pulsar-candidates

July 16, 2020

1 Progetto di Programmazione di Applicazioni DataIntensive

- Candidato: Casadei Andrea (800898)
- Dataset: Pulsar Discovery

1.0.1 Quotes

- R. J. Lyon, B. W. Stappers, S. Cooper, J. M. Brooke, J. D. Knowles, Fifty Years of Pulsar Candidate Selection: From simple filters to a new principled real-time classification approach, Monthly Notices of the Royal Astronomical Society 459 (1), 1104-1123, DOI: 10.1093/mnras/stw656
- R. J. Lyon, HTRU2, DOI: 10.6084/m9.figshare.3080389.v1.

2 HTRU2 Data Set

`HTRU2` è un dataset che descrive un campione di stelle candidate come *Pulsar* (una rara tipologia di stelle di neutroni)

Ogni pulsar produce un segnale leggermente differente ad ogni rotazione, per questo un potenziale *'candidato'* viene mediato su diverse rotazioni

Il *Machine Learning* ha aiutato molto lo studio di queste particolari stelle in quanto la classificazione veniva fatta a mano

[26]:
```
%matplotlib inline
from collections import Counter
from imblearn.over_sampling import SMOTE
from sklearn import metrics
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Perceptron
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
```

```
from sklearn.svm import SVC
import io
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import requests
import seaborn as sns


pd.options.display.float_format = '{:.2f}'.format

PULS_URL = "https://gitlab.com/emrevoid/uni/dia/project/seintz/-/raw/master/
 ↪HTRU_2.csv"
response = requests.get(PULS_URL)
file_object = io.StringIO(response.content.decode('utf-8'))
pulsds = pd.read_csv(file_object, header=0, index_col=False, sep=r'\s*,\s*',␣
 ↪engine='python')

pulsds.head()
```

[26]:

	Profile_mean	Profile_stdev	Profile_skewness	Profile_kurtosis	DM_mean	DM_stdev	DM_skewness	DM_kurtosis	class
0	140.56	55.68	-0.23	-0.70	3.20	19.11	7.98	74.24	0
1	102.51	58.88	0.47	-0.52	1.68	14.86	10.58	127.39	0
2	103.02	39.34	0.32	1.05	3.12	21.74	7.74	63.17	0
3	136.75	57.18	-0.07	-0.64	3.64	20.96	6.90	53.59	0
4	88.73	40.67	0.60	1.12	1.18	11.47	14.27	252.57	0

Ogni candidato viene descritto da 8 variabili continue ed una singola classe

1. Mean of the integrated profile
2. Standard deviation of the integrated profile
3. Excess kurtosis of the integrated profile
4. Skewness of the integrated profile
5. Mean of the DM-SNR curve
6. Standard deviation of the DM-SNR curve.
7. Excess kurtosis of the DM-SNR curve
8. Skewness of the DM-SNR curve
9. Class

Definizioni: - Integrated Profile: profilo reale (cioè misurato tramite radiotelescopi) di una pulsar - DM-SNR curve: profilo teorico della pulsar basato su una curva

La variabile target è `class`, è binaria ed assume valore 1 se la stella in esame è una pulsar altrimenti 0

2.1 Distribuzione dei dati

[27]:
```
pulsds.shape
```

[27]:
```
(17898, 9)
```

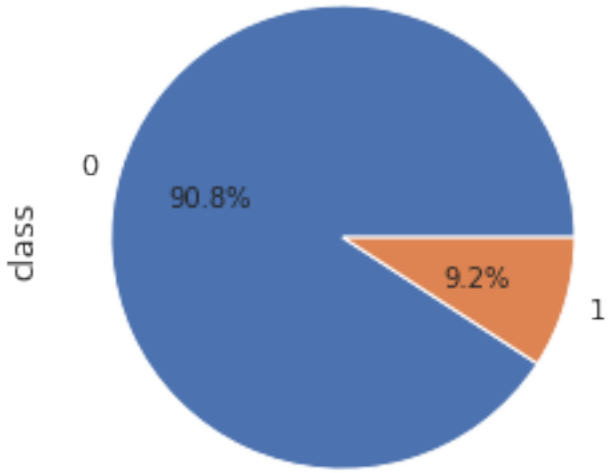
[28]:
```
pulsds["class"].value_counts()
```

