

[Open in app](#)[Sign up](#)[Sign in](#)**Medium**

Search

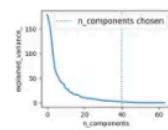
Building A Logistic Regression in Python, Step by Step



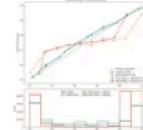
Susan Li · Follow

Published in Towards Data Science

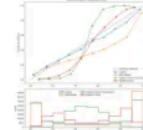
9 min read · Sep 29, 2017

[Listen](#)[Share](#)

Pipelining: chaining a PCA and a logistic regression



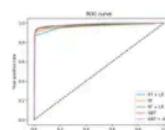
Probability Calibration curves



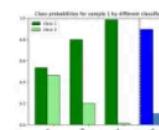
Comparison of Calibration of Classifiers



Plot classification probability



Feature transformations with ensembles of trees



Plot class probabilities calculated by the VotingClassifier

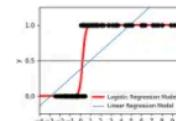
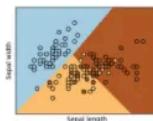


Photo Credit: Scikit-Learn

Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.). In other words, the logistic regression model predicts P(Y=1) as a function of X.

Logistic Regression Assumptions

- Binary logistic regression requires the dependent variable to be binary.
- For a binary regression, the factor level 1 of the dependent variable should represent the desired outcome.
- Only the meaningful variables should be included.
- The independent variables should be independent of each other. That is, the model should have little or no multicollinearity.
- The independent variables are linearly related to the log odds.
- Logistic regression requires quite large sample sizes.

Keeping the above assumptions in mind, let's look at our dataset.

Data

The dataset comes from the [UCI Machine Learning repository](#), and it is related to direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict whether the client will subscribe (1/0) to a term deposit (variable y). The dataset can be downloaded from [here](#).

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt
plt.rc("font", size=14)
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import seaborn as sns
sns.set(style="white")
sns.set(style="whitegrid", color_codes=True)
```

The dataset provides the bank customers' information. It includes 41,188 records and 21 fields.

```
In [39]: data = pd.read_csv('bank.csv', header=0)
data = data.dropna()
print(data.shape)
print(list(data.columns))

(41188, 21)
['age', 'job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays', 'previous', 'poutcome', 'emp_var_rate', 'cons_price_idx', 'cons_conf_idx', 'euribor3m', 'nr_employed', 'y']

In [37]: data.head()

Out[37]:
   age      job marital education default housing loan contact month day_of_week ... campaign pdays previous poutcome emp_var_rate
0  44  blue-collar  married    basic.4y  unknown    yes    no  cellular    aug    thu ...        1     999      0 nonexistent    1.4
1  53  technician  married  unknown    no    no  cellular    nov    fri ...        1     999      0 nonexistent   -0.1
2  28 management  single  university.degree    no    yes    no  cellular    jun    thu ...        3      6      2   success   -1.7
3  39    services  married  high.school    no    no    no  cellular    apr    fri ...        2     999      0 nonexistent   -1.8
4  55    retired  married    basic.4y    no    yes    no  cellular    aug    fri ...        1      3      1   success   -2.9

5 rows × 21 columns
```

Figure 1

Input variables

1. age (numeric)
2. job : type of job (categorical: “admin”, “blue-collar”, “entrepreneur”, “housemaid”, “management”, “retired”, “self-employed”, “services”, “student”, “technician”, “unemployed”, “unknown”)
3. marital : marital status (categorical: “divorced”, “married”, “single”, “unknown”)
4. education (categorical: “basic.4y”, “basic.6y”, “basic.9y”, “high.school”, “illiterate”, “professional.course”, “university.degree”, “unknown”)
5. default: has credit in default? (categorical: “no”, “yes”, “unknown”)
6. housing: has housing loan? (categorical: “no”, “yes”, “unknown”)
7. loan: has personal loan? (categorical: “no”, “yes”, “unknown”)
8. contact: contact communication type (categorical: “cellular”, “telephone”)
9. month: last contact month of year (categorical: “jan”, “feb”, “mar”, ..., “nov”, “dec”)
10. day_of_week: last contact day of the week (categorical: “mon”, “tue”, “wed”, “thu”, “fri”)
11. duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). The duration is not known before a call is performed, also, after the end of the call, y

is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model

12. campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
13. pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
14. previous: number of contacts performed before this campaign and for this client (numeric)
15. poutcome: outcome of the previous marketing campaign (categorical: “failure”, “nonexistent”, “success”)
16. emp.var.rate: employment variation rate — (numeric)
17. cons.price.idx: consumer price index — (numeric)
18. cons.conf.idx: consumer confidence index — (numeric)
19. euribor3m: euribor 3 month rate — (numeric)
20. nr.employed: number of employees — (numeric)

Predict variable (desired target):

y — has the client subscribed a term deposit? (binary: “1”, means “Yes”, “0” means “No”)

The education column of the dataset has many categories and we need to reduce the categories for a better modelling. The education column has the following categories:

```
In [4]: data['education'].unique()
Out[4]: array(['basic.4y', 'unknown', 'university.degree', 'high.school',
   'basic.9y', 'professional.course', 'basic.6y', 'illiterate'], dtype=object)
```

Figure 2

Let us group “basic.4y”, “basic.9y” and “basic.6y” together and call them “basic”.

```
data['education']=np.where(data['education'] =='basic.9y', 'Basic',
data['education'])
data['education']=np.where(data['education'] =='basic.6y', 'Basic',
data['education'])
data['education']=np.where(data['education'] =='basic.4y', 'Basic',
data['education'])
```

After grouping, this is the columns:

```
In [6]: data['education'].unique()
Out[6]: array(['Basic', 'unknown', 'university.degree', 'high.school',
   'professional.course', 'illiterate'], dtype=object)
```

Figure 3

Data exploration

```
In [7]: data['y'].value_counts()
Out[7]: 0    36548
1    4640
Name: y, dtype: int64

In [17]: sns.countplot(x='y',data=data, palette='hls')
plt.show()
plt.savefig('count_plot')
```

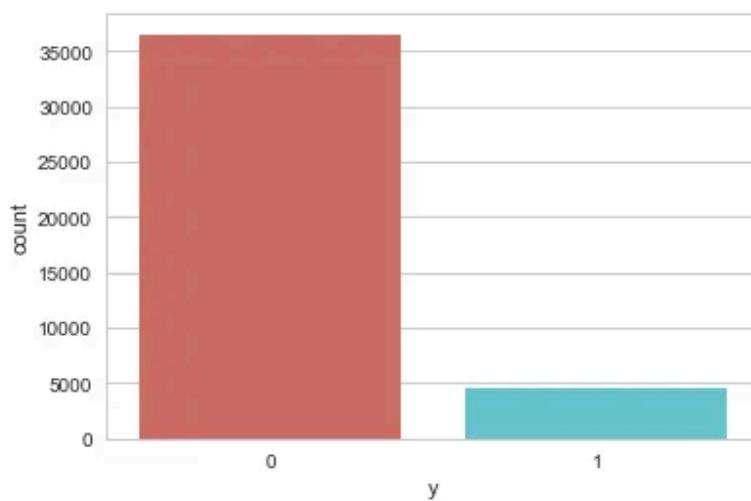


Figure 4

```
count_no_sub = len(data[data['y']==0])
count_sub = len(data[data['y']==1])
pct_of_no_sub = count_no_sub/(count_no_sub+count_sub)
print("percentage of no subscription is", pct_of_no_sub*100)
```

