



Get unlimited access

Open in app



Chirumamilla Nandavardhan

Follow

Oct 10, 2019 · 7 min read · Listen



Save



Divorce Prediction using Machine Learning

In this blog-post, I will go through the whole process of creating a machine learning model on the Divorce Predictors dataset. It provides a prediction on whether a couple will get divorced or not based on some input features.

Dataset Description

In this dataset, we have 170 samples with 54 input features, which are :



1. When one of our apologies apologizes when our discussions go in a bad direction, the issue does not extend.
2. I know we can ignore our differences, even if things get hard sometimes.
3. When we need it, we can take our discussions with my wife from the beginning and correct it.
4. When I argue with my wife, it will eventually work for me to contact him.
5. The time I spent with my wife is special for us.
6. We don't have time at home as partners.
7. We are like two strangers who share the same environment at home rather than family.
8. I enjoy our holidays with my wife.
9. I enjoy traveling with my wife.
10. My wife and most of our goals are common.
11. I think that one day in the future, when I look back, I see that my wife and I are in harmony with each other.
12. My wife and I have similar values in terms of personal freedom.





Get unlimited access

Open in app

17. We share the same views with my wife about being happy in your life
18. My wife and I have similar ideas about how marriage should be
19. My wife and I have similar ideas about how roles should be in marriage
20. My wife and I have similar values in trust
21. I know exactly what my wife likes.
22. I know how my wife wants to be taken care of when she's sick.
23. I know my wife's favorite food.
24. I can tell you what kind of stress my wife is facing in her life.
25. I have knowledge of my wife's inner world.
26. I know my wife's basic concerns.
27. I know what my wife's current sources of stress are.
28. I know my wife's hopes and wishes.
29. I know my wife very well.
30. I know my wife's friends and their social relationships.
31. I feel aggressive when I argue.  24 |  1 | ...
32. When discussing with my wife, I usually use expressions such as "you always" or "you never" .
33. I can use negative statements about my wife's personality during our discussions.
34. I can use offensive expressions during our discussions.
35. I can insult our discussions.
36. I can be humiliating when we argue.
37. My argument with my wife is not calm.
38. I hate my wife's way of bringing it up.
39. Fights often occur suddenly.
40. We're just starting a fight before I know what's going on.
41. When I talk to my wife about something, my calm suddenly breaks.
42. When I argue with my wife, it only snaps in and I don't say a word.
43. I'm mostly thirsty to calm the environment a little bit.
44. Sometimes I think it's good for me to leave home for a while.
45. I'd rather stay silent than argue with my wife.
46. Even if I'm right in the argument, I'm thirsty not to upset the other side.
47. When I argue with my wife, I remain silent because I am afraid of not being able





Get unlimited access

Open in app

51. I'm not the one who's wrong about problems at home.
52. I wouldn't hesitate to tell her about my wife's inadequacy.
53. When I discuss it, I remind her of my wife's inadequate issues.
54. I'm not afraid to tell her about my wife's incompetence.

The above dataset can be downloaded from

<http://archive.ics.uci.edu/ml/datasets/Divorce+Predictors+data+set>

Getting the Data

```
import pandas as pd
from sklearn import preprocessing
from scipy.io import arff

df = pd.read_csv("divorce.csv", sep=";")

df
```

