Project Work - Contact Classification

Alberto Crivellari
Structural Bioinformatics, Academic Year 2021 - 2022

July 10, 2022

Part I
Introduction

This is the report for the project of Structural Bioinformatics 2021/22.
For this project we have to create:

- a report on the project, this file;
- some python code for classification of contacts between pairs of residues, either python scripts or python notebooks;
- a file .md, markdown file, on how to use/run the code.

1 Goal of the Project

The goal of the project is to correctly classify the contacts between each protein residue. There are 7 possible type of contacts:

- Hydrogen Bonds (HBOND)
- Van der Waals interactions (VDW)
- Disulfide bridges (SBOND)
- Salt bridges (IONIC)
- π-π stacking (PIPISTACK)
- π-cation (PICATION)
- Unclassified contacts

In the training dataset there are very unbalanced classes:

Contact Type	Samples
HBOND	333,346
VDW	155,789
PIPISTACK	10,403
IONIC	9,068
SSBOND	866
PICATION	626
Unclassified	225,412

Part II
Pre-Processing

After a through read of the project specification, the first part involves the pre-processing of the files.

Specifically I need to build a dataframe starting from over 1500 .tsi files. This can be done using a combination of Jupyter and Python. Unfortunately, due to the number of the files that need to be imported, I had to do this part off-line, since it requires some time and resources.

So, I will start the notebook directly with the dataframe created.

2 Downloading

I dowloaded the resources, of pdb files, from the shared Google Doc document and unzip them inside a folder

3 DataFrame Building

Then, I built the dataframe using simple lines of code:

```
# Combine all PDBs into a single dataframe
dfs = []
for filename in os.listdir('features_ring'):
    dfs.append(pd.read_csv('features_ring/' + filename, sep='\t'))
df = pd.concat(dfs)
```

4 DataFrame Importing

Next, I exported the dataframe into a csv and imported it on an online tool.

The final choice was Google Colab. (Source file: https://colab.research.google.com/)

For reading the dataframe in .csv file, I used the following code:

```
df = pd.read_csv(path + 'train.csv')
```

Where "path" is a variable containing the path to the directory containing the source file csv.

I renamed the .csv for training, *train.csv* and the csv for testing, *test.csv*. So if someone wants to use others datasets to train or test, it is possible to just replace them and rename these new datasets *train.csv* and *test.csv*.

5 Dataframe Analysis

To get an idea of what are the contents of the dataframe I used these commands:

```
df.describe()
df.info()
```

I noticed that the dataframe has 35 features with 10 of them being categorical features and one of them being the target feature: "Interaction". There are 735.510 observations (one for each bond in each protein of the dataset).

The dataset features can be splitted in four parts:

- **Source Residue Identifier:** which contains the chain, the index, the insertion code and the name of the source residue

- **Source Residue Features:** which contains the secondary structure 8 states, the relative solvent accessibility, the half sphere exposure up, the half sphere exposure down, the phi angle, the psi angle, the secondary structure 3 states and the Atchley features of the source residue
- **Target Residue Identifier:** which contains the chain, the index, the insertion code and the name of the target residue
- **Source Residue Features:** which contains the secondary structure 8 states, the relative solvent accessibility, the half sphere exposure up, the half sphere exposure down, the phi angle, the psi angle, the secondary structure 3 states and the Atchley features of the target residue

6 DataFrame Cleaning

I removed from the dataset the observations with nan values and the duplicated ones:

```
df.dropna(inplace = True)
```

Part III
Machine Learning

7 Feature Selection

The "pdb_id" feature is useless to create prediction on the bond type, so I decided to drop it.

8 Algorithm Exploration

For the model I used the *pycaret* package.

```
df = df.sample(frac = 1) #Takes all the dataset

exp = setup(df, target = 'Interaction', fold = 5, normalize = True)
bests = compare_models(exclude = ['gbc','lr', 'svm',
    'et', 'knn', 'ada','rf', 'dt', 'qda','lda','nb'])
```

Notice that I take all the dataset, although computationally-heavy algorithm for resource reasons, it gives a little better results.