```
pct_of_sub = count_sub/(count_no_sub+count_sub)
print("percentage of subscription", pct_of_sub*100)
```

percentage of no subscription is 88.73458288821988

percentage of subscription 11.265417111780131

Our classes are imbalanced, and the ratio of no-subscription to subscription instances is 89:11. Before we go ahead to balance the classes, let's do some more exploration.

In [9]:	data.groupby('y').mean()
Out[9]:	
	age duration campaign pdays previous emp_var_rate cons_price_idx cons_conf_idx euribor3m nr_employed
	y
0	39.911185 220.844807 2.633085 984.113878 0.132374 0.248875 93.603757 -40.593097 3.811491 5176.166600
1	40.913147 553.191164 2.051724 792.035560 0.492672 -1.233448 93.354386 -39.789784 2.123135 5095.115991

Figure 5

Observations:

- The average age of customers who bought the term deposit is higher than that of the customers who didn't.
- The pdays (days since the customer was last contacted) is understandably lower for the customers who bought it. The lower the pdays, the better the memory of the last call and hence the better chances of a sale.
- Surprisingly, campaigns (number of contacts or calls made during the current campaign) are lower for customers who bought the term deposit.

We can calculate categorical means for other categorical variables such as education and marital status to get a more detailed sense of our data.

```
In [10]: data.groupby('job').mean()
```

```
Out[10]:
```

job	age	duration	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed	y
admin.	38.187296	254.312128	2.623489	954.319229	0.189023	0.015563	93.534054	-40.245433	3.550274	5164.125350	0.129726
blue-collar	39.555760	264.542360	2.558461	985.160363	0.122542	0.248995	93.656656	-41.375816	3.771996	5175.615150	0.068943
entrepreneur	41.723214	263.267857	2.535714	981.267170	0.138736	0.158723	93.605372	-41.283654	3.791120	5176.313530	0.085165
housemaid	45.500000	250.454717	2.639623	960.579245	0.137736	0.433396	93.676576	-39.495283	4.009645	5179.529623	0.100000
management	42.362859	257.058140	2.476060	962.647059	0.185021	-0.012688	93.522755	-40.489466	3.611316	5166.650513	0.112175
retired	62.027326	273.712209	2.476744	897.936047	0.327326	-0.698314	93.430786	-38.573081	2.770066	5122.262151	0.252326
self-employed	39.949331	264.142153	2.660802	976.621393	0.143561	0.094159	93.559982	-40.488107	3.689376	5170.674384	0.104856
services	37.926430	258.399085	2.587805	979.974049	0.154951	0.175359	93.634659	-41.290048	3.699187	5171.600126	0.081381
student	25.894857	283.683429	2.104000	840.217143	0.524571	-1.408000	93.331613	-40.187543	1.884224	5085.939086	0.314286
technician	38.507638	250.232241	2.577339	964.408127	0.153789	0.274566	93.561471	-39.927569	3.820401	5175.648391	0.108260
unemployed	39.733728	249.451677	2.564103	935.316568	0.199211	-0.111736	93.563781	-40.007594	3.466583	5157.156509	0.142012
unknown	45.563636	239.675758	2.648485	938.727273	0.154545	0.357879	93.718942	-38.797879	3.949033	5172.931818	0.112121

Figure 6

```
In [11]: data.groupby('marital').mean()
```

```
Out[11]:
```

marital	age	duration	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed	y
divorced	44.899393	253.790330	2.61340	968.639853	0.168690	0.163985	93.606563	-40.707069	3.715603	5170.878643	0.103209
married	42.307165	257.438623	2.57281	967.247673	0.155608	0.183625	93.597367	-40.270659	3.745832	5171.848772	0.101573
single	33.158714	261.524378	2.53380	949.909578	0.211359	-0.167989	93.517300	-40.918698	3.317447	5155.199265	0.140041
unknown	40.275000	312.725000	3.18750	937.100000	0.275000	-0.221250	93.471250	-40.820000	3.313038	5157.393750	0.150000

```
In [12]: data.groupby('education').mean()
```

```
Out[12]:
```

education	age	duration	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed	y
Basic	42.163910	263.043874	2.559498	974.877967	0.141053	0.191329	93.639933	-40.927595	3.729654	5172.014113	0.087029
high.school	37.998213	260.886810	2.568576	964.358382	0.185917	0.032937	93.584857	-40.940641	3.556157	5164.994735	0.108355
illiterate	48.500000	276.777778	2.277778	943.833333	0.111111	-0.133333	93.317333	-39.950000	3.516556	5171.777778	0.222222
professional.course	40.080107	252.533855	2.586115	960.765974	0.163075	0.173012	93.569864	-40.124108	3.710457	5170.155979	0.113485
university.degree	38.879191	253.223373	2.563527	951.807692	0.192390	-0.028090	93.493466	-39.975805	3.529663	5163.226298	0.137245
unknown	43.481225	262.390526	2.596187	942.830734	0.226459	0.059099	93.658615	-39.877816	3.571098	5159.549509	0.145003

Figure 7

Visualizations

```
%matplotlib inline
pd.crosstab(data.job,data.y).plot(kind='bar')
plt.title('Purchase Frequency for Job Title')
plt.xlabel('Job')
plt.ylabel('Frequency of Purchase')
plt.savefig('purchase_fre_job')
```

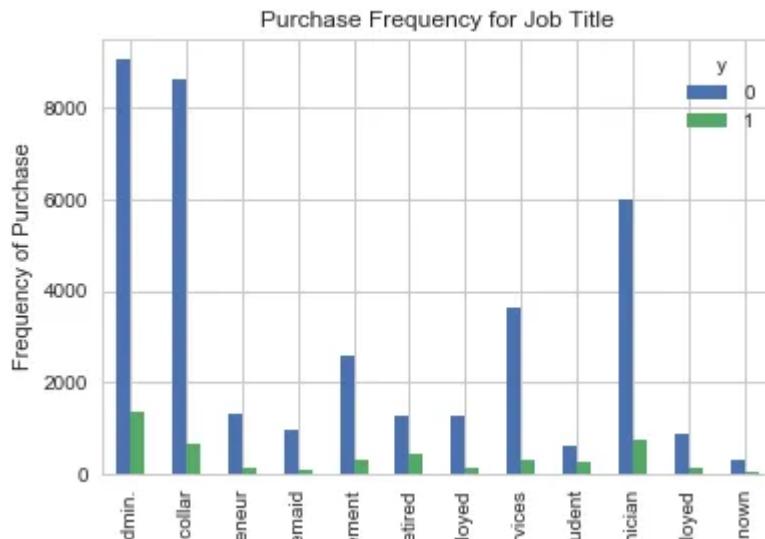


Figure 8

The frequency of purchase of the deposit depends a great deal on the job title. Thus, the job title can be a good predictor of the outcome variable.