	Atr1	Atr2	Atr3	Atr4	Atr5	Atr6	Atr7	Atr8	Atr9	Atr10	Atr11	Atr12	Atr13	Atr14	Atr15	Atr16	Atr17	Atr18	Atr19	Atr20	Atr21	Atr22	Atr23	Atr24	Atr25	Atr26	Atr27
0	2	2	4	1	0	0	0	0	0	0	1	0	1	1	0	1	0	0	0	1	0	0	0	0	0	0	0
1	4	4	4	4	4	0	0	4	4	4	4	3	4	0	4	4	4	4	3	2	1	1	0	2	2	1	2
2	2	2	2	2	1	3	2	1	1	2	3	4	2	3	3	3	3	3	3	2	1	0	1	2	2	2	2
3	3	2	3	2	3	3	3	3	3	3	4	3	3	4	3	3	3	3	3	4	1	1	1	1	2	1	1
4	2	2	1	1	1	0	0	0	0	0	1	0	1	1	1	1	1	1	2	1	1	0	0	0	0	2	1
5	0	0	1	0	0	2	0	0	0	1	0	2	1	0	2	0	2	1	0	1	0	0	0	0	2	2	0
6	3	3	3	2	1	3	4	3	2	2	2	2	2	3	2	3	3	3	3	2	3	3	3	3	2	3	3
7	2	1	2	2	2	1	0	3	3	2	4	3	2	3	4	3	2	3	2	1	2	1	1	2	3	3	2
8	2	2	1	0	0	4	1	3	3	3	3	3	3	3	3	3	3	3	3	3	2	2	2	3	2	3	2
9	1	1	1	1	1	2	0	2	2	2	3	0	0	2	1	0	1	2	1	0	0	0	0	1	1	1	1
10	4	4	4	3	4	0	0	4	4	3	4	4	4	4	4	3	4	4	4	4	4	3	4	4	4	4	4
11	4	4	4	3	4	0	0	4	4	3	4	4	4	4	4	3	4	4	4	4	4	3	4	4	4	4	4
12	3	4	3	4	3	0	1	4	3	4	3	4	3	4	3	4	3	4	3	4	3	4	3	4	3	4	3
13	3	4	3	4	3	0	1	4	3	4	3	4	3	4	3	4	3	4	3	4	3	4	3	4	3	4	3
14	3	4	3	4	3	0	1	4	3	4	3	4	3	4	3	4	3	4	3	4	3	4	3	4	3	4	3
15	4	4	3	2	4	0	0	4	3	2	4	4	4	4	3	2	4	4	4	4	3	2	4	4	4	4	3
16	4	4	3	2	4	0	0	4	3	2	4	4	4	4	3	2	4	4	4	4	3	2	4	4	4	4	3
17	4	4	4	3	4	0	0	4	4	3	4	4	4	4	4	3	4	4	4	4	4	3	4	4	4	4	4

Preview of the Dataset

Preprocessing

Check for null values

```
total = df.isnull().sum() # Check for null values (according to blog)
```





Get unlimited access

Open in app

```
from sklearn import preprocessing

x = df.values #returns a numpy array
print(x)
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
df = pd.DataFrame(x_scaled)
df
```

Feature Selection

We can select the important features and observe how much a feature is effecting our prediction. For this, we use the pearson correlation matrix.

```
import seaborn as sns

# load the R package ISLR

# load the Auto dataset
auto_df = df

# calculate the correlation matrix
corr = auto_df.corr()

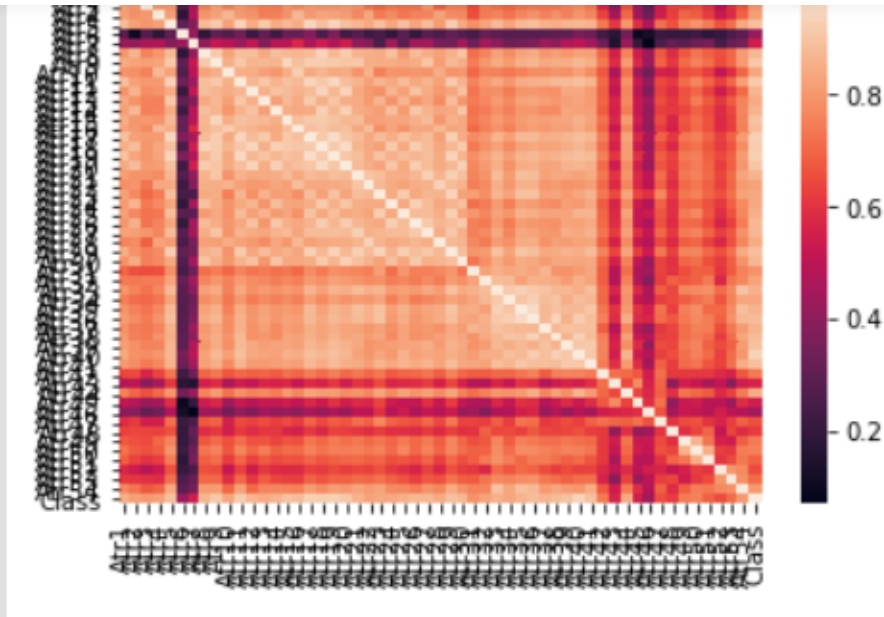
# plot the heatmap
sns.heatmap(corr,
            xticklabels=corr.columns,
            yticklabels=corr.columns)
```





