Piero Paialunga Follow

Dec 13, 2020 · 6 min read · · Listen

Shot by Takahiro Sakamoto

Ensemble Learning with Support Vector Machines and Decision Trees

> *Note: This article is a part of a bigger project that can be seen and hopefully loved in this GitHub repository*

Let's pretend for a second that Machine Learning models are real human beings: **none of them is perfect** (besides you, of course).

Some models could be too anxious, someone too jealous, someone too arrogant. The real magic happens when you fall in love with someone that is able to see your weak points and helps you improve them, and he/she emphasises your good sides. This is the exact idea of **Ensemble Learning.**

In fact, it is based on the idea that a wide collection of models is able to perform better than each model that is taken in isolation.

It may sound complicated at a first sight, so let's proceed with a real dataset and learn it by example.

The Libraries:

Open in app Get started

```
from matplotlib import cm
import numpy as np
from sklearn.utils.testing import ignore_warning
from sklearn.exceptions import ConvergenceWarnin
from sklearn.decomposition import PCA
import seaborn as sns
from sklearn.svm import SVC
from sklearn.model_selection import train_test_s
```

Hosted on Jovian View File

The Dataset:

The dataset comes from the astrophysics framework and it is a collection of stars. **The goal of this Machine Learning adventure is to get the sign of the Sharp value that represents how much the star is broader with respect to the astrophysics model profile.** In other terms, you have a wide variety of features like spatial coordinates (X,Y), fluxes at a certain wavelength and their errors (F606W,F814W, error and error.1) and the goodness of fit statistics (Chi) and you want to get if the star's profile:

- A) Is broader than it should be (Sharp>0)
- B) Is stricter than it should be (Sharp<0)
- C) Is just as it should be (Sharp=0)

We are thus talking about a 3 classes classification.

```
#Importing the dataset and displaying the first
```

Hosted on Jovian View File

	#ID	X	Y	F606W	error	F814W	error.1
0	8	4462.947	140.859	28.197	0.1036	27.127	0.1068
1	120	5002.486	186.138	20.843	0.0552	19.815	0.0661

Hosted on Jovian View File

The Methods:

0. Principal Component Analysis

A dimensionality reduction has been applied to this dataset in order to increase the algorithm's performance. Interesting stuff about it can be found here.

```
#Dropping the X and the Y to perform PCA
notar=data.drop(columns=['X','Y'])
pca=PCA(n_components=3)
pca=pca.fit(notar)
pca_data=pd.DataFrame(pca.transform(notar))
pca_data=pca_data.rename(columns={0:'FirstCompon
pca_data['X']=data.X
```

Hosted on Jovian View File

1. Support Vector Machines

The Support Vector Machine algorithm is one of the most powerful one out there in terms of classification. It is based on the idea of getting the largest margin (distance) between the points of the dataset (in particular a set of them, call support vectors) and the separation hyperplane.

report to explain this properly but for our specific goal let's say that there are two hyperparameters that we can tune to get the best classification algorithm.

- Kernel Type: 'linear', 'rbf', 'poly','sigmoid'
- C value: from 0 to infinity

Hey, I almost forgot, SVMs are damn expensive.

For this reason, only two features, the most informative two components of the P.C.A. have been used in this context. Unfortunately, it is not enough. A small portion (but still consistent) of the original data has been considered (10%) for the training set, and 90% for the test set too.

The training set has been split in half (training and validation) to perform the hyperparameter tuning that has been explained above. The training and validation set has been changed iteratively 5 times.

You can see the hyperparameter tuning code in this Jovian link if you want, but it is not that interesting, so I'm going to show the results on the validation set:

- Preferred Kernel Type: 'rbf'
- Preferred C value = 14.5

Now that we have them, the performance of the algorithm has been tested slightly increasing the training set size (30% of the dataset).