```
table=pd.crosstab(data.marital,data.y)
table.div(table.sum(1).astype(float), axis=0).plot(kind='bar',
stacked=True)
plt.title('Stacked Bar Chart of Marital Status vs Purchase')
plt.xlabel('Marital Status')
plt.ylabel('Proportion of Customers')
plt.savefig('mariral_vs_pur_stack')
```

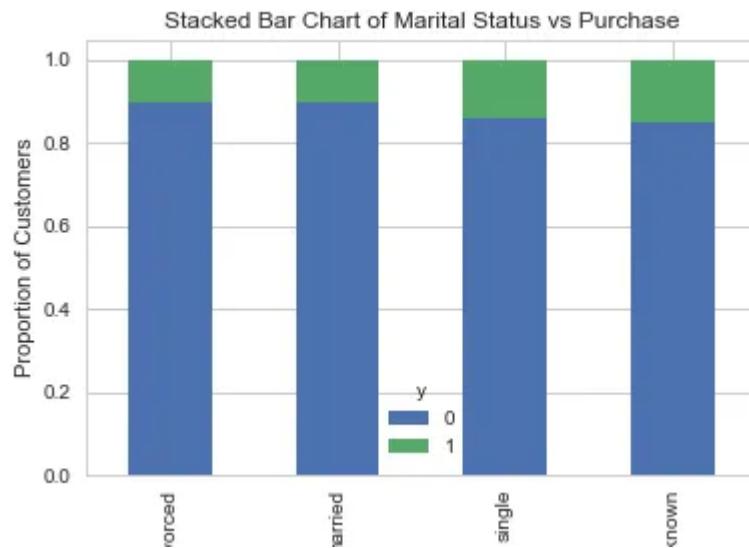


Figure 9

The marital status does not seem a strong predictor for the outcome variable.

```
table=pd.crosstab(data.education,data.y)
table.div(table.sum(1).astype(float), axis=0).plot(kind='bar',
stacked=True)
plt.title('Stacked Bar Chart of Education vs Purchase')
plt.xlabel('Education')
plt.ylabel('Proportion of Customers')
plt.savefig('edu_vs_pur_stack')
```

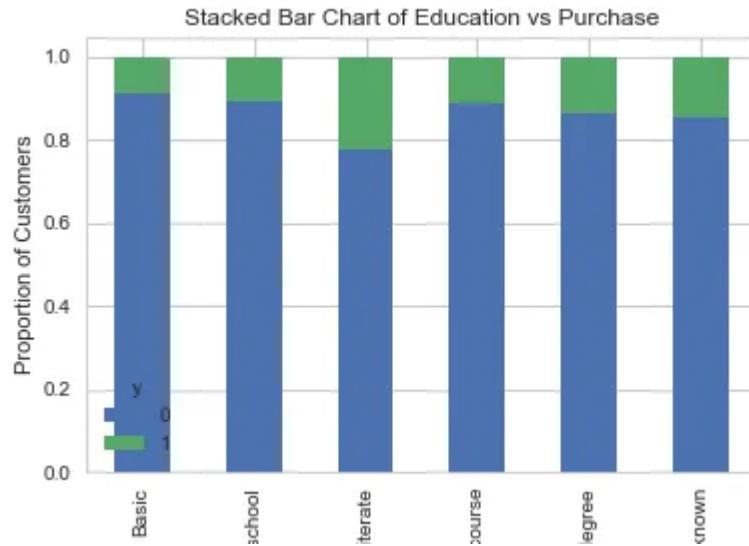


Figure 10

Education seems a good predictor of the outcome variable.

```
pd.crosstab(data.day_of_week,data.y).plot(kind='bar')
plt.title('Purchase Frequency for Day of Week')
plt.xlabel('Day of Week')
plt.ylabel('Frequency of Purchase')
plt.savefig('pur_dayofweek_bar')
```

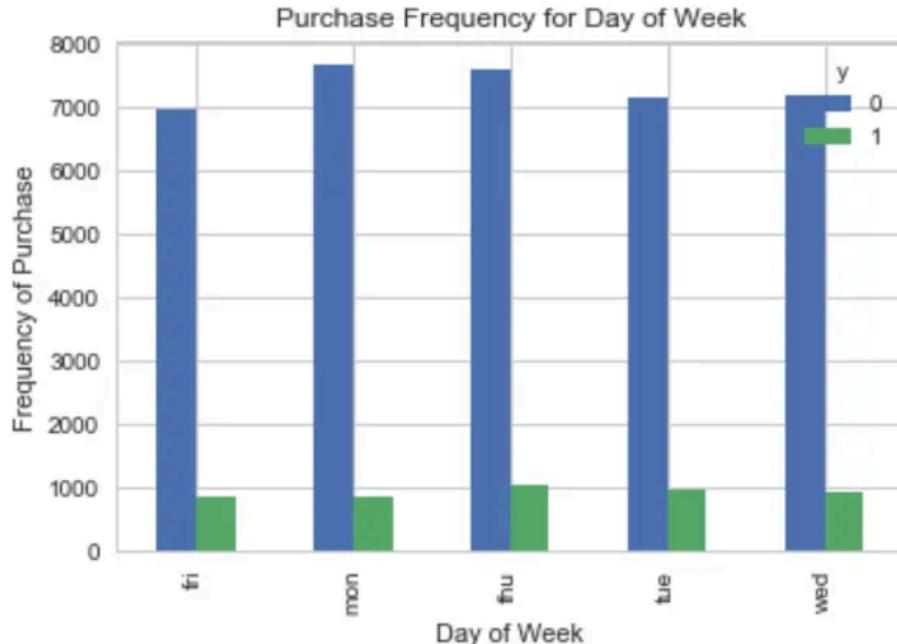


Figure 11

Day of week may not be a good predictor of the outcome.

```
pd.crosstab(data.month,data.y).plot(kind='bar')
plt.title('Purchase Frequency for Month')
plt.xlabel('Month')
plt.ylabel('Frequency of Purchase')
plt.savefig('pur_fre_month_bar')
```

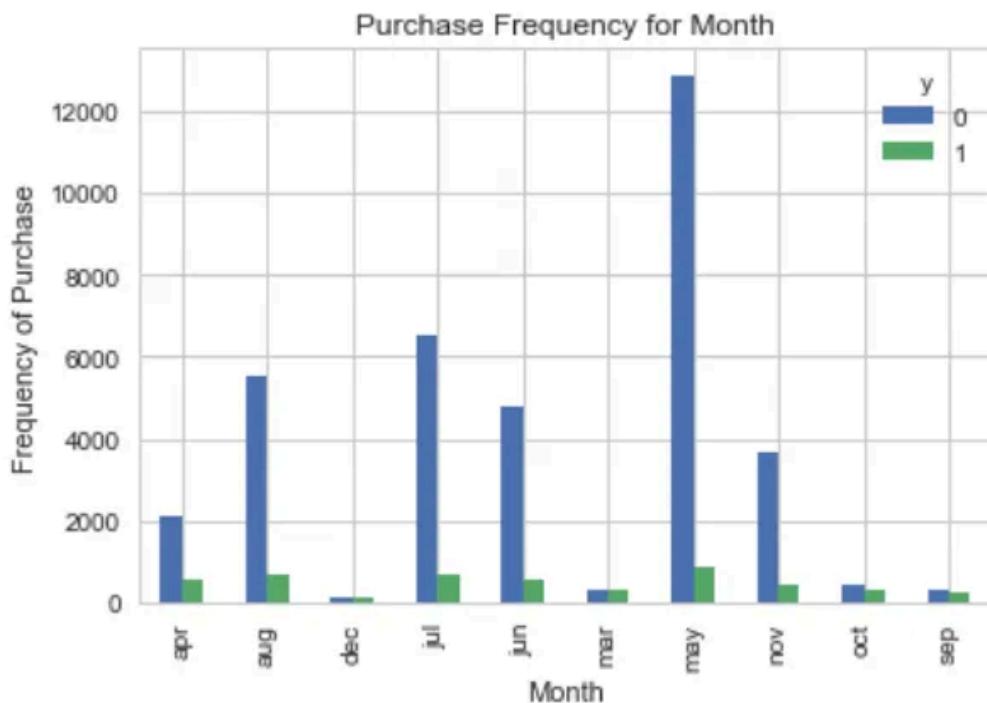


Figure 12

Month might be a good predictor of the outcome variable.

