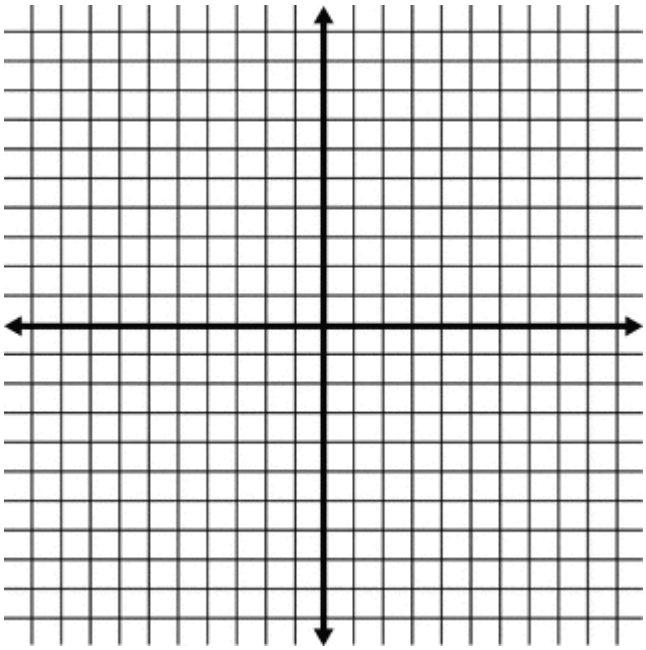
Grid Search for model tuning

Rohan Joseph Follow
Dec 30, 2018 · 5 min read



A model ***hyperparameter*** is a characteristic of a model that is external to the model and whose value cannot be estimated from data. The value of the hyperparameter has to be set before the learning process begins. For example, *c* in Support Vector Machines, *k* in k-Nearest Neighbors, *the number of hidden layers* in Neural Networks.

In contrast, a ***parameter*** is an internal characteristic of the model and its value can be estimated from data. Example, beta coefficients of linear/logistic regression or support vectors in Support Vector Machines.

> *Grid-search is used to find the optimal* hyperparameters *of a model which results in the most 'accurate' predictions.*

Let's look at Grid-Search by building a classification model on the Breast Cancer dataset.

1. Import the dataset and view the top 10 rows.

```
#import data
data = pd.read_csv('breast-cancer-wisconsin.csv',header
#set column names
data.columns = ['Sample Code Number','Clump Thickness',
```

Output :

	Sample Code Number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2
5	1017122	8	10	10	8	7	10	9	7	1	4
6	1018099	1	1	1	1	2	10	3	1	1	2
7	1018561	2	1	2	1	2	1	3	1	1	2
8	1033078	2	1	1	1	2	1	1	1	5	2
9	1033078	4	2	1	1	2	1	2	1	1	2

Each row in the dataset have one of two possible classes: benign (represented by 2) and malignant (represented by 4). Also, there are 10 attributes in this dataset (shown above) which will be used for prediction, except Sample Code Number which is the id number.

2. Clean the data and rename the class values as 0/1 for model building (where 1 represents a malignant case). Also, let's observe the distribution of the class.

```
data = data.drop(['Sample Code Number'],axis=1) #Drop 1
data = data[data['Bare Nuclei'] != '?'] #Remove rows wi
data['Class'] = np.where(data['Class'] ==2,0,1) #Change
data['Class'].value_counts() #Class distribution
```

Output :

```
0    444
1    239
Name: Class, dtype: int64
```

There are 444 benign and 239 malignant cases.

3. Before building a classification model, let's build a Dummy Classifier to determine the 'baseline' performance. This answers the question—'What would be the success rate of the model, if one were simply guessing?' The dummy classifier we are using will simply predict the majority class.

```
#Split data into attributes and class
X = data.drop(['Class'],axis=1)
y = data['Class']

#perform training and test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,

#Dummy Classifier
from sklearn.dummy import DummyClassifier
clf = DummyClassifier(strategy= 'most_frequent').fit(X
y_pred = clf.predict(X_test)
```

Output :

```
y actual :
0    103
1     68


y predicted :
0    171
```

From the output, we can observe that there are 68 malignant and 103 benign cases in the test dataset. However, our classifier predicts all cases as benign (as it is the majority class).