After computing, the Light Gradient Boosting Machine (LGBM) is the better model, between the ones provided from *pycaret*.

```
model = create_model('lightgbm')
tuned_model = tune_model(model)
```

9 Some Graphics on the Model

In this section I will show a couple graphics on the model selected, the LGBM classifier:

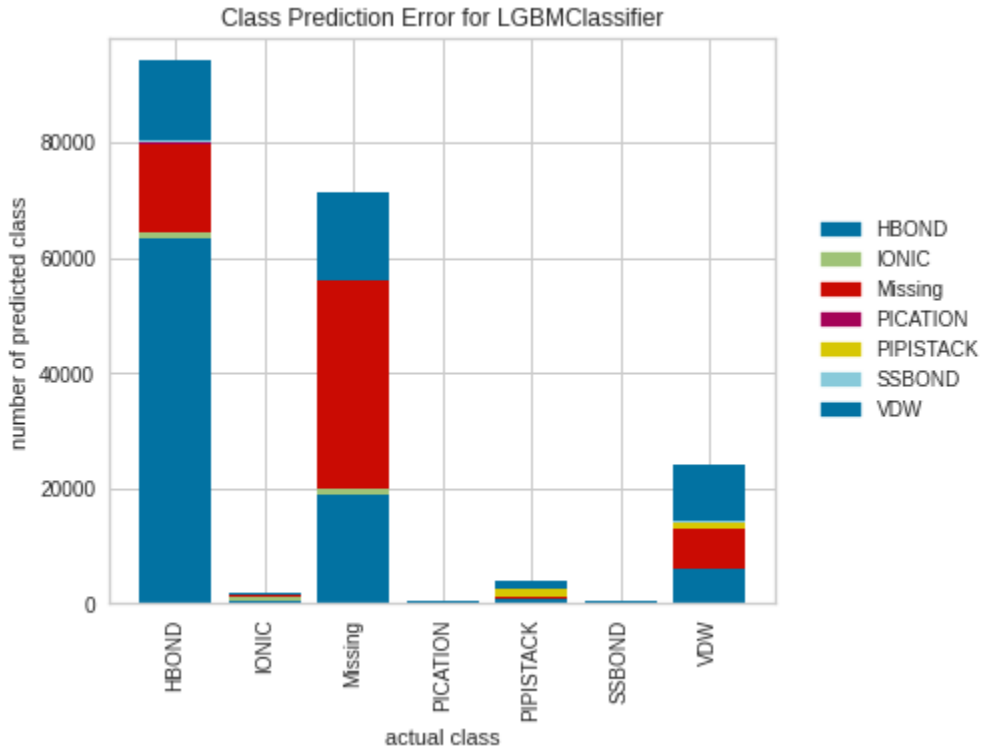

Figure 1: Class Prediction Error Graphic

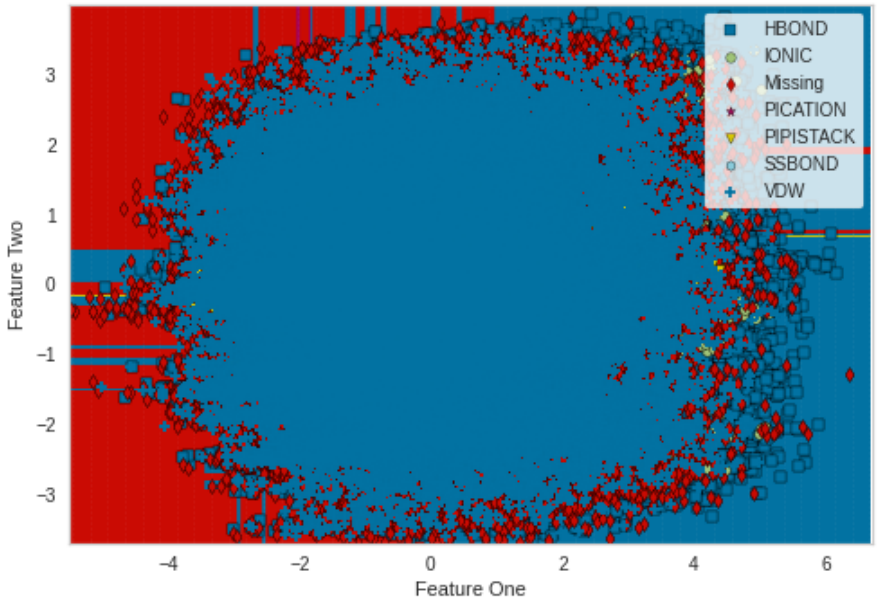

Figure 2: Boundaries plot

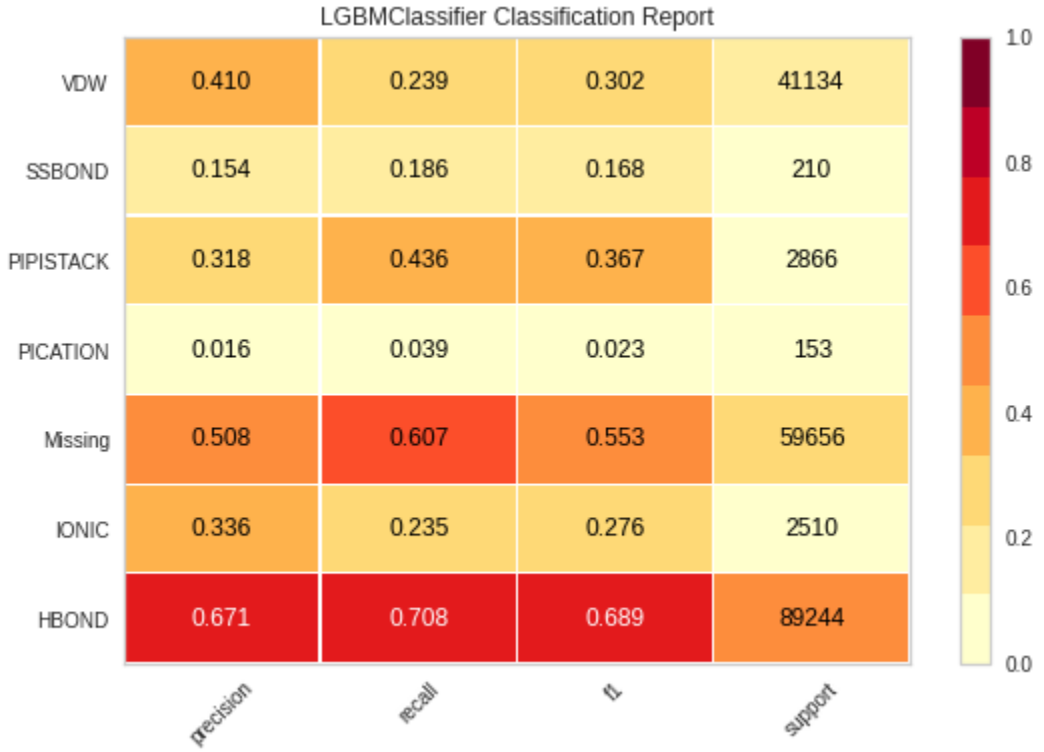

Figure 3: Classification Report

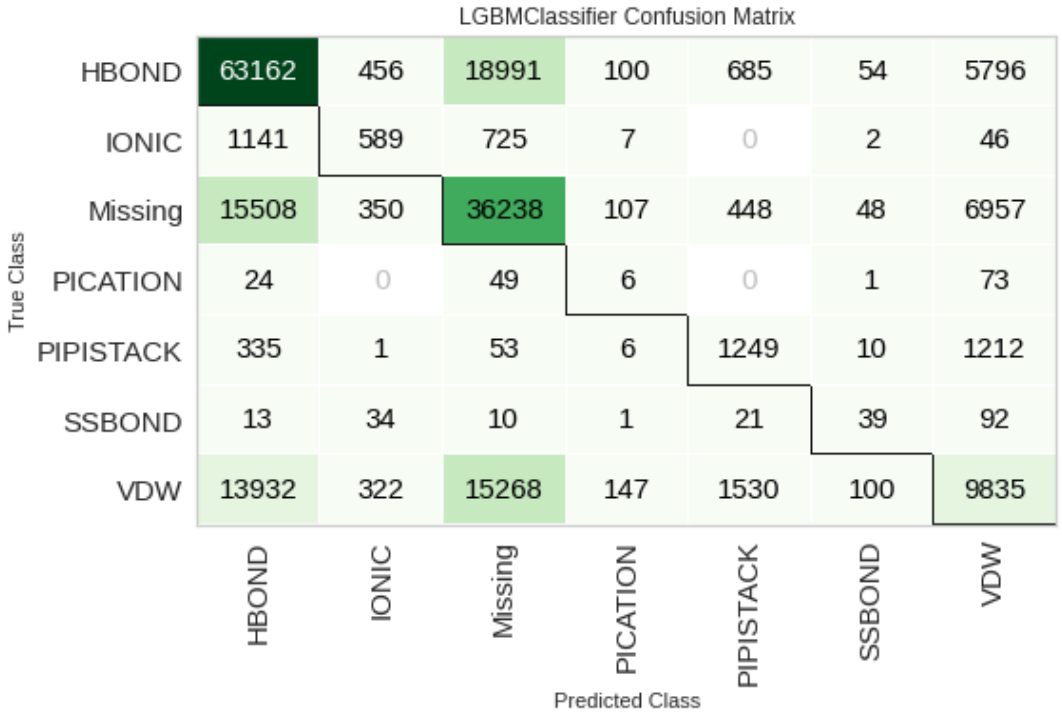

Figure 4: Confusion Matrix

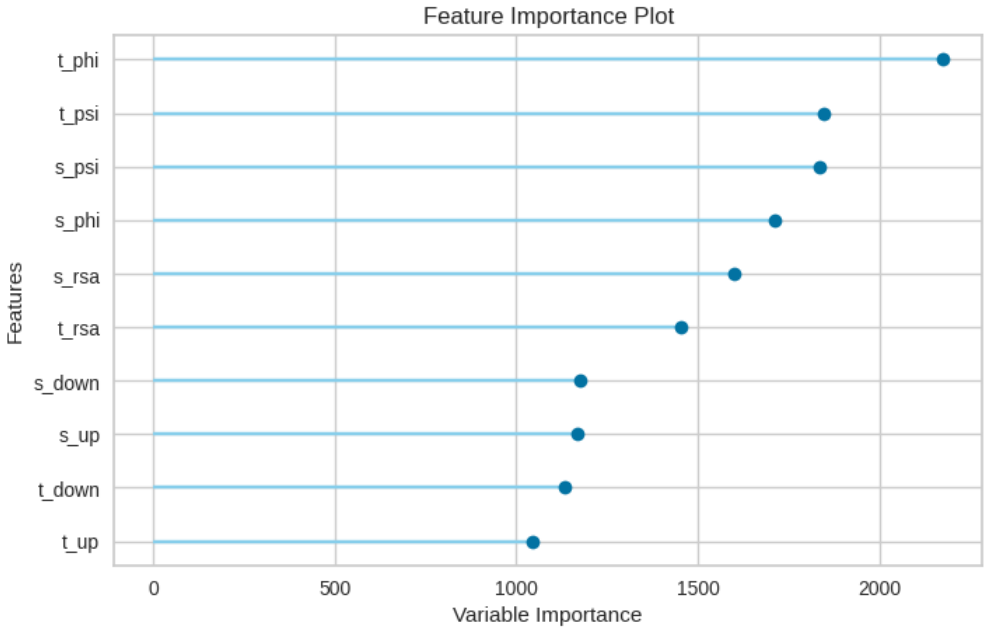

Figure 5: Feature-Importance graphic

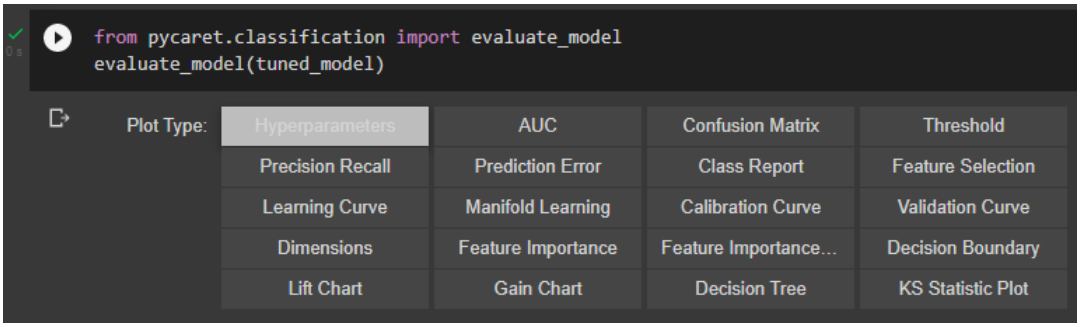

Figure 6: GUI of model evaluation, which lets us select a variety of plots/graphics

To recap, we have :

- *ClassPredictionError* plot, which tells us the quantity of errors per class, compared to the whole occurences of that class;
- *Boundaries* plot, based on the main 2 features (Missing and HBOND);
- $ClassificationReport$ plot, which gives us information on the classification per class, specifically on precision, recall, f1 and support;