```
data.age.hist()  
plt.title('Histogram of Age')  
plt.xlabel('Age')  
plt.ylabel('Frequency')  
plt.savefig('hist_age')
```

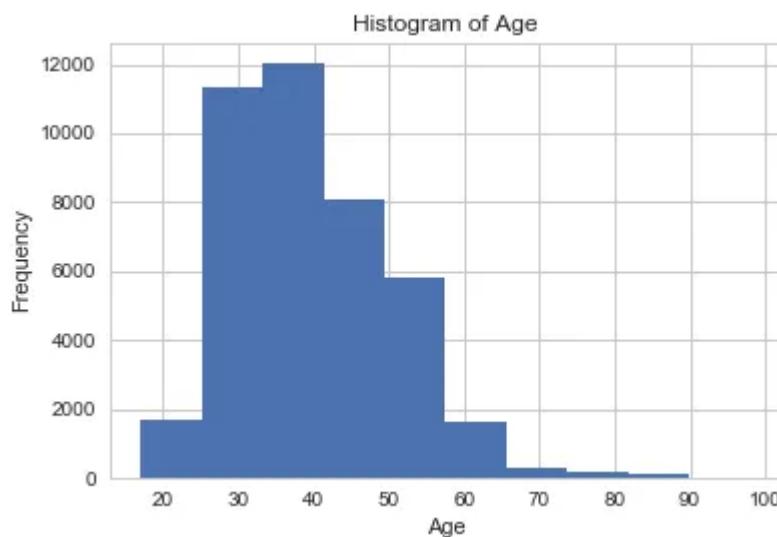


Figure 13

Most of the customers of the bank in this dataset are in the age range of 30–40.

```
pd.crosstab(data.poutcome,data.y).plot(kind='bar')
plt.title('Purchase Frequency for Poutcome')
plt.xlabel('Poutcome')
plt.ylabel('Frequency of Purchase')
plt.savefig('pur_fre_pout_bar')
```

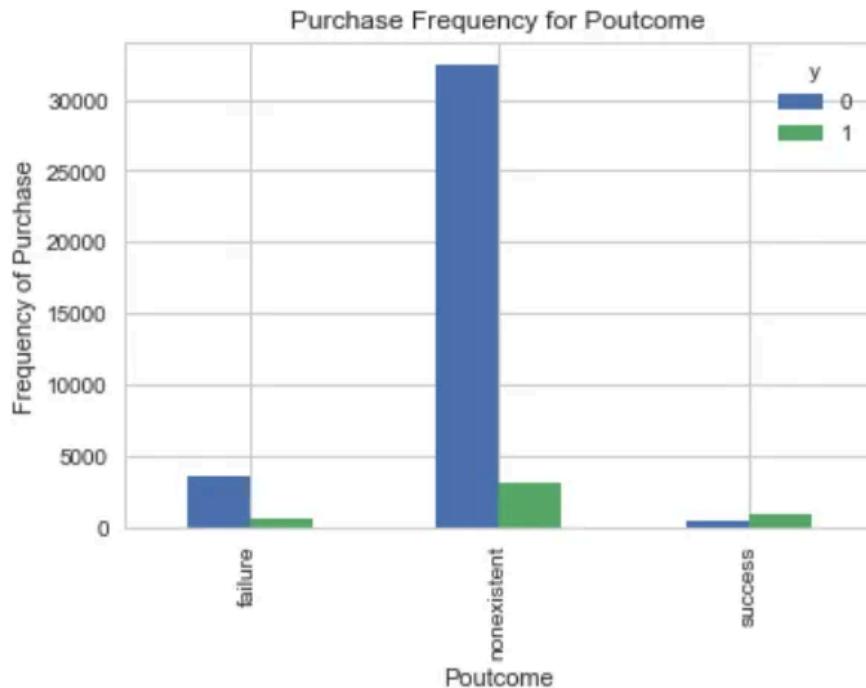


Figure 14

Poutcome seems to be a good predictor of the outcome variable.

Create dummy variables

That is variables with only two values, zero and one.

```
cat_vars=
['job','marital','education','default','housing','loan','contact','month','day_of_week','poutcome']
for var in cat_vars:
    cat_list='var'+'_'+var
    cat_list = pd.get_dummies(data[var], prefix=var)
    data1=data.join(cat_list)
    data=data1

cat_vars=
['job','marital','education','default','housing','loan','contact','month','day_of_week','poutcome']
```

```
onth', 'day_of_week', 'poutcome']
data_vars=data.columns.values.tolist()
to_keep=[i for i in data_vars if i not in cat_vars]
```

Our final data columns will be:

```
data_final=data[to_keep]
data_final.columns.values
```

```
array(['age', 'duration', 'campaign', 'pdays', 'previous', 'emp_var_rate',
       'cons_price_idx', 'cons_conf_idx', 'euribor3m', 'nr_employed', 'y',
       'job_admin.', 'job_blue-collar', 'job_entrepreneur',
       'job_housemaid', 'job_management', 'job_retired',
       'job_self-employed', 'job_services', 'job_student',
       'job_technician', 'job_unemployed', 'job_unknown',
       'marital_divorced', 'marital_married', 'marital_single',
       'marital_unknown', 'education_Basic', 'education_high.school',
       'education_illiterate', 'education_professional.course',
       'education_university.degree', 'education_unknown', 'default_no',
       'default_unknown', 'default_yes', 'housing_no', 'housing_unknown',
       'housing_yes', 'loan_no', 'loan_unknown', 'loan_yes',
       'contact_cellular', 'contact_telephone', 'month_apr', 'month_aug',
       'month_dec', 'month_jul', 'month_jun', 'month_mar', 'month_may',
       'month_nov', 'month_oct', 'month_sep', 'day_of_week_fri',
       'day_of_week_mon', 'day_of_week_thu', 'day_of_week_tue',
       'day_of_week_wed', 'poutcome_failure', 'poutcome_nonexistent',
       'poutcome_success'], dtype=object)
```

Figure 15

Over-sampling using SMOTE

With our training data created, I'll up-sample the no-subscription using the [SMOTE algorithm](#)(Synthetic Minority Oversampling Technique). At a high level, SMOTE:

1. Works by creating synthetic samples from the minor class (no-subscription) instead of creating copies.
2. Randomly choosing one of the k-nearest-neighbors and using it to create a similar, but randomly tweaked, new observations.

We are going to implement [SMOTE in Python](#).