4. Calculate the evaluation metrics of this model.

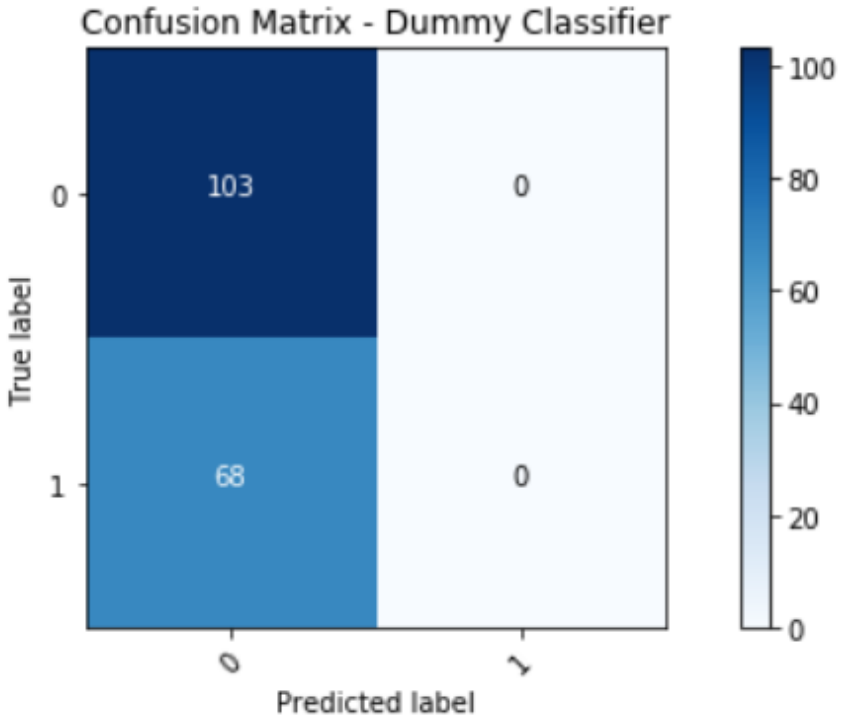
```
# Model Evaluation metrics
from sklearn.metrics import accuracy_score,recall_scor
print('Accuracy Score : ' + str(accuracy_score(y_test,
print('Precision Score : ' + str(precision_score(y_tes
print('Recall Score : ' + str(recall_score(y_test,y_pr
print('F1 Score : ' + str(f1_score(y_test,y_pred)))

#Dummy Classifier Confusion matrix
```

Output :

```
Accuracy Score : 0.6023391812865497
Precision Score : 0.0
Recall Score : 0.0
F1 Score : 0.0
```

```
Confusion Matrix :
[[103   0]
 [ 68   0]]
```

The accuracy of the model is 60.2%, but this is a case where accuracy may not be the best metric to evaluate the model. So, let's take a look at the other evaluation metrics.



The above figure is the confusion matrix, with labels and colours added for better intuition (Code to generate this can be found here). To summarize the confusion matrix : TRUE POSITIVES (TP) = 0,TRUE NEGATIVES (TN) = 103,FALSE POSITIVES (FP) = 0, FALSE NEGATIVES (FN) = 68. The formulae for the evaluation metrics are as follows :

$$\boldsymbol{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN} = \frac{0 + 103}{0 + 0 + 68 + 103} = 60.23\%$$

$$\boldsymbol{Precision} = \frac{TP}{TP + FP} = \frac{0}{0 + 0} = 0\%\ (not\ defined)$$

$$\boldsymbol{Recall} = \frac{TP}{TP + FN} = \frac{0}{0 + 68} = 0\%$$

$$\boldsymbol{F1} = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * 0 * 0}{0 + 0} = 0\%\ (not\ defined)$$

Since the model does not classify any malignant case correctly, the recall and precision metrics are 0.

5. Now that we have the baseline accuracy, let's build a Logistic regression model with default parameters and evaluate the model.

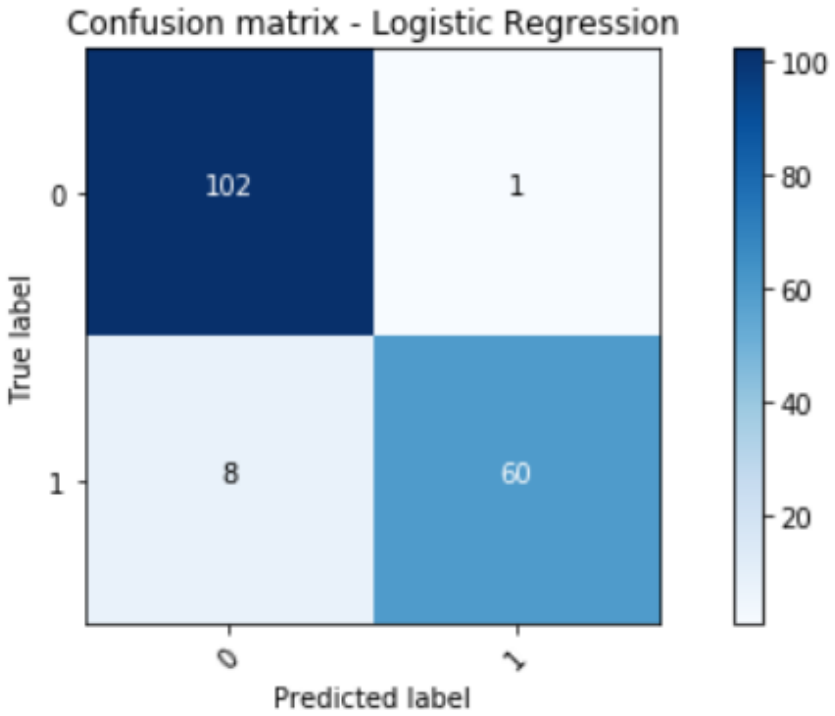
```
#Logistic regression
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression().fit(X_train,y_train)
y_pred = clf.predict(X_test)

# Model Evaluation metrics
from sklearn.metrics import accuracy_score,recall_scor
print('Accuracy Score : ' + str(accuracy_score(y_test,
print('Precision Score : ' + str(precision_score(y_tes
print('Recall Score : ' + str(recall_score(y_test,y_pr
```