The entire process is described here line by line:

```
#Two features, three classes preprocessing
```

Hosted on Jovian View File

Hosted on Jovian View File

The Target (Sharp Sign)

```
#Dataset and target
X=opt_data.drop(columns=['Target'])
```

Hosted on Jovian View File

Dataset+Target

```
#Best parameters
best_c=14.5
```

Hosted on Jovian View File

Best parameters

```
#Train/Test rigid split
X_train, X_test, y_train, y_test = train_test_sp
```

Hosted on Jovian View File

30/70% split

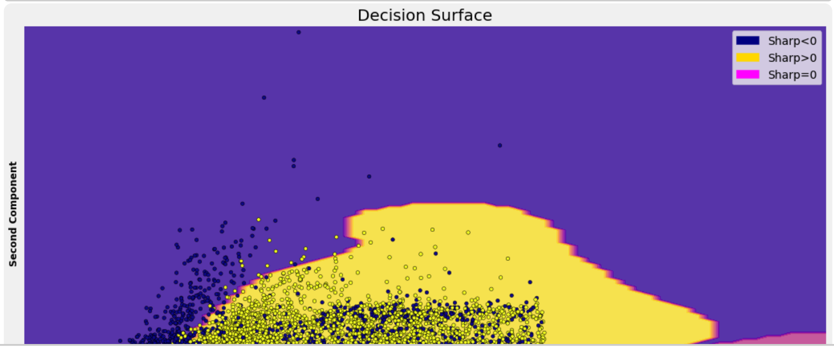
```
#Fit with the best parameters
clf=SVC(kernel=best_kernel,C=best_c)
clf.fit(X_train,y_train)
fin_score=clf.score(X_test,y_test)
#Target/prediction comparison
pred_data=X_test.copy()
```

Prediction

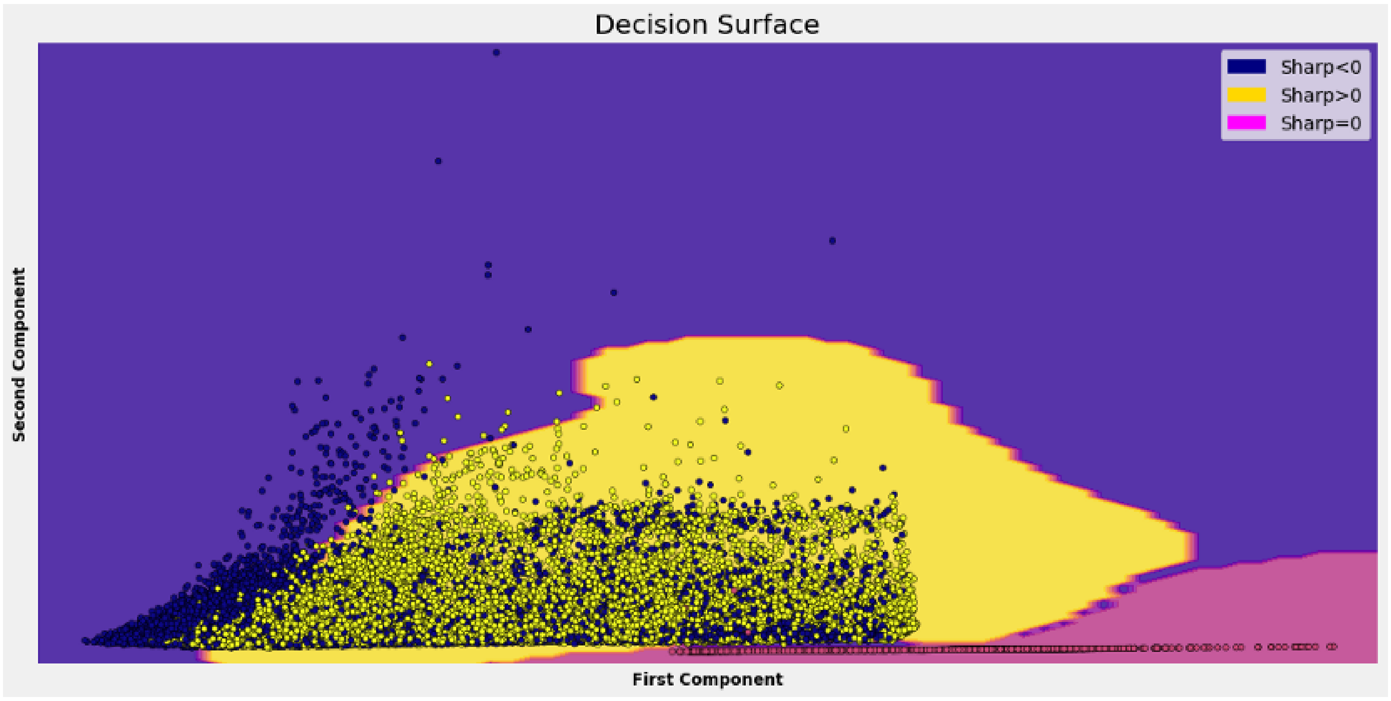
72% of accuracy has been obtained by using this model (no, the classes are not well balanced, but this is not the problem).