[28]:
```
0    16259
1     1639
Name: class, dtype: int64
```

[29]:
```
pulsds["class"].value_counts(normalize=True)
```

[29]:
```
0   0.91
1   0.09
Name: class, dtype: float64
```

[30]:
```
pulsds["class"].value_counts().plot.pie(autopct='%1.1f%%');
```



Il nostro dataset è composto da circa 18000 righe e 9 feature (descritte nel capitolo precendente)

Come possiamo notare i dati sono molto sbilanciati, infatti solo il 9% risultano essere effettivamente delle pulsar, per via di questo sbilanciamento verrà fatto un bilanciamento della distribuzione

Questo sbilanciamento è dovuto al fatto che i raggi emissi dalle pulsar sono molto simili alle interferenze di onde radio presenti nell'universo, motivo per il quale sono difficili da distinguere

[31]:
```
pulsds.describe()
```

[31]:

	Profile_mean	Profile_stdev	Profile_skewness	Profile_kurtosis
count	17898.00	17898.00	17898.00	17898.00
mean	111.08	46.55	0.48	1.77
std	25.65	6.84	1.06	6.17
min	5.81	24.77	-1.88	-1.79
25%	100.93	42.38	0.03	-0.19
50%	115.08	46.95	0.22	0.20
75%	127.09	51.02	0.47	0.93
max	192.62	98.78	8.07	68.10

	DM_mean	DM_stdev	DM_skewness	DM_kurtosis	class
count	17898.00	17898.00	17898.00	17898.00	17898.00
mean	12.61	26.33	8.30	104.86	0.09
std	29.47	19.47	4.51	106.51	0.29
min	0.21	7.37	-3.14	-1.98	0.00
25%	1.92	14.44	5.78	34.96	0.00
50%	2.80	18.46	8.43	83.06	0.00
75%	5.46	28.43	10.70	139.31	0.00
max	223.39	110.64	34.54	1191.00	1.00

2.2 Preparazione del dataset

2.2.1 Gestione dei valori nulli

[32]:
```
pulsds.isna().sum()
```

[32]:
```
Profile_mean        0
Profile_stdev       0
Profile_skewness    0
Profile_kurtosis    0
DM_mean             0
DM_stdev            0
DM_skewness         0
DM_kurtosis         0
class               0
dtype: int64
```

Non essendoci valori nulli non abbiamo bisogno di effettuare alcuna operazione su di essi

2.3 Correlazione tra le feature

[33]:
```
def plot_correlation(dataset):
    cmap = sns.diverging_palette(220, 10, as_cmap=True)
    # Generate a mask for the upper triangle
    mask = np.zeros_like(dataset, dtype=np.bool)
    mask[np.triu_indices_from(mask)] = True

    # Set up the matplotlib figure
    f, ax = plt.subplots(figsize=(11, 9))
    # Draw the heatmap with the mask and correct aspect ratio
    sns.heatmap(dataset, mask=mask, cmap=cmap, vmax=.3, center=0,
                annot = True, square=True, linewidths=.5, cbar_kws={"shrink": .
 ↪5});


plot_correlation(pulsds.corr())
```