Output :

```
Accuracy Score : 0.9473684210526315
Precision Score : 0.9836065573770492
Recall Score : 0.8823529411764706
F1 Score : 0.9302325581395349
```

```
Confusion Matrix :
[[102   1]
 [  8  60]]
```

By fitting the Logistic Regression model with the default parameters, we have a much 'better' model. The accuracy is 94.7% and at the same time, the Precision is a staggering 98.3%. Now, let's take a look at the confusion matrix again for this model results again :



Looking at the misclassified instances, we can observe that ***8 malignant cases have been classified incorrectly as benign (False negatives)***. Also, just one benign case has been classified as malignant (False positive).

A false negative is more serious as a disease has been ignored, which can lead to the death of the patient. At the same time, a false positive would lead to an unnecessary treatment—incurring additional cost.

Let's try to minimize the false negatives by using Grid Search to find the optimal parameters. Grid search can be used to improve any specific evaluation metric.

> *The metric we need to focus on to reduce false negatives is* ***Recall****.*

6. Grid Search to maximize Recall

```
#Grid Search
from sklearn.model_selection import GridSearchCV
clf = LogisticRegression()
grid_values = {'penalty': ['l1', 'l2'],'C':[0.001,.009
grid_clf_acc = GridSearchCV(clf, param_grid = grid_val
grid_clf_acc.fit(X_train, y_train)

#Predict values based on new parameters
y_pred_acc = grid_clf_acc.predict(X_test)

# New Model Evaluation metrics
print('Accuracy Score : ' + str(accuracy_score(y_test,
```

Output :

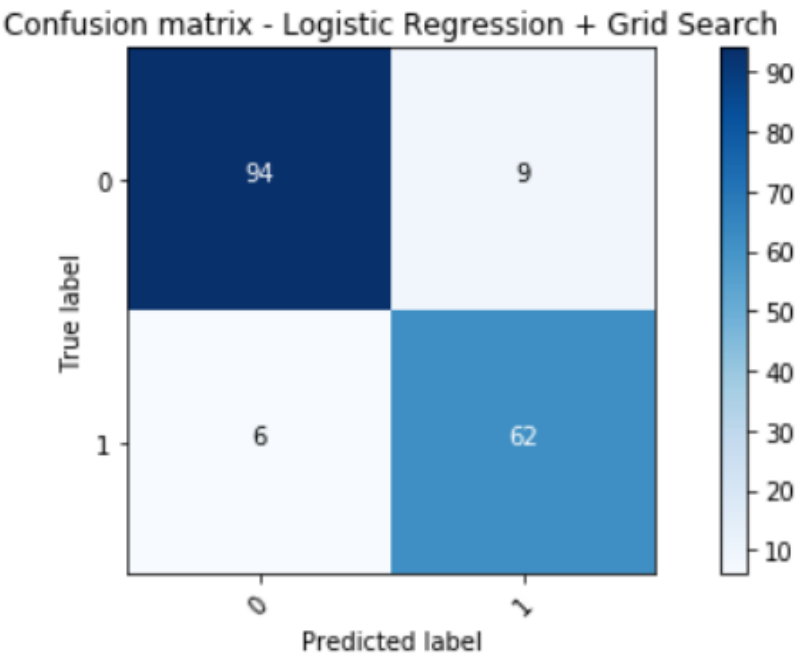
```
Accuracy Score : 0.9122807017543859
Precision Score : 0.8732394366197183
Recall Score : 0.9117647058823529
F1 Score : 0.8920863309352517
```

```
Confusion Matrix :
[[94  9]
 [ 6 62]]
```

The hyperparameters we tuned are:

1. Penalty: l1 or l2 which species the norm used in the penalization.
2. C: Inverse of regularization strength- smaller values of C specify stronger regularization.

Also, in Grid-search function, we have the scoring parameter where we can specify the metric to evaluate the model on (We have chosen recall as the metric). From the confusion matrix below, we can see that the number of false negatives has reduced, however, it is at the cost of increased false positives. The recall after grid search has jumped from 88.2% to 91.1%, whereas the precision has dropped to 87.3% from 98.3%.



You can further tune the model to strike a balance between precision and recall by using 'f1' score as the evaluation metric. Check out this article for a better understanding of the evaluation metrics.

Grid search builds a model for every combination of hyperparameters specified and evaluates each model. A more efficient technique for hyperparameter tuning is the Randomized search—where random combinations of the hyperparameters are used to find the best solution.

. . .

Connect on LinkedIn and, check out Github (below) for the complete notebook.

rohanjoseph93/Python-for-data-science

Learn data science with Python. Contribute to rohanjoseph93/Python-for-data-science...

github.com