```
X = data_final.loc[:, data_final.columns != 'y']
y = data_final.loc[:, data_final.columns == 'y']

from imblearn.over_sampling import SMOTE
```

```

os = SMOTE(random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=0)
columns = X_train.columns

os_data_X,os_data_y=os.fit_sample(X_train, y_train)
os_data_X = pd.DataFrame(data=os_data_X,columns=columns )
os_data_y= pd.DataFrame(data=os_data_y,columns=['y'])
# we can Check the numbers of our data
print("length of oversampled data is ",len(os_data_X))
print("Number of no subscription in oversampled data",len(os_data_y[os_data_y['y']==0]))
print("Number of subscription",len(os_data_y[os_data_y['y']==1]))
print("Proportion of no subscription data in oversampled data is ",len(os_data_y[os_data_y['y']==0])/len(os_data_X))
print("Proportion of subscription data in oversampled data is ",len(os_data_y[os_data_y['y']==1])/len(os_data_X))

```

```

length of oversampled data is  51134
Number of no subscription in oversampled data 25567
Number of subscription 25567
Proportion of no subscription data in oversampled data is  0.5
Proportion of subscription data in oversampled data is  0.5

```

Figure 16

Now we have a perfect balanced data! You may have noticed that I over-sampled only on the training data, because by oversampling only on the training data, none of the information in the test data is being used to create synthetic observations, therefore, no information will bleed from test data into the model training.

Recursive Feature Elimination

Recursive Feature Elimination (RFE) is based on the idea to repeatedly construct a model and choose either the best or worst performing feature, setting the feature aside and then repeating the process with the rest of the features. This process is applied until all features in the dataset are exhausted. The goal of RFE is to select features by recursively considering smaller and smaller sets of features.

```

data_final_vars=data_final.columns.values.tolist()
y=['y']
X=[i for i in data_final_vars if i not in y]

from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()

```

```
rfe = RFE(logreg, 20)
rfe = rfe.fit(os_data_X, os_data_y.values.ravel())
print(rfe.support_)
print(rfe.ranking_)

[False False False False False False False False True False False True
 False True False False False False False False False False False False
 False True False False False False True False False False True True
 False False False False False False False True True True True True
 True True True True True False False False False False False False False
 True False True]

[39 38 26 42 9 12 24 36 1 35 8 1 7 1 5 32 2 4 31 3 6 10 23 21
 17 1 14 18 15 22 1 20 16 19 1 1 41 28 44 37 33 43 34 1 1 1 1 1
 1 1 1 1 1 29 30 11 27 40 25 1 13 1]
```

Figure 16

The RFE has helped us select the following features: “euribor3m”, “job_blue-collar”, “job_housemaid”, “marital_unknown”, “education_illiterate”, “default_no”, “default_unknown”, “contact_cellular”, “contact_telephone”, “month_apr”, “month_aug”, “month_dec”, “month_jul”, “month_jun”, “month_mar”, “month_may”, “month_nov”, “month_oct”, “poutcome_failure”, “poutcome_success”.

```
cols=['euribor3m', 'job_blue-collar', 'job_housemaid',
'marital_unknown', 'education_illiterate', 'default_no',
'default_unknown',
'contact_cellular', 'contact_telephone', 'month_apr',
'month_aug', 'month_dec', 'month_jul', 'month_jun', 'month_mar',
'month_may', 'month_nov', 'month_oct', "poutcome_failure",
"poutcome_success"]
X=os_data_X[cols]
y=os_data_y['y']
```

Implementing the model

```
import statsmodels.api as sm
logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```
Warning: Maximum number of iterations has been exceeded.
```

```
Current function value: 0.545891
```

```
Iterations: 35
```

```
Results: Logit
```

Model:	Logit	No. Iterations:	35.0000			
Dependent Variable:	y	Pseudo R-squared:	0.212			
Date:	2018-09-10 12:16	AIC:	55867.1778			
No. Observations:	51134	BIC:	56044.0219			
Df Model:	19	Log-Likelihood:	-27914.			
Df Residuals:	51114	LL-Null:	-35443.			
Converged:	0.0000	Scale:	1.0000			
	Coef.	Std.Err.	z	P> z	[0.025	0.975]
euribor3m	-0.4634	0.0091	-50.9471	0.0000	-0.4813	-0.4456
job_blue-collar	-0.1736	0.0283	-6.1230	0.0000	-0.2291	-0.1180
job_housemaid	-0.3260	0.0778	-4.1912	0.0000	-0.4784	-0.1735
marital_unknown	0.7454	0.2253	3.3082	0.0009	0.3038	1.1870
education_illiterate	1.3156	0.4373	3.0084	0.0026	0.4585	2.1727
default_no	16.1521	5414.0744	0.0030	0.9976	-10595.2387	10627.5429
default_unknown	15.8945	5414.0744	0.0029	0.9977	-10595.4963	10627.2853
contact_cellular	-13.9393	5414.0744	-0.0026	0.9979	-10625.3302	10597.4515
contact_telephone	-14.0065	5414.0744	-0.0026	0.9979	-10625.3973	10597.3843
month_apr	-0.8356	0.0913	-9.1490	0.0000	-1.0145	-0.6566
month_aug	-0.6882	0.0929	-7.4053	0.0000	-0.8703	-0.5061
month_dec	-0.4233	0.1655	-2.5579	0.0105	-0.7477	-0.0990
month_jul	-0.4056	0.0935	-4.3391	0.0000	-0.5889	-0.2224
month_jun	-0.4817	0.0917	-5.2550	0.0000	-0.6614	-0.3021
month_mar	0.6638	0.1229	5.3989	0.0000	0.4228	0.9047
month_may	-1.4752	0.0874	-16.8815	0.0000	-1.6465	-1.3039
month_nov	-0.8298	0.0942	-8.8085	0.0000	-1.0144	-0.6451
month_oct	0.5065	0.1175	4.3111	0.0000	0.2762	0.7367
poutcome_failure	-0.5000	0.0363	-13.7706	0.0000	-0.5711	-0.4288
poutcome_success	1.5788	0.0618	25.5313	0.0000	1.4576	1.7000

Figure 17

The p-values for most of the variables are smaller than 0.05, except four variables, therefore, we will remove them.

```
cols=['euribor3m', 'job_blue-collar', 'job_housemaid',
'marital_unknown', 'education_illiterate',
'month_apr', 'month_aug', 'month_dec', 'month_jul',
'month_jun', 'month_mar',
'month_may', 'month_nov', 'month_oct', "poutcome_failure",
"poutcome_success"]
X=os_data_X[cols]
y=os_data_y['y']

logit_model=sm.Logit(y,X)
result=logit_model.fit()
print(result.summary2())
```

```

Optimization terminated successfully.
Current function value: 0.555865
Iterations 7
Results: Logit
=====
Model:          Logit           No. Iterations: 7.0000
Dependent Variable: y            Pseudo R-squared: 0.198
Date:          2018-09-10 12:38  AIC:      56879.2425
No. Observations: 51134        BIC:      57020.7178
Df Model:       15             Log-Likelihood: -28424.
Df Residuals:   51118         LL-Null:   -35443.
Converged:      1.0000        Scale:     1.0000
-----
              Coef.  Std.Err.    z  P>|z|  [0.025  0.975]
-----
euribor3m      -0.4488  0.0074 -60.6837 0.0000 -0.4633 -0.4343
job_blue-collar -0.2060  0.0278 -7.4032 0.0000 -0.2605 -0.1515
job_housemaid   -0.2784  0.0762 -3.6519 0.0003 -0.4278 -0.1290
marital_unknown  0.7619  0.2244  3.3956 0.0007  0.3221  1.2017
education_illiterate 1.3080  0.4346  3.0096 0.0026  0.4562  2.1598
month_apr        1.2863  0.0380 33.8180 0.0000  1.2118  1.3609
month_aug        1.3959  0.0411 33.9688 0.0000  1.3153  1.4764
month_dec        1.8084  0.1441 12.5483 0.0000  1.5259  2.0908
month_jul        1.6747  0.0424 39.5076 0.0000  1.5916  1.7578
month_jun        1.5574  0.0408 38.1351 0.0000  1.4773  1.6374
month_mar        2.8215  0.0908 31.0891 0.0000  2.6437  2.9994
month_may        0.5848  0.0304 19.2166 0.0000  0.5251  0.6444
month_nov        1.2725  0.0445 28.5720 0.0000  1.1852  1.3598
month_oct        2.7279  0.0816 33.4350 0.0000  2.5680  2.8878
poutcome_failure -0.2797  0.0351 -7.9753 0.0000 -0.3485 -0.2110
poutcome_success  1.9617  0.0602 32.5939 0.0000  1.8438  2.0797
=====
```

Figure 18

Logistic Regression Model Fitting

```

from sklearn.linear_model import LogisticRegression
from sklearn import metrics

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=0)
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
verbose=0, warm_start=False)
```

Figure 19

Predicting the test set results and calculating the accuracy

```
y_pred = logreg.predict(X_test)
print('Accuracy of logistic regression classifier on test set:
{:.2f}'.format(logreg.score(X_test, y_test)))
```

Accuracy of logistic regression classifier on test set: 0.74

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
```

`[[6124 1542]`

`[2505 5170]]`

The result is telling us that we have $6124+5170$ correct predictions and $2505+1542$ incorrect predictions.