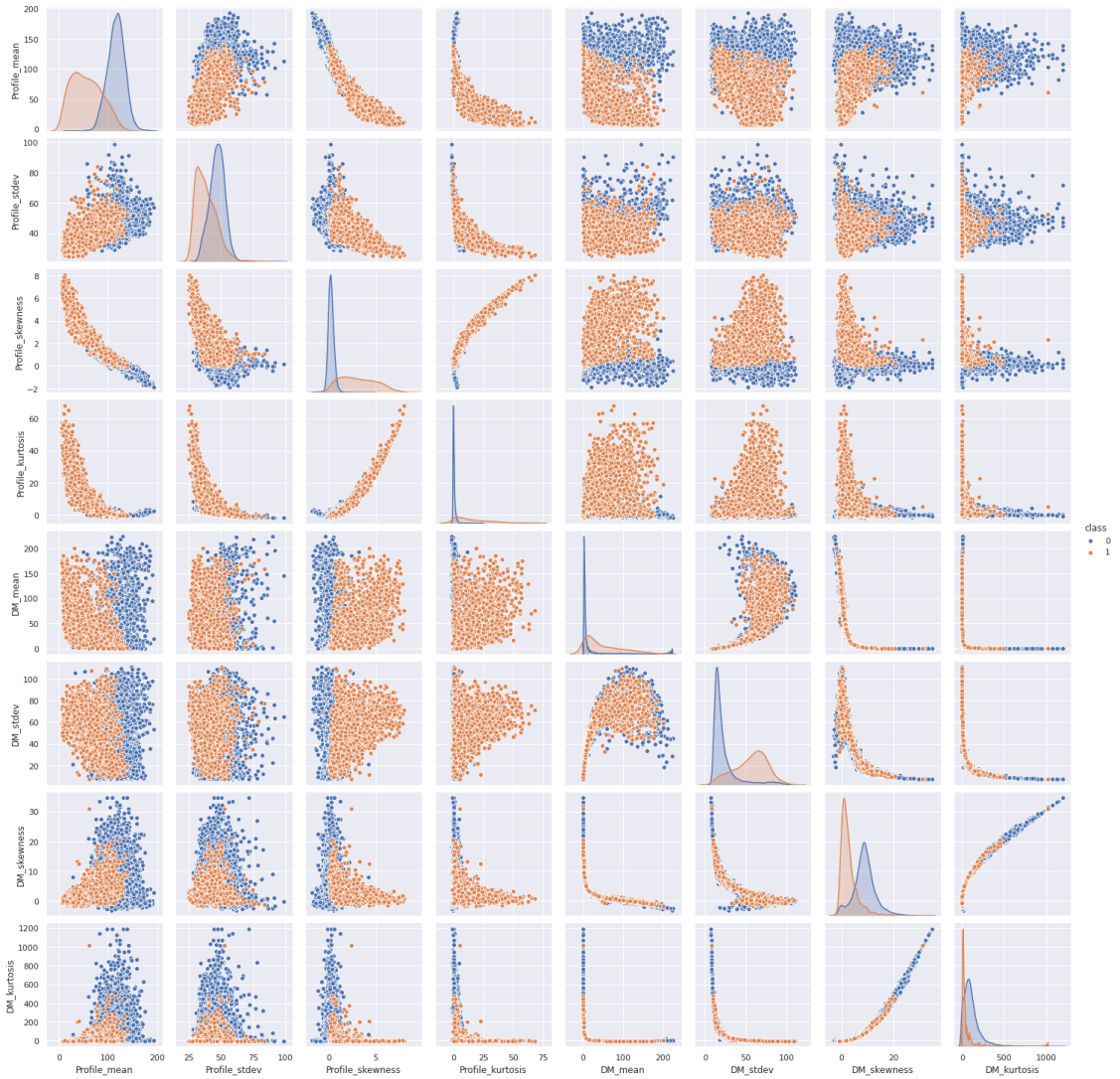
Dalla mappa di correlazione possiamo notare che la nostra variabile `class` è influenzata in modo particolare da `profile_skewness` e `profile_kurtosis`

Inoltre possiamo notare che `profile_kurtosis` è strettamente correlata da `profile_skewness`

La variabile `class` è inoltre influenzata, anche se minormente, da `DM_mean` e `DM_stdev`, che, come nel caso precedente, sono molto correlate tra loro

2.3.1 Correlazione a coppie

[34]:
```
sns.set();
sns.pairplot(data=pulsds, hue="class");
```



Dai grafici a dispersione si notano le correlazioni tra le varie feature del dataset, in particolare possiamo notare che alcune, come per esempio `Profile_mean` e `DM_mean`, sono lineari mentre altre come `DM_skewness` e `Profile_stdev` sono fortemente non lineari

I grafici nella diagonale principale ci mostrano la frequenza della feature con se stessa, per esempio dal primo grafico possiamo notare che le *pulsar* hanno una media dell'`integrated profile` molto maggiore rispetto alle *non-pulsar*

2.3.2 Divisione delle X e della y

[35]:
```
X = pulsds.drop('class', axis=1)
y = pulsds['class']
```

2.4 Risoluzione classi sbilanciate

Come già visto in precedenza le classi sono sbilanciate, utilizzeremo il metodo `SMOTE` di `imblearn` per effettuare le operazioni di bilanciamento