Let's see how the Support Vector Machine algorithm intends this classification.

```
ax.scatter(X0, X1, c=y, cmap='plasma', s=20, edg
ax.set_ylabel('Second Component')
ax.set_xlabel('First Component')


violet_patch = mpatches.Patch(color='navy', labe
yellow_patch = mpatches.Patch(color='gold', labe
pink_patch = mpatches.Patch(color='magenta', lab

plt.legend(handles=[violet_patch,yellow_patch,pi

ax.set_xticks(())
ax.set_yticks(())
ax.set_title('Decision Surface', fontsize=20)
#ax.legend()
plt.show()
```



Hosted on Jovian View File



It's interesting.

The 0 (null) class is actually well classified by the Support Vector Machine algorithm. Even the -1 zone for the Support Vector Machine algorithm is pretty precise. Anyway the algorithm has huge problem in the recall of the negative points. In fact a lot of negative points are actually classified as positive. **Concretely speaking: the "positive" zone is a mess.**

You may think "Ok, but you only used 2 features out of 7", and this is right. But the main problem is that increasing the features means stressing the computational power of our computer, so it is not the best scenario ever.

What can we do to increase the performance of our algorithm without using days of computational power?

2. Decision Trees

Decision trees are powerful algorithms that are cheaper than the Support Vector

can have an high number of the points that belong to a specific class. You can build a massive amount of trees out of a dataset, and for this reason you can try to add randomness to this process and build not a single tree but an entire forest where each tree can see only a specific portion of the dataset or a specific number of features. This is per se an ensemble learning algorithm and it is called **"random forest".**

Even the random forest have their hypeparameter tuning that can be applied on a bunch of different hyperparameters. I don't want to kill you, so I'm going fast on this one: you can see all the work on these hyperparameters on this notebook.

The interesting things here are that you can use the entire dataset and all the features to have your classification, as this method is not so expensive!

The performance of this model are actually even better than the SVM model (80%)

3. Ensemble Learning

So why don't we use the Random Forest optimum model on the messy zone of the dataset?

In order to help the SVM algorithm let's train again the Random Forest optimum model (best hyperparameters) on a portion of the test set of the SVM algorithm.

```
#Implement random forest on a split set
X=data.loc[X_test.index].drop(columns=['#ID','Sh
y=data.loc[X_test.index].SharpSign
```

Hosted on Jovian View File

80/20 split

```
m Forest Model
```

```
odel on training data
```

Now we have:

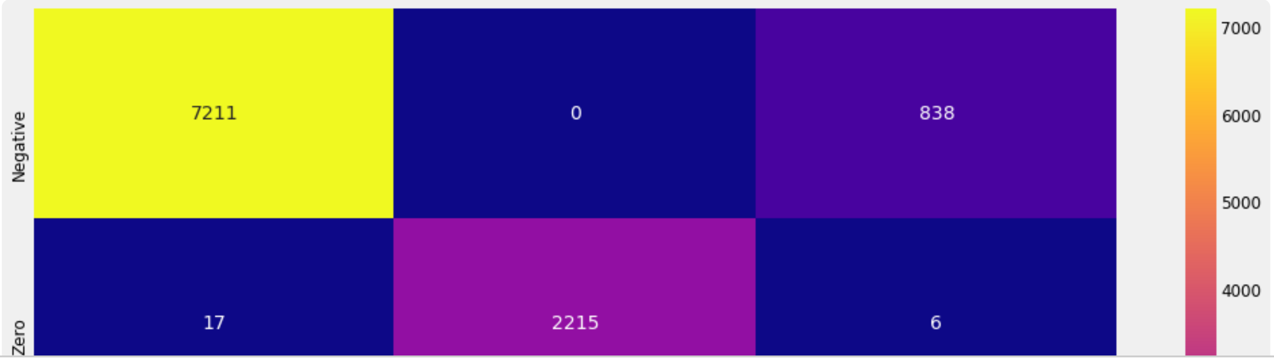
- The 0 class that is almost correctly classified by the SVM algorithm
- The -1 class that is almost correctly classified by the SVM algorithm
- The 1 class that is classified by the Random Forest

Let's merge the result and see what we've got:

```
#Collecting the results of the random forest on this middle area
#And the SVM on the -1 and 0 points
#Combining the features for the Random Forest results
Results=X_test.copy()
Results['Target']=y_test
Results['Pred']=rf.predict(X_test)
Results['FirstComponent']=opt_data['FirstComponent'].loc[Results.index]
Results['SecondComponent']=opt_data['SecondComponent'].loc[Results.index]
#Doing the same for the SVM results
good_ones=pred_data[(pred_data.Prediction==0.) |(pred_data.Prediction==-1.
for feat in feature_names:
    good_ones[feat]=data[feat].loc[good_ones.index]
good_ones=good_ones.rename(columns={'Prediction':'Pred'})
#Building the total resume
Results=Results.append(good_ones)
acc=accuracy_score(Results.Pred, Results.Target)

#Computing the confusion matrix
y_test=Results.Target
```

Open in app Get started

```
ax = sns.heatmap(df_cm, cmap='plasma',annot=True,fmt='g')
```



Hosted on Jovian View File

Almost 82% of accuracy (+2% on the previous model)

```
#Summary of all the methods
Tot_res=pd.DataFrame({'Performance':[71,74,80,82]})
Tot_res.index=['SVM','Decision Tree','Random Forest','Ensemble Learning']
Tot_res
```

	Performance
SVM	71
Decision Tree	74
Random Forest	80
Ensemble Learning	82

```
ERROR! Session/line number was not unique in database. History logging moved
to new session 1008
```

Hosted on Jovian View File

Some notes:

1. **The Machine Learning methods "explore" the same dataset with different perspectives.** In this scenario the best algorithm was obviously the Random Forest,

2. **The result is better than you may think**, as the SVM uses only the 30% of the dataset and they want to perform the remaining 70%
3. **The dataset is pretty huge and it imposed some practical restrictions**, but you could use this approach to a smaller dataset and maybe use the entire dataset for the SVM algorithm too.
4. **The entire data analysis and Machine Learning project is reported in this GitHub repository.**

If you liked the article and you want to know more about Machine Learning, or you just want to ask me something you can:

A. Follow me on **Linkedin**, where I publish all my stories
B. Subscribe to my **newsletter**. It will keep you updated about new stories and give you the chance to text me to receive all the corrections or doubts you may have.
C. Become a **referred member**, so you won't have any "maximum number of stories for the month" and you can read whatever I (and thousands of other Machine Learning and Data Science top writer) write about the newest technology available.