- $ConfusionMatrix$ plot, gives us information on values predicted against actual values(respectively columns and rows);
- $FeatureImportance$ plot, simply how important are the features;
- And finally, the result of the function *evaluate_model*, which provide a GUI, to view all plots performed by the function *plot_model*, with this model.

And so, these were the all the interesting graphics/images I provided with *pycaret*, regarding the creation and analysis of the model.

10 Use of the Model

In this section, I will cover the final part: how the model is used to analyze and predict the datasets.

1. Predict_Model, a function of *pycaret* to predict the classification of the dataset with our model;
2. Save_Model, a function of *pycaret* to save the model as a file, precisely a .pkl file;
3. Load_Model, this function of *pycaret* reads the .pkl file and loads the model in a variable.

10.1 Save_Model

In this subsection I will cover the function Save_Model of *pycaret*:

```
from pycaret.internal.tabular import save_model
save_model(tuned_model, path+'saved_lr_model')
#save the model as a file named "saved_lr_model" with extension ".pkl"
#(so "saved_lr_model.pkl")
```

Notice that I saved my model *model* in $path + 'saved\_lr\_model'$, so the file *saved_lr_model.pkl* will be inside the folder designed by the *path* variable.

10.2 Load_Model

In this subsection I will cover the function Load_Model of *pycaret*:

```
from pycaret.internal.tabular import load_model
lmodel = load_model(path+'saved_lr_model')
```

These lines of code do the job of loading inside the variable *lmodel*, the model from the *saved_lr_model.pkl* inside the folder designed by *path* variable.

10.3 Predict_Model

In this subsection I will cover the function Predict_Model of *pycaret*:

```
from pycaret.internal.tabular import predict_model
test = pd.read_csv(path+'test.csv')
pmodel = predict_model(model, data = test, ml_usecase = 'classification')
```

Here first there is the function $read_csv$ of pandas, that reads the file *test.csv* in the folder designed by path, and put its content inside the variable *test*.

The variable *test* contains the dataset used for testing the model.

Finally, predict_model takes as input the model and some data, in this case, since I used it for testing the test dataset, I put as data the variable test: $data = test$.

Part IV
Code and markdown file "how to use"

Other than this report in pdf regarding the project, we have 2 python notebooks that work on colab, and a file .md that explains how to use those 2 python notebooks, and a folder containing the train.csv and test.csv datasets, that in this case are the same dataset.

The folder has to be put on the drive along with the 2 notebooks, since they work only on GoogleColab. I decided to have them used only with GoogleCOlab, because I was having trouble with *pycaret* package, and to avoid problems for the teacher I decided to use GoogleColab, that should give zero problems regarding packages.

So, we have 2 notebooks :

1. StructuralBioinformatics_TrainModel.ipynb
2. StructuralBioinformatics_TestModel.ipynb

The first one is to train the model, using the train dataset, be careful, because if there is a need to re-train the model, it will take around 1h and 1h and half. The second one is to test the model with a new, or even the same dataset, this notebook takes about 3 minutes, so there is no problem.

Either way, the markdown file will go into more detail to use these 2 python notebooks.