[36]:
```
X_resampled, y_resampled = SMOTE().fit_resample(X, y)

print(sorted(Counter(y_resampled).items()))
```

```
[(0, 16259), (1, 16259)]
```

[37]:
```
pulsds_oversampled = pd.concat([X_resampled, y_resampled], axis=1)
```

[38]:
```
pulsds_oversampled.shape
```

[38]:
```
(32518, 9)
```

[39]:
```
pulsds_oversampled["class"].value_counts()
```

[39]:
```
1    16259
0    16259
Name: class, dtype: int64
```

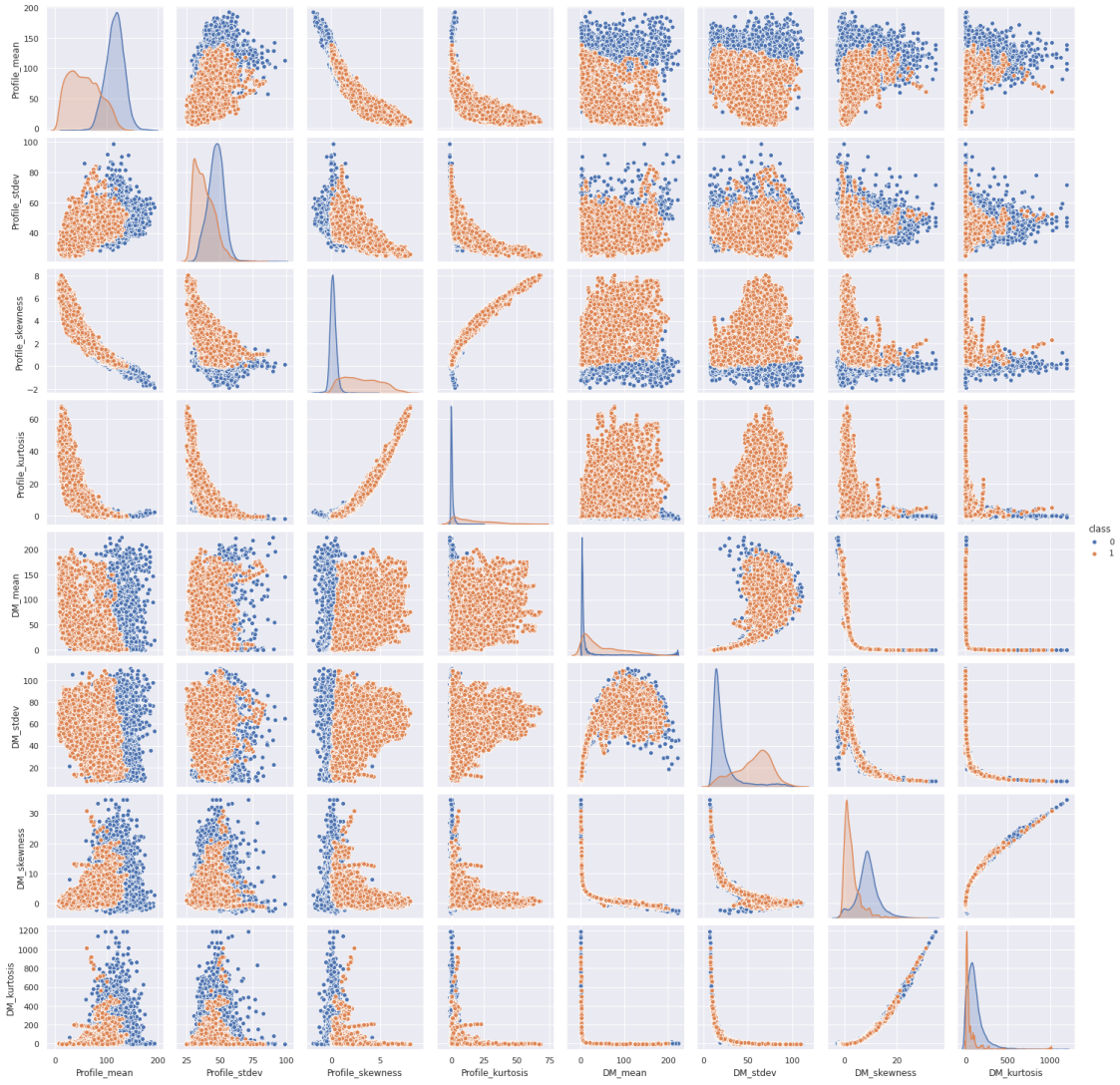
[40]:
```
pulsds_oversampled["class"].value_counts(normalize=True)
```

