Surya Group Of Instutions

Naan Mudhalvan

IBM -Artificial Intelligence

Sadhasivam M

42221104032

Team - 08

AI-Driven Exploration And Predication Of Company Registration Trends With Register Of Companies

Design Definition :

This kernel has been divided into four parts. The first part deals with the development of a baseline model. This model should allow us to quickly understand the problem and the data.

Afterwards, we will go into detail. Data will be studied and enriched through exploratory data analysis and feature extraction, to improve the performance of our machine learning model. Finally, some conclusions will be drawn from this kernel and its impact in our data science journey.

Belfast, an earlier incubator :

Incubators are companies that support the creation of startups and their first years of activity. They are important because they help entrepreneurs solve some issues commonly associated with running a business, such as workspace, training, and seed funding.

Our engineering masterpiece also needs a starting point. In this section, we start the assemblage of our work by importing some libraries and general functions.

Imports :

```
# Import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# Put this when it's called
from sklearn.model_selection import train_test_split
from sklearn.model_selection import learning_curve
from sklearn.model_selection import validation_curve
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
```

Functions :

```
# Create table for missing data analysis
def draw_missing_data_table(df):
    total =
df.isnull().sum().sort_values(ascending=False)
```

```
    percent = (df.isnull().sum()/df.isnull().count()).sort_values(ascending=False)
    missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
    return missing_data


# Plot learning curve
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
```

```
                color="r")
    plt.fill_between(train_sizes, test_scores_mean -
test_scores_std,
                test_scores_mean + test_scores_std,
alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-',
color="r",
            label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
            label="Validation score")


    plt.legend(loc="best")
    return plt
# Plot validation curve

def plot_validation_curve(estimator, title, X, y,
param_name, param_range, ylim=None, cv=None,

                  n_jobs=1, train_sizes=np.linspace(.1, 1.0,
5)):

    train_scores, test_scores =
validation_curve(estimator, X, y, param_name,
param_range, cv)

    train_mean = np.mean(train_scores, axis=1)

    train_std = np.std(train_scores, axis=1)

    test_mean = np.mean(test_scores, axis=1)

    test_std = np.std(test_scores, axis=1)

    plt.plot(param_range, train_mean, color='r',
marker='o', markersize=5, label='Training score')
```

```
    plt.fill_between(param_range, train_mean +
train_std, train_mean - train_std, alpha=0.15, color='r')

    plt.plot(param_range, test_mean, color='g',
linestyle='--', marker='s', markersize=5,
label='Validation score')

    plt.fill_between(param_range, test_mean + test_std,
test_mean - test_std, alpha=0.15, color='g')

    plt.grid()

    plt.xscale('log')

    plt.legend(loc='best')

    plt.xlabel('Parameter')

    plt.ylabel('Score')

    plt.ylim(ylim)
```

1. The lean data set :

1.Build. Figure out the problem that needs to be solved, generate ideas about how to solve it, and select the best one. Turn your best idea into a Minimum Viable Product (MVP).

2.Measure. Test your product. Go to your customers and measure their reactions and behaviors against your product.

3.Learn. Analyse the data you collected when testing the product with your customers. Draw conclusions from the experiment and decide what to do next. In other words, this is a validated learning

process that quickly builds, tests, and rebuilds products, according to users' feedback. This reduces your market risks by failing fast and cheap, to get you closer and closer to what the market really needs.

1.1. Doing the pitch :

Let's return the first rows of our data set to get a clear and concise picture of what is there and what we can do with it.

```
# Import data
df = pd.read_csv('../input/train.csv')
df_raw = df.copy()  # Save original data set, just in case.

# Overview
df.head()
```

1.2. Showing the numbers :

In the same way, we will generate the descriptive statistics to get the basic quantitative information about the features of our data set.

```
# Descriptive statistics
df.describe()
```

There are three aspects that usually catch my attention when I analyse descriptive statistics :

1. **Min and max values**.
2. **Mean and standard deviation**.
3. **Count**.

1.3. Filling the gaps :

There are several strategies to deal with missing data. Some of the most common are:

- Use only valid data, deleting the cases where data is missing.
- Impute data using values from similar cases or using the mean value.
- Impute data using model-based methods, in which models are defined to predict the missing values.

Now that we can see the tip of the iceberg, let's dive into the subject.

```
# Analyse missing data
draw_missing_data_table(df)
```

First thoughts:

- 'Cabin' has too many missing values (>25%). Dogma! We need to delete this variable right away.
- 'Age' can be imputed. For now, I'll associate a value that allows me to know that I'm imputing data. Later, I'll revise this strategy.
- Due to the low percentage of missing values, I'll delete the observations where we don't know 'Embarked'.

```
# Drop Cabin
df.drop('Cabin', axis=1, inplace=True)
df.head()
```

```
# Fill missing values in Age with a specific value
value = 1000
df['Age'].fillna(1000, inplace=True)
df['Age'].max()
```

Output :

1000.0

1.4. Minimum viable model :

The authors propose a practical four-steps methodology:

1. Select a performance metric and a target value for this metric. This metric will guide your work and allow you to know how well you're performing. In our case, our performance metric will be 'accuracy' because it is the one defined by Kaggle.
2. Quickly set up a working end-to-end pipeline. This should allow you to estimate the selected performance metric.
3. Monitor the system to understand its behaviour, in particular to understand whether its poor performance is related to underfitting, overfitting or defects.
4. Improve the system by iteration. Here we can apply feature engineering, tune hyperparameters or even change the algorithm, according to the outputs of our monitoring system.