Get unlimited access

Open in app



Graph Plotted for the Pearson Matrix

corr

	Atr1	Atr2	Atr3	Atr4	Atr5	Atr6	Atr7	Atr8	Atr9	Atr10	Atr11	Atr12	Atr13	Atr14	Atr15	Atr16
Atr1	1.000000	0.819066	0.832508	0.825066	0.881272	0.287140	0.427989	0.802357	0.845916	0.790183	0.892253	0.794307	0.842996	0.817099	0.848754	0.831822
Atr2	0.819066	1.000000	0.805876	0.791313	0.819360	0.102843	0.417616	0.864284	0.827711	0.782286	0.823380	0.862835	0.791073	0.875800	0.801316	0.806497
Atr3	0.832508	0.805876	1.000000	0.806709	0.800774	0.263032	0.464071	0.757264	0.816653	0.753017	0.805915	0.780258	0.758969	0.750602	0.806909	0.775528
Atr4	0.825066	0.791313	0.806709	1.000000	0.818472	0.185963	0.474806	0.798347	0.829053	0.873636	0.808533	0.793992	0.751623	0.757000	0.794184	0.878416
Atr5	0.881272	0.819360	0.800774	0.818472	1.000000	0.297834	0.381378	0.877584	0.916327	0.823659	0.936955	0.846513	0.915033	0.845576	0.879461	0.853561
Atr6	0.287140	0.102843	0.263032	0.185963	0.297834	1.000000	0.424212	0.184019	0.301342	0.266076	0.340135	0.209801	0.305109	0.224459	0.323787	0.311056
Atr7	0.427989	0.417616	0.464071	0.474806	0.381378	0.424212	1.000000	0.412807	0.517522	0.498266	0.432479	0.511761	0.373361	0.491021	0.494110	0.573290
Atr8	0.802357	0.864284	0.757264	0.798347	0.877584	0.184019	0.412807	1.000000	0.915301	0.828031	0.889795	0.890338	0.840350	0.888822	0.873804	0.865680
Atr9	0.845916	0.827711	0.816653	0.829053	0.916327	0.301342	0.517522	0.915301	1.000000	0.852385	0.911557	0.869088	0.873048	0.868122	0.949041	0.893377
Atr10	0.790183	0.782286	0.753017	0.873636	0.823659	0.266076	0.498266	0.828031	0.852385	1.000000	0.855596	0.847474	0.819715	0.797964	0.853898	0.922320
Atr11	0.892253	0.823380	0.805915	0.808533	0.936955	0.340135	0.432479	0.889795	0.911557	0.855596	1.000000	0.869857	0.909464	0.863584	0.894711	0.877091
Atr12	0.794307	0.862835	0.780258	0.793992	0.846513	0.209801	0.511761	0.890338	0.869088	0.847474	0.869857	1.000000	0.856477	0.899930	0.886885	0.862743
Atr13	0.842996	0.791073	0.758969	0.751623	0.915033	0.305109	0.373361	0.840350	0.873048	0.819715	0.909464	0.856477	1.000000	0.814915	0.876541	0.844350
Atr14	0.817099	0.875800	0.750602	0.757000	0.845576	0.224459	0.491021	0.888822	0.868122	0.797964	0.863584	0.899930	0.814915	1.000000	0.851043	0.837063
Atr15	0.848754	0.801316	0.806909	0.794184	0.879461	0.323787	0.494110	0.873804	0.949041	0.853898	0.894711	0.886885	0.876541	0.851043	1.000000	0.889704
Atr16	0.831822	0.806497	0.775528	0.878416	0.853561	0.311056	0.573290	0.865680	0.893377	0.922320	0.877091	0.862743	0.844350	0.837063	0.889704	1.000000
Atr17	0.895970	0.822317	0.808161	0.809968	0.947429	0.377330	0.461450	0.881005	0.922307	0.843101	0.944807	0.882208	0.917924	0.845456	0.916223	0.892004
Atr18	0.853739	0.883856	0.797395	0.835296	0.894474	0.251856	0.544550	0.941084	0.925543	0.867250	0.909451	0.943030	0.854894	0.929789	0.919839	0.906975

