual_productivity']
```

Now, we would need to split the dataset into training and tests. The training sets are used to train the machine learning algorithms and the test sets are used to evaluate their performance.

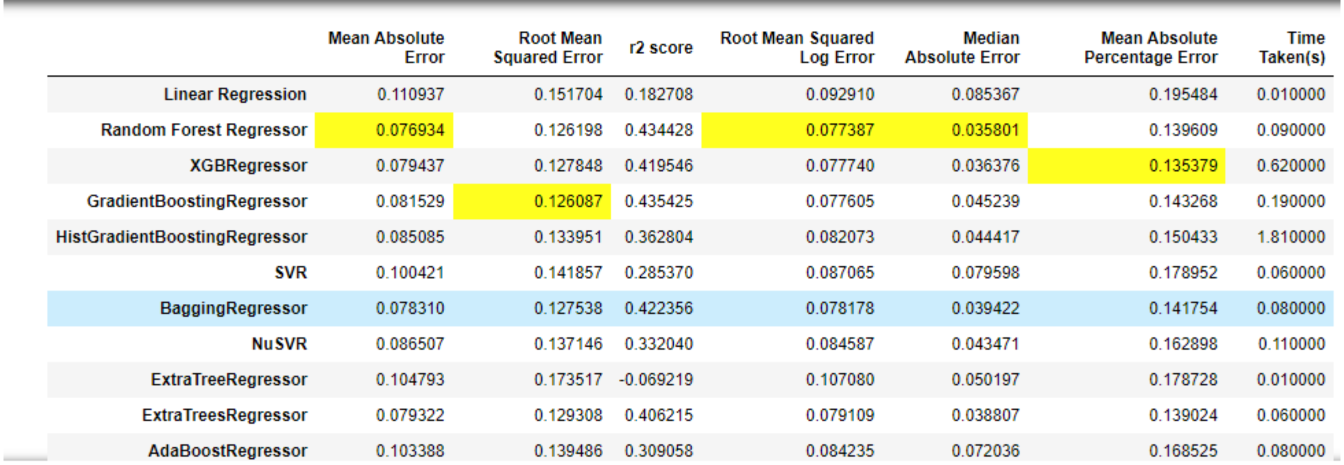
```
train = MultiRegressor(random_state=42,                        cores=-1,                        verbose=True)
```

```
split = train.split(X=features, y=labels, sizeOfTest=0.2, randomState=42, normalize='StandardScaler',
columns_to_scale=num_col, shuffle=True)
```

The normalize parameter in the split method allows you to scale your numerical columns by just passing in any scalers of your choice; the columns_to_scale parameter then receives a list of the columns you’d like to scale instead of having to scale all columns automatically.

After splitting the features and labels into train, test is appended to a variable named split. This variable then holds X_train, X_test, y_train, and y_test; we would need it in the next function below.

```
fit = train.fit(X=features, y=labels, splitting=True, split_data=split)
```



	Mean Absolute Error	Root Mean Squared Error	r2 score	Root Mean Squared Log Error	Median Absolute Error	Mean Absolute Percentage Error	Time Taken(s)
Linear Regression	0.110937	0.151704	0.182708	0.092910	0.085367	0.195484	0.010000
Random Forest Regressor	0.076934	0.126198	0.434428	0.077387	0.035801	0.139609	0.090000
XGBRegressor	0.079437	0.127848	0.419546	0.077740	0.036376	0.135379	0.620000
GradientBoostingRegressor	0.081529	0.126087	0.435425	0.077605	0.045239	0.143268	0.190000
HistGradientBoostingRegressor	0.085085	0.133951	0.362804	0.082073	0.044417	0.150433	1.810000
SVR	0.100421	0.141857	0.285370	0.087065	0.079598	0.178952	0.060000
BaggingRegressor	0.078310	0.127538	0.422356	0.078178	0.039422	0.141754	0.080000
NuSVR	0.086507	0.137146	0.332040	0.084587	0.043471	0.162898	0.110000
ExtraTreeRegressor	0.104793	0.173517	-0.069219	0.107080	0.050197	0.178728	0.010000
ExtraTreesRegressor	0.079322	0.129308	0.406215	0.079109	0.038807	0.139024	0.060000
AdaBoostRegressor	0.103388	0.139486	0.309058	0.084235	0.072036	0.168525	0.080000

Run this code in your notebook to view the full model list and scores.

Visualize Model Results

For the sake of people who might prefer to view the model performance results in charts rather than a dataframe. There's also an option available for you to convert the dataframe into charts. All you have to do is to run the code below.

```
train.show(param=fit,            t_split=True)
```

Conclusion

MultiTrain exists to help data scientists and machine learning engineers make their job easy. It eliminates repetition and the boring process. With just a few lines of code, you can get your model training and testing immediately.

The assessment metrics shown on the dataframe might also differ based on the problem you're attempting to solve, such as multiclass, binary, classification, regression, imbalanced datasets, or balanced datasets. You have even more freedom to work on these challenges by passing values to parameters in different methods rather than writing extensive lines of code.

After fitting the models, the results generated in the dataframe shouldn't be your final results. Since MultiTrain aims to determine the best models that work best for your particular use case, you should pick the best models and perform some hyperparameter tuning on them or even feature engineering on your dataset to further boost the performance.

Article Url - https://www.analyticsvidhya.com/blog/2022/09/make-model-training-and-testing-easier-with-multitrain/

[Samson Shittu]()