1.4.1. Preparing the data :

```
# Data types
df.dtypes
```

Output :

```
PassengerId     int64
```

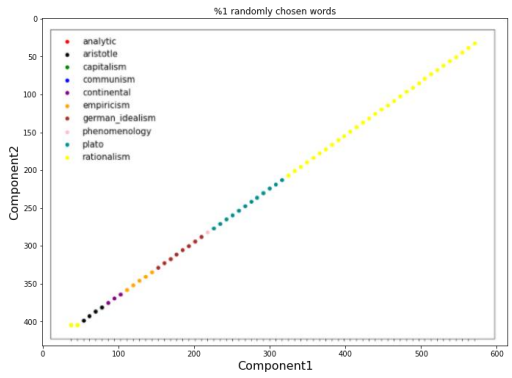
```
Survived       int64
Pclass         int64
Name          object
Sex           object
Age          float64
SibSp          int64
Parch          int64
Ticket        object
Fare         float64
Embarked      object
dtype: object
```

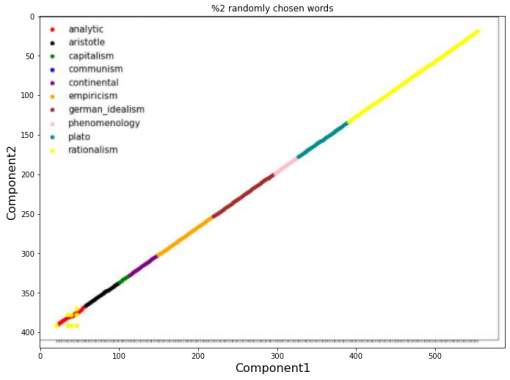
1.4.2. Launching the model :

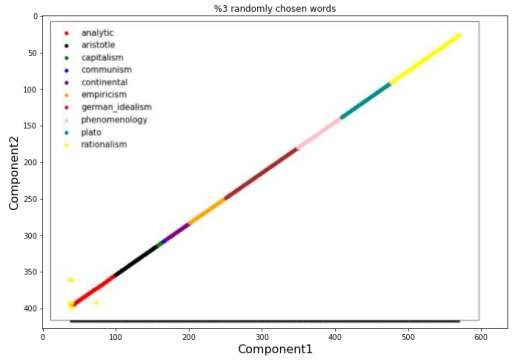
```
# Fit logistic regression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

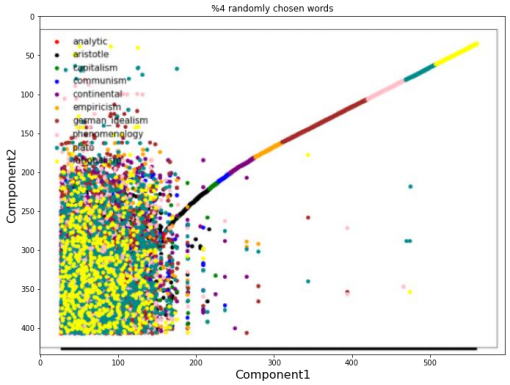
Output :

```
    LogisticRegression(C=1.0, class_weight=None, dua
l=False, fit_intercept=True,
              intercept_scaling=1, l1_ratio=None, max_ite
r=100,
              multi_class='warn', n_jobs=None, penalty='l
2',
              random_state=None, solver='warn', tol=0.0
001, verbose=0,
              warm_start=False)
```









We will define this difficultness of reading as (for each philosopher and then school):

- $UncommonWordDensity = [\frac{\sum_{x=FirstIndex}^{LastIndex} NumNotions}{\sum_{x=FirstIndex}^{LastIndex} NumberOfWords}$

```
my_cmap = plt.get_cmap("viridis")
rescale = lambda y: (y - np.min(y)) / (np.max(y) - np.min(y))

Uncommonness=(DF.groupby("author").sum()["NumOfNotions"]/DF.groupby("autho
r").sum()["NumOfWords"]).sort_values()
fig,ax=plt.subplots(figsize=(30,10))
plt.bar(Uncommonness.index,Uncommonness.values,color=my_cmap(rescale(Unco
mmonness.values)))
ax.tick_params(labelsize=15)
for item in ax.xaxis.get_ticklabels():
```

```
    item.set_rotation(40)
plt.xlabel('Philosopher', fontsize=18)
plt.ylabel('Uncommon word density', fontsize=16)
plt.title("Difficultness of reading according to use of uncommon words",fontsize=30)
```