Sample of the Correlation Matrix.

Bigger numbers mean more correlation,irrespective of the sign.

Store the predicting class into another variable and drop it from the Table



Get unlimited access

Open in app

Split Dataset into Training and Testing sets

```
import numpy as np
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42)
print((X_train.shape))
```

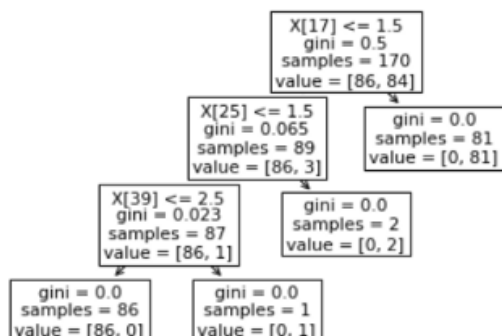
Decision Tree using Sklearn

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import Decision
Tree Classifier
from sklearn import metrics
from sklearn import preprocessing
from sklearn.metrics import accuracy_score
```

```
clf = DecisionTreeClassifier()
clf = clf.fit(X_train, y_train)
```

```
from sklearn import tree
tree.plot_tree(clf.fit(X, y))
```


```
[Text(223.20000000000002, 190.26, 'X[17] <= 1.5\ngini = 0.5\nsamples = 170\nvalue = [86, 84]'),
Text(167.4, 135.9, 'X[25] <= 1.5\ngini = 0.065\nsamples = 89\nvalue = [86, 3]'),
Text(111.60000000000001, 81.53999999999999, 'X[39] <= 2.5\ngini = 0.023\nsamples = 87\nvalue = [86, 1]'),
Text(55.800000000000004, 27.180000000000007, 'gini = 0.0\nsamples = 86\nvalue = [86, 0]'),
Text(167.4, 27.180000000000007, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(223.20000000000002, 81.53999999999999, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(279.0, 135.9, 'gini = 0.0\nsamples = 81\nvalue = [0, 81]')]
```





Get unlimited access

Open in app

 `print(accuracy_score(y_test, y_predict))` 1.0

Resulting accuracy score after being trained on Decision Tree



Training on Perceptron

```
from sklearn.datasets import load_digits
from sklearn.linear_model import Perceptron
```

```
clf = Perceptron(tol=1e-3, random_state=0)
clf.fit(X_train, y_train)
```

```
> Perceptron(alpha=0.0001, class_weight=None, early_stopping=False, eta0=1.0,
fit_intercept=True, max_iter=1000, n_iter_no_change=5, n_jobs=None,
penalty=None, random_state=0, shuffle=True, tol=0.001,
validation_fraction=0.1, verbose=0, warm_start=False)
```

```
clf.score(X_test, y_test)
```

 `clf.score(X_test, y_test)` 1.0

Logistic Regression

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
```





Get unlimited access

Open in app

`clf.score(X_test, y_test)|`

1.0

Neural Networks

First we import all the libraries

```
import keras
import keras
from keras.datasets import mnist
from keras.models import Sequential,model_from_json
from keras.layers import Dense
from keras.optimizers import RMSprop
```

