Classifying Malignant or Benignant Breast Cancer using SVM

Gabriel Mayers | Follow

May 26 · 4 min read ★

GIF here

Support Vector Machines(SVM) is a very powerful algorithm, mainly when we're talking about Classification Problems.

In this post, we'll implement **SVM Algorithm** to *Classify Breast Cancer into Malignant or Benignant.*

But first, let's see a small intuition about SVM!

An Small Intuition of SVM

SVM is defined as: Binary Linear Classifier, where, the principal goal is draw a hyperplan to divide the 2 classes, like the GIF above.

In outlier cases, the SVM search for the best classification, or if necessary, disregard the outlier. SVM's tends to work very good in problems where we have a clear data separation.

Besides that, SVM's can perform badly in Datasets where we have many noises!

If you wanna know more about SVM's see this article.

Now, let's understand our problem!

Understanding the Problem

We wanna use the Breast Cancer Dataset from sklearn, where we have:

- ***2 Classes: Malignant(0) and Benignant(1)***
- ***569 Examples***
- ***31 Columns with Attributes and the respective class***

I really don't if this data is True, but will serve too good for our Model.

Basically, the challenge is: Given a list of features, our model needs ***classify*** if, based on these features, the breast cancer is ***Malignant or Benignant***.

A very good practice is visualize your data before start to build your Model, in ***real-world problems***, you need to discover what approach is better to solve your problem. In this post, we already know we wanna use SVM, but let's visualize our Data to know more about what we're inputting into our Model.

Visualizing the Dataset

First, to improve our visualization, we can convert our data into a Pandas DataFrame using the code below:

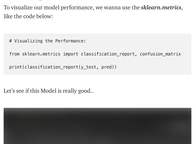
```
cancer = load_breast_cancer()

# Convert into DataFrame:

data = pd.DataFrame(cancer.data, columns=cancer.feature_names)

data['target'] = pd.Series(cancer.target)
```

Now, we can visualize our data into a ***DataFrame***!



Too columns to visualize complete!

We wanna use all the columns, more data is better to our Model.

> ***Remember:*** *More data not necessarily means more performance! in Machine Learning,* **Garbage in Garbage out!** *You need to decide what can improve the performance of your Model and what cannot.*

After decide what we wanna use and what not, we can split our data into ***Train and Test***.

To do that, we can use the ***sklearn.model_selection***, like the code below:

```
# Splitting data:

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data, np.ravel(data['target']), test_size=0.3)
```

Note: **np.ravel(data['target'])** *transform our target data into 1D array!*

After split our data into train and test, we can build our Model!

Building our Model

To build our Model, we wanna use ***sklearn.svm***, like the code below:

```
# Building our Model:

from sklearn.svm import SVC

model = SVC()
```

Now, we can fit our Model using the ***fit()*** method passing as parameters our ***X_train and y_train***.

```
model.fit(X_train, y_train)
```

We already have a Model trained and ready to make predictions, now, we can make predictions in our ***X_test***.

```
pred = model.predict(X_test)
```

Now, we wanna visualize how our Model are performing!

Visualizing the Performance of our Model

To visualize our model performance, we wanna use the ***sklearn.metrics***, like the code below:

```
# Visualizing the Performance:

from sklearn.metrics import classification_report, confusion_matrix

print(classification_reporty(y_test, pred))
```

Let's see if this Model is really good...

Very Good Results, Sally!

Now, we have a Model trained and ready to make predictions with a very good accuracy!

Bonus

We already have a good performance for our Model, but we can try to improve it by using a process called **"Grid Search"**.

Basically, Grid Search is a process to find the best parameters to our model, making him performe better. This process consists in try a small or big number of parameters to find the best.

See more about GridSearch in here.

Let's make it into our Model!

First, we need to import ***GridSearchCV*** from ***sklearn.model_selection***, and after, create a *dictionary* with the values for each parameter of we wanna test, like the code below:

```
# GRID SEARCH:

from sklearn.model_selection import GridSearchCV

param_grid = {'C': [0.1, 1, 10, 100, 1000], 'gamma': [1, 0.1, 0.01, 0.001, 0.0001], 'kernel': ['rbf']}
```

Now, we can instantiate the ***GridSearchCV*** into our variable ***grid_svm*** and train our GridModel testing all the parameters of we established before:

```
grid_svm = GridSearchCV(SVC(), param_grid, refit=True, verbose=3)

grid_svm.fit(X_train, y_train)
```

We can see the best parameters chose by our GridSearch using the ***grid_svm.best_params_***.

We already can make predictions using our GridModel:

```
pred_grid = grid_svm.predict(X_test)
```

Now, we can visualize our metrics using the ***classification_report*** and see if our Model really improved:

Improve and Not. Improve

Our model improved the accuracy to ***Benignant***, but not improved to ***Malignant***. This is not bad, actually, this is better the our previous metrics!

You can make the ***GridSearch*** using different parameters and try to improve your Model!

You can access the notebook of this post in here.

For now, this is all!

See you next time!

My Social Medias:

LinkedIn: https://www.linkedin.com/in/gabriel-mayer-779b5a162/

GitHub: https://github.com/gabrielmayers

Instagram: https://www.instagram.com/gabrielmayerf/

Data Science | Machine Learning | Machine Learning AI | Classification | Classification Algorithms

Written by

Gabriel Mayers | Follow

Artificial Intelligence Engineer, Science enthusiast, Self-taught and so curious.

Analytics Vidhya | Follow

Analytics Vidhya is a community of Analytics and Data Science professionals. We are building the next-gen data science ecosystem https://www.analyticsvidhya.com

Write the first response

More From Medium

Deploying Across Heterogeneous Accelerators at the Edge in Kubernetes
Paul Mundt in The Startup

Dog Breed Identifier with CNN
Nivu Leung

Understanding Gradient Descent
Nilesh Barla in Analytics Vidhya

The Science of Adaptive Bit Rates
The ALLAN in DataArena

Classification
Arshed Patil in The Startup

The Science of Building a Confidence Score
The ALLAN in The Startup

PAC Learning Theory for the Everyman
Allison Kelly in The Startup

Federated Learning: A collaborative approach of machine learning
Pragati Baheti

Discover Medium

Welcome to a place where words matter. On Medium, smart voices and original ideas take center stage - with no ads in sight. Watch

Make Medium yours

Follow all the topics you care about, and we'll deliver the best stories for you to your homepage and inbox. Explore

Become a member

Get unlimited access to the best stories on Medium — and support writers while you're at it. Just $5/month. Upgrade

Medium | About | Help | Legal