[40]:
```
1   0.50
0   0.50
Name: class, dtype: float64
```

Dopo aver eseguito le operazioni di bilanciamento possiamo notare che il numero di entry è stato raddoppiato e feature è rimasto invariato, ma il numero di entry appartenenti alle *pulsar* è aumentato notevolmente portando la distribuzione della classe del dataset ad un "50/50"

Il grafico qui sotto ci conferma che la correlazione tra le classi non ha subito enormi cambiamenti, a parte nei grafici della diagonale in cui si possono notare picchi più marcati

[41]:
```
sns.pairplot(data=pulsds_oversampled, hue="class");
```



2.5 Classificazione

2.5.1 Separazione del dataset

Separiamo il dataset in due parti, una di training e una di test

[42]:
```
X_train, X_val, y_train, y_val = train_test_split(
    X_resampled, y_resampled,
    test_size=1/3,
    random_state=42,
)
```

[43]:
```
skf = StratifiedKFold(3, shuffle=True, random_state=42)

for train, val in skf.split(X_train, y_train):
```

```
    print(y_train.iloc[val].value_counts())
```

```
1    3619
0    3607
Name: class, dtype: int64
1    3619
0    3607
Name: class, dtype: int64
1    3618
0    3608
Name: class, dtype: int64
```

2.6 Perceptron

Viene eseguito `perceptron` con GridSearch su standardizzazione e regolarizzazione (coefficiente a 5 valori nell'intervallo 10^{-2}, 10^{2})

2.6.1 balanced

[44]:
```
scale = [None, StandardScaler()]
grid = [
    {
        "std" : scale,
        "perceptron__penalty": ["l2", "l1", "elasticnet"],
        "perceptron__alpha": np.logspace(-2, 2, 5)
    },
]

model_perceptron = Pipeline([
    ('std', None),
    ('perceptron', Perceptron(random_state=42))
])

gs = GridSearchCV(model_perceptron, grid, cv=skf)
gs.fit(X_train, y_train)

print('Accuracy on train {:.2f}%'.format(gs.score(X_train, y_train)*100))
print('Accuracy on val {:.2f}%'.format(gs.score(X_val, y_val)*100))
print(gs.best_params_)
```

```
Accuracy on train 89.48%
Accuracy on val 89.82%
{'perceptron__alpha': 0.1, 'perceptron__penalty': 'l2', 'std': StandardScaler()}
```

2.6.2 unbalanced

[45]:
```
X_train_unb, X_val_unb, y_train_unb, y_val_unb = train_test_split(
    X, y,
    test_size=1/3,
    random_state=42,
)
```

[46]:
```
scale = [None, StandardScaler()]
grid = [
    {
        "std" : scale,
        "perceptron__penalty": ["l2", "l1", "elasticnet"],
        "perceptron__alpha": np.logspace(-2, 2, 5)
    },
]

model_perceptron_unb = Pipeline([
    ('std', None),
    ('perceptron', Perceptron(random_state=42))
])

gs_unb = GridSearchCV(model_perceptron_unb, grid, cv=skf)
gs_unb.fit(X_train_unb, y_train_unb);

print('Accuracy on train {:.2f}%'.format(gs_unb.score(X_train_unb,␣
 ↪y_train_unb)*100))
print('Accuracy on val {:.2f}%'.format(gs_unb.score(X_val_unb, y_val_unb)*100))
print(gs_unb.best_params_)
```

```
Accuracy on train 96.77%
Accuracy on val 96.83%
{'perceptron__alpha': 0.01, 'perceptron__penalty': 'l2', 'std':
StandardScaler()}
```