Compute precision, recall, F-measure and support

To quote from [Scikit Learn](#):

The precision is the ratio $tp / (tp + fp)$ where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier to not label a sample as positive if it is negative.

The recall is the ratio $tp / (tp + fn)$ where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples.

The F-beta score can be interpreted as a weighted harmonic mean of the precision and recall, where an F-beta score reaches its best value at 1 and worst score at 0.

The F-beta score weights the recall more than the precision by a factor of beta. beta = 1.0 means recall and precision are equally important.

The support is the number of occurrences of each class in y_{test} .

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.71	0.80	0.75	7666
1	0.77	0.67	0.72	7675
avg / total	0.74	0.74	0.74	15341

Figure 20

Interpretation: Of the entire test set, 74% of the promoted term deposit were the term deposit that the customers liked. Of the entire test set, 74% of the customer's preferred term deposits that were promoted.

ROC Curve

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test, logreg.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test,
logreg.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % 
logit_roc_auc)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```

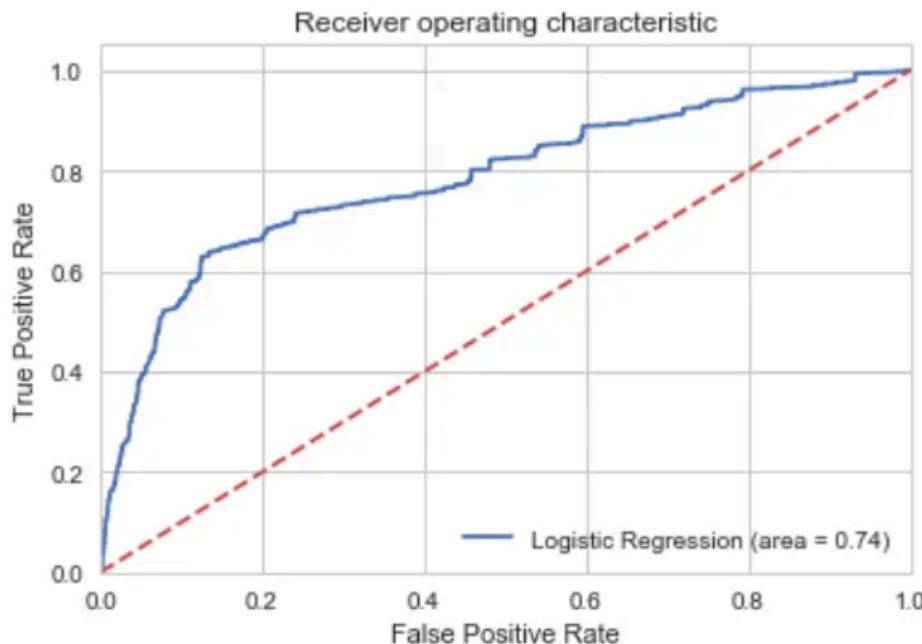


Figure 21

The receiver operating characteristic (ROC) curve is another common tool used with binary classifiers. The dotted line represents the ROC curve of a purely random classifier; a good classifier stays as far away from that line as possible (toward the top-left corner).

The Jupyter notebook used to make this post is available [here](#). I would be pleased to receive feedback or questions on any of the above.

Reference: [Learning Predictive Analytics with Python book](#)

Machine Learning

Data Science

Python

Logistic Regression

Classification



Follow

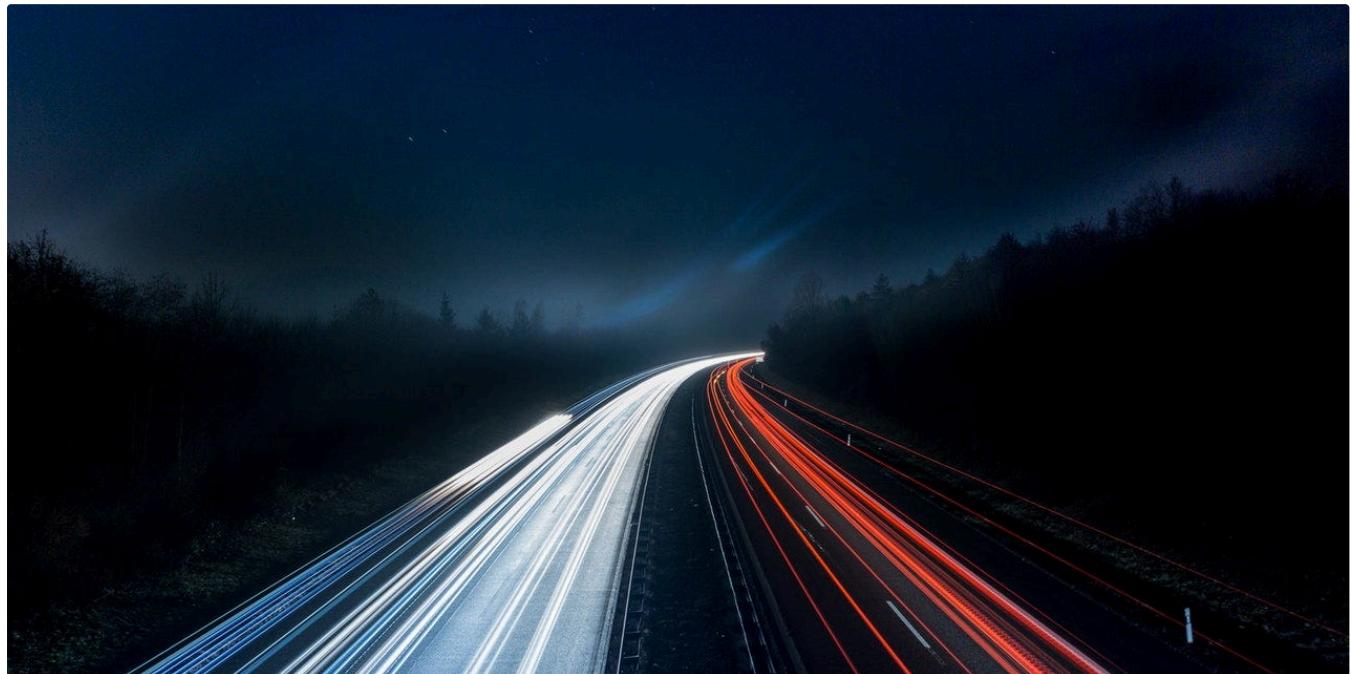
Written by Susan Li

28K Followers · Writer for Towards Data Science

Changing the world, one post at a time. Sr Data Scientist, Toronto Canada.