Now,we build our model

```
model = Sequential()
model.add(Dense(54, activation='sigmoid', input_shape=(54,)))
model.add(Dense(128, activation='sigmoid'))
model.add(Dense(32, activation='sigmoid'))
model.add(Dense(1, activation='softmax'))

model.summary()
```





Get unlimited access

Open in app

Layer (type)	Output Shape	Param #
dense_17 (Dense)	(None, 54)	2970
dense_18 (Dense)	(None, 128)	7040
dense_19 (Dense)	(None, 32)	4128
dense_20 (Dense)	(None, 1)	33
Total params: 14,171		
Trainable params: 14,171		
Non-trainable params: 0		

```
model.compile(loss='binary_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])
```

```
model.fit(X_train,y_train,
          batch_size=16,
          epochs=100,
          verbose=1)
```

```
Epoch 1/100
113/113 [=====] - 5s 44ms/step - loss: 8.1828 - acc: 0.4867
Epoch 2/100
113/113 [=====] - 0s 430us/step - loss: 8.1828 - acc: 0.4867
Epoch 3/100
113/113 [=====] - 0s 388us/step - loss: 8.1828 - acc: 0.4867
Epoch 4/100
113/113 [=====] - 0s 446us/step - loss: 8.1828 - acc: 0.4867
Epoch 5/100
113/113 [=====] - 0s 438us/step - loss: 8.1828 - acc: 0.4867
Epoch 6/100
113/113 [=====] - 0s 455us/step - loss: 8.1828 - acc: 0.4867
Epoch 7/100
113/113 [=====] - 0s 379us/step - loss: 8.1828 - acc: 0.4867
Epoch 8/100
113/113 [=====] - 0s 397us/step - loss: 8.1828 - acc: 0.4867
Epoch 9/100
113/113 [=====] - 0s 373us/step - loss: 8.1828 - acc: 0.4867
Epoch 10/100
113/113 [=====] - 0s 400us/step - loss: 8.1828 - acc: 0.4867
Epoch 11/100
113/113 [=====] - 0s 327us/step - loss: 8.1828 - acc: 0.4867
Epoch 12/100
113/113 [=====] - 0s 315us/step - loss: 8.1828 - acc: 0.4867
Epoch 13/100
```





Get unlimited access

Open in app

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification

clf = RandomForestClassifier(n_estimators=100, max_depth=2,
                           random_state=0)

clf.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                        max_depth=2, max_features='auto', max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=100,
                        n_jobs=None, oob_score=False, random_state=0, verbose=0,
                        warm_start=False)
```

```
clf.score(X_test, y_test)
```



0.9824561403508771

Thus, we have successfully trained different models for predicting whether a couple will get divorced or not.

Other Evaluation Metrics

All these metrics are being computed for the random forest model

Confusion Matrix

```
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, predictions)
```





Get unlimited access

Open in app

The first row is about the not-divorced-predictions: 28 **couples were correctly classified as not divorced** (called true negatives) and 0 **where wrongly classified as not divorced** (false positives).

The second row is about the divorced-predictions: 1 **couples where wrongly classified as divorced** (false negatives) and 28 **were correctly classified as divorced** (true positives).

A confusion matrix gives you a lot of information about how well your model does, but there's a way to get even more, like computing the classifiers precision.

Precision and Recall

Recall literally is how many of the true positives were recalled. Precision is how many of the returned hits were true positives.

```
from sklearn.metrics import precision_score, recall_score

print("Precision:", precision_score(y_test, predictions))
print("Recall:", recall_score(y_test, predictions))
```

```
➤ Precision: 1.0
Recall: 0.9655172413793104
```

F1 Score

```
from sklearn.metrics import f1_score
f1_score(y_test, predictions)
```

```
[45] from sklearn.metrics import f1_score
      f1_score(y_test, predictions)
```





Get unlimited access

Open in app

methods which measure how well we have performed.