2.7 Confusion Matrix

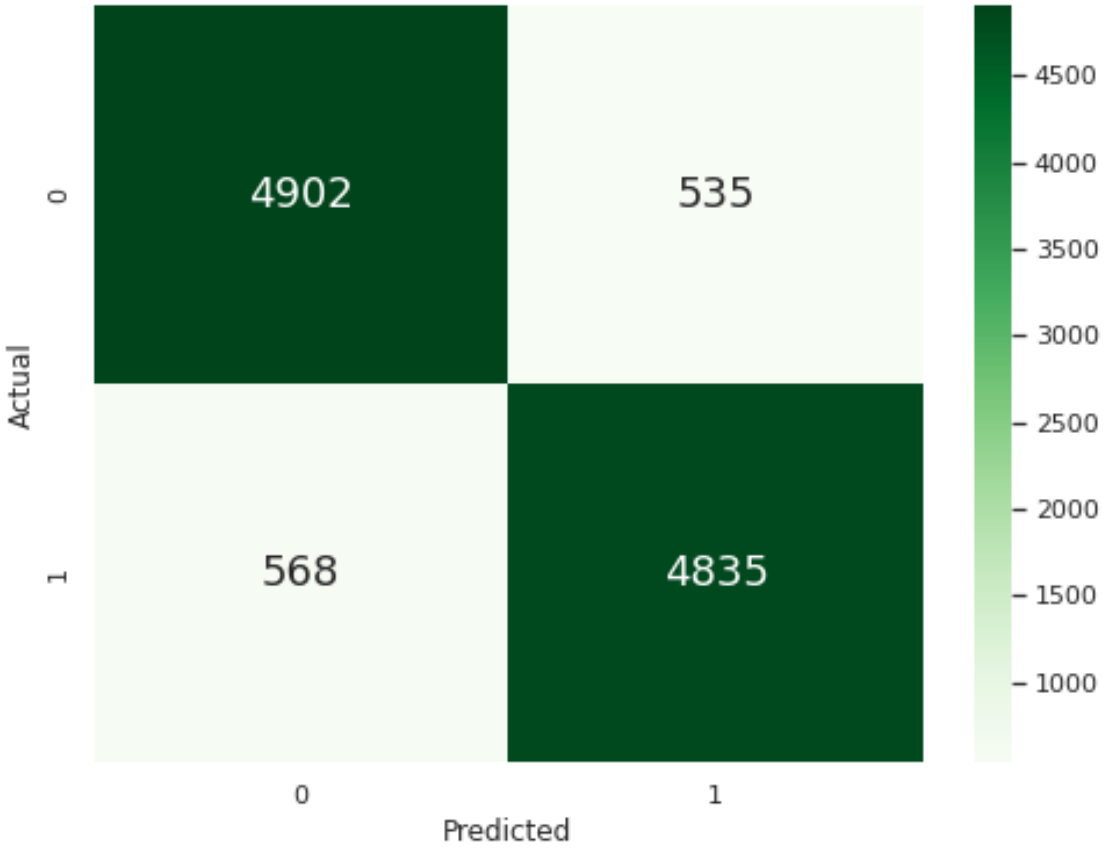
Viene creata una funzione per generare la `confusion matrix` in modo tale da non avere codice duplicato, dopodichè calcoliamo la `confusion matrix` sia per il dataset bilanciato sia per quello sbilanciato

[47]:
```
def plot_confusion_matrix(model, X_val, y_val, y_train, label_X='Actual',␣
 ↪label_y='Predicted'):
    y_pred = model.predict(X_val)
    cm = confusion_matrix(y_val, y_pred)
    cm_df = pd.DataFrame(cm, columns=np.unique(y_train), index = np.
 ↪unique(y_train))
    cm_df.index.name = label_X
```

```
    cm_df.columns.name = label_y
    plt.figure(figsize = (8,6));
    sns.heatmap(cm_df, cmap="Greens", annot=True, annot_kws={"size": 18},␣
↪fmt='d')
```