<https://www.linkedin.com/in/susanli/>

More from Susan Li and Towards Data Science



Susan Li in Towards Data Science

An End-to-End Project on Time Series Analysis and Forecasting with Python

Time series analysis comprises methods for analyzing time series data in order to extract meaningful statistics and other characteristics...

Jul 9, 2018 9.6K 93



dog run	dog run
cat dog	cat dog

Turned into token ids:

Original	Translated
1 4 2	1 4 2
1 3 4	1 3 4

For this whole example we'll use a single training batch of size 2.

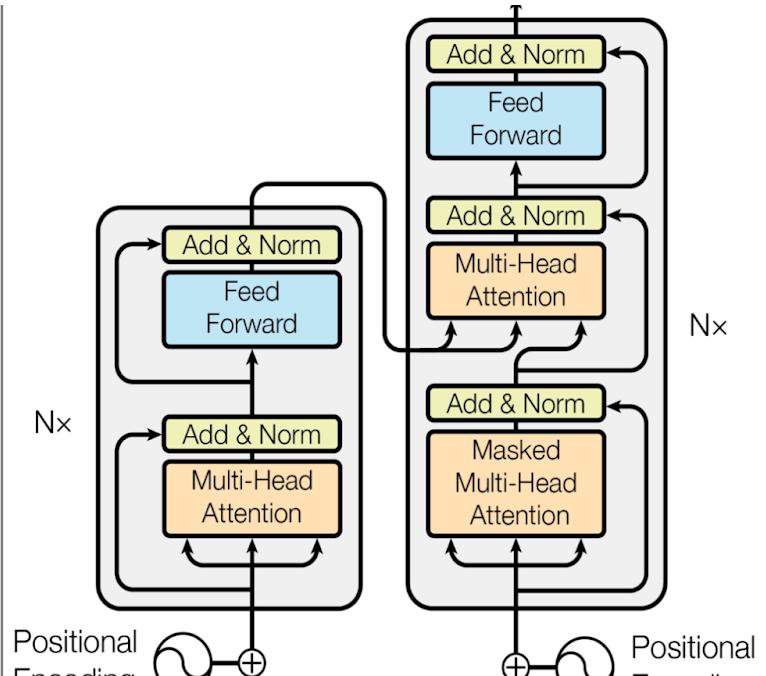
As shown on right, we feed these **inputs**:

```
[[1, 4, 2],  
 [1, 3, 4]]
```

And we feed these “**outputs**”:

```
[[1, 4],  
 [1, 3]]
```

Once trained, the model will translate text. It



Eric Silberstein in Towards Data Science

Tracing the Transformer in Diagrams

What exactly do you put in, what exactly do you get out, and how do you generate text with it?

17h ago 129 1





 Sergey Kotlov in Towards Data Science

Adopting Spark Connect

How we use a shared Spark server to make our Spark infrastructure more efficient

17h ago  11



 Susan Li in Towards Data Science

Multi-Class Text Classification Model Comparison and Selection

Natural Language Processing, word2vec, Support Vector Machine, bag-of-words, deep learning

Sep 25, 2018 3.5K 37[See all from Susan Li](#)[See all from Towards Data Science](#)

Recommended from Medium



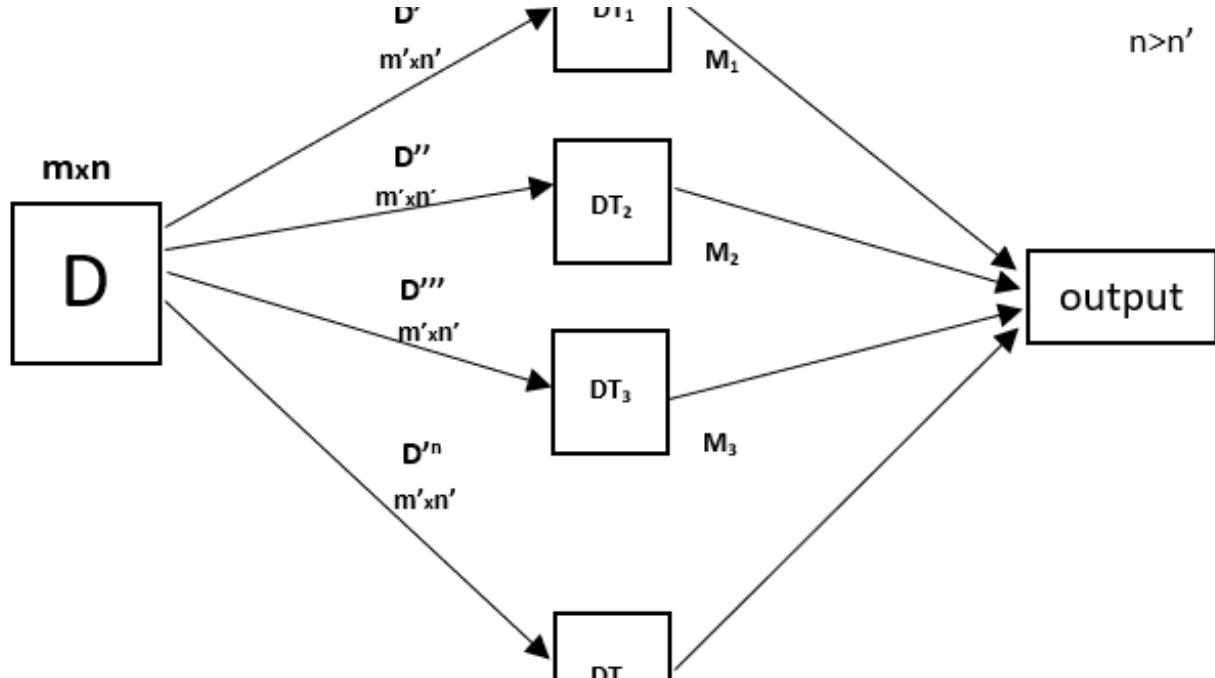
Vikash Singh

Regression vs. Classification: the Davengers Style

Beginner friendly introduction in a fun-friendly manner!

Oct 24 37





AI Wrld

Random Forest: The Ultimate Guide to Regression and Classification

Random Forest is one of the most powerful and versatile machine learning algorithms, frequently used for both classification and regression...

Sep 10 5



Lists



Predictive Modeling w/ Python

20 stories · 1641 saves



Practical Guides to Machine Learning

10 stories · 2005 saves



Coding & Development

11 stories · 890 saves



Natural Language Processing

1798 stories · 1408 saves

FEATURE SCALING IN LINEAR REGRESSION - PYTHON

- FEATURE SCALING (SCRATCH + SCIKIT LEARN)
- WHAT IS FEATURE SCALING?
- WHEN SHOULD WE USE FEATURE SCALING?
- WHY DO WE NEED FEATURE SCALING IN LINEAR OR MULTIPLE LINEAR REGRESSION

Let's
Code
More.



Areeba Seher | Lets Code More in Artificial Intelligence in Plain English

Feature Scaling in Linear Regression in Python

In this article, you will learn What is feature scaling? When should we use feature scaling? Do We need to do feature scaling for simple...