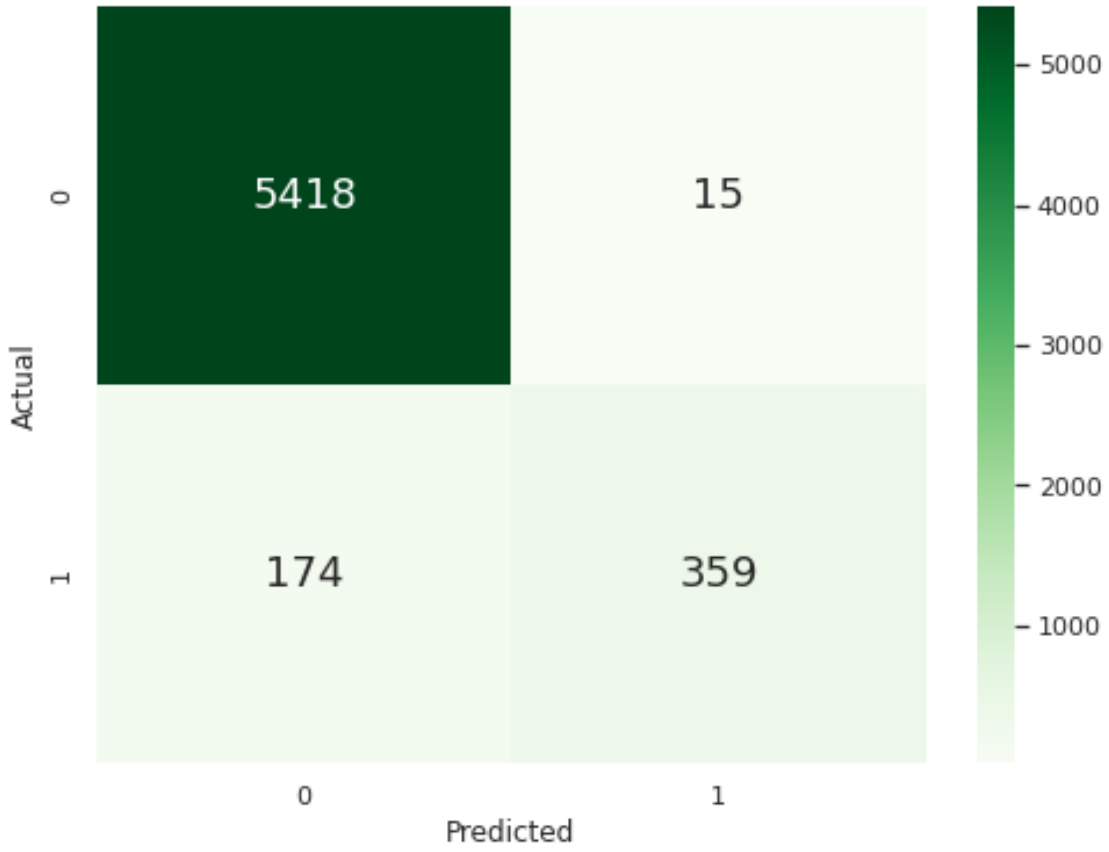
2.7.1 balanced

[48]:
```
plot_confusion_matrix(gs, X_val, y_val, y_train)
```



2.7.2 unbalanced

[49]:
```
plot_confusion_matrix(gs_unb, X_val_unb, y_val_unb, y_train_unb)
```



2.8 Precison & Recall

2.8.1 balanced

[53]:
```
y_pred = gs.predict(X_val)
print(classification_report(y_val, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.90      0.90      5437
           1       0.90      0.89      0.90      5403

    accuracy                           0.90     10840
   macro avg       0.90      0.90      0.90     10840
weighted avg       0.90      0.90      0.90     10840
```

2.8.2 unbalanced

[59]:
```
y_pred_unb = gs_unb.predict(X_val_unb)
print(classification_report(y_val_unb, y_pred_unb))
```

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.98      5433
           1       0.96      0.67      0.79       533

    accuracy                           0.97      5966
   macro avg       0.96      0.84      0.89      5966
weighted avg       0.97      0.97      0.97      5966
```

2.9 Confidence Range

Definiamo delle funzioni per il calcolo dell'intervallo di confidenza, con probabilità di default del 95%

[60]:
```
def conf_interval(a, N, Z=1.96):
    c = (2 * N * a + Z**2) / (2 * (N + Z**2))
    d = Z * np.sqrt(Z**2 + 4*N*a - 4*N*a**2) / (2 * (N + Z**2))
    return c - d, c + d