◆ May 31



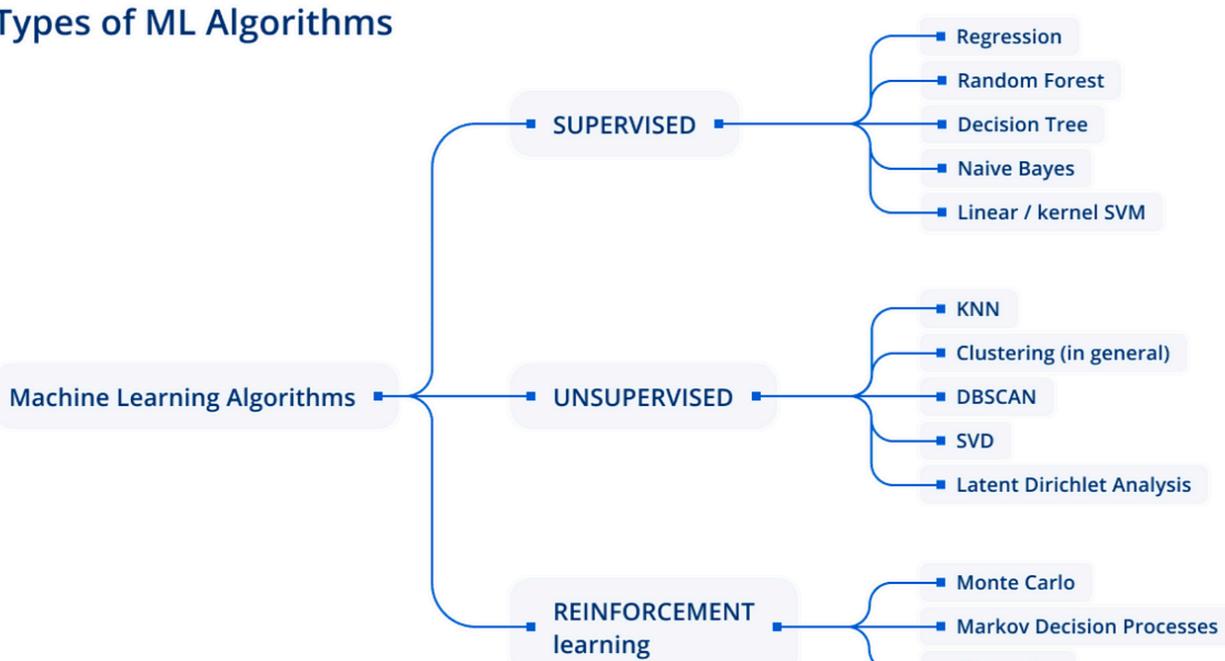
```

1111 // ngSwitchElement, element, attr, ngModel
1112 // selected = attr.ngSwitch || attr.on,
1113 // previousElements = [],
1114 // currentElements = [],
1115 // previousElements = [],
1116 // currentElements = [],
1117 // selectedScopes = []
1118
1119 // scope.$watch(expr, function ngSwitchWatchAction(value) {
1120 //   var i, ii;
1121 //   for (i = 0, ii = previousElements.length; i < ii; ++i) {
1122 //     previousElements[i].remove();
1123 //   }
1124 //   previousElements.length = 0;
1125
1126 //   for (i = 0, ii = selectedScopes.length; i < ii; ++i) {
1127 //     var selected = selectedElements[i];
1128 //     selectedScopes[i].$destroy();
1129 //     previousElements[i] = selected;
1130 //     $animate.leave(selected, function() {
1131 //       previousElements.splice(i, 1);
1132 //     });
1133 //   }
1134
1135 //   selectedElements.length = 0;
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2388
2389
2389
2390
2391
2392
2393
2394
2395
2396
2397
2397
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2448
2449
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2488
2489
2489
2490
2491
2492
2493
2494
2495
2496
2497
2497
2498
2499
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2548
2549
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2588
2589
2589
2590
2591
2592
2593
2594
2595
2596
2597
2597
2598
2599
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2638
2639
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2648
2649
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2688
2689
2689
2690
2691
2692
2693
2694
2695
2696
2697
2697
2698
2699
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2738
2739
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2748
2749
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2788
2789
2789
2790
2791
2792
2793
2794
2795
2796
2797
2797
2798
2799
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2838
2839
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2848
2849
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2888
2889
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2898
2899
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2938
2939
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2948
2949
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2988
2989
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2998
2999
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009

```



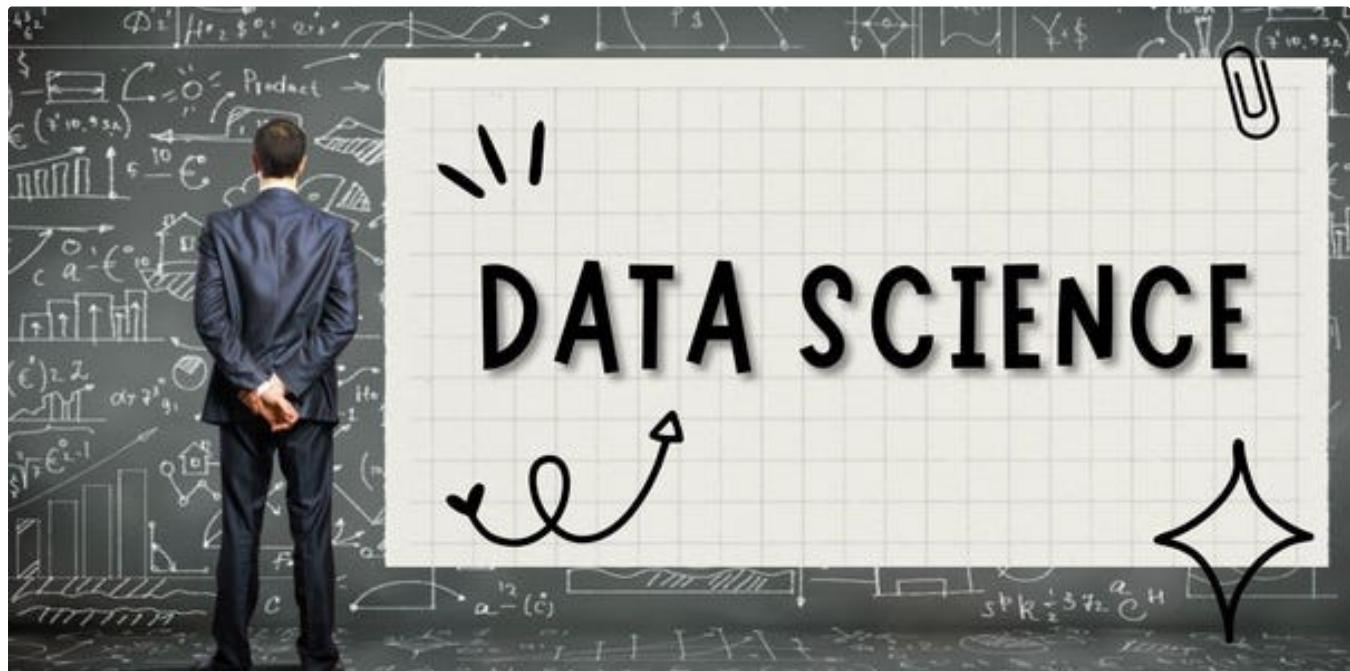
Types of ML Algorithms



 John Vastola

10 Must-Know Machine Learning Algorithms for Data Scientists

Machine learning is the science of getting computers to act without being explicitly programmed.”—Andrew Ng



 Emmanuel Ikogho

Data Science is dying; here's why

Why 85% of data science projects fail

◆ Sep 3 ⌘ 1.92K 🔞 90



See more recommendations