def model_conf_interval(model, X, y, level=0.95):
    from scipy.stats import norm
    a = model.score(X, y)
    N = len(X)
    Z = norm.ppf((1 + level) / 2)
    return conf_interval(a, N, Z)
```

2.9.1 balanced

[61]:
```
model_conf_interval(gs, X_val, y_val)
```

[61]:
```
(0.8924142031834277, 0.9037981015415602)
```

2.9.2 unbalanced

[62]:
```
model_conf_interval(gs_unb, X_val, y_val)
```

[62]:
```
(0.8281379485401695, 0.8421042370911963)
```

Possiamo notare che il modello di perceptron sulle classi sbilanciate ha una precisione minore (seppure molto simile)

2.10 Regressione Logistica

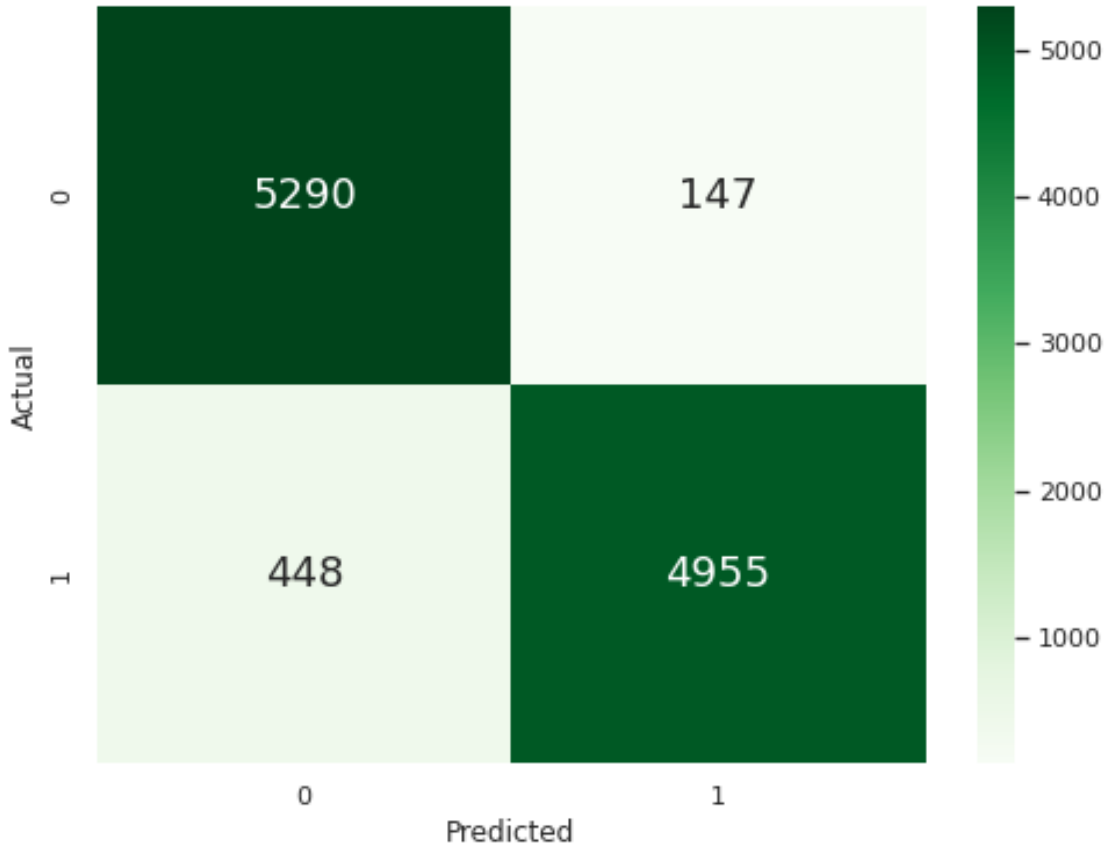
[63]:
```
scale = [None, StandardScaler()]
grid = [
    {
        "std" : scale,
        "logreg__penalty": ["l1"],
        "logreg__C": np.logspace(-2, 2, 5)
    },
]

model_logreg = Pipeline([
    ('std', None),
    ('logreg', LogisticRegression(solver="saga", dual=False, random_state=42,␣
 ↪max_iter=10000))
])

gs_logreg = GridSearchCV(model_logreg, grid, cv=skf)
gs_logreg.fit(X_train, y_train)

print('Accuracy on train {:.2f}%'.format(gs_logreg.score(X_train, y_train)*100))
print('Accuracy on val {:.2f}%'.format(gs_logreg.score(X_val, y_val)*100))
print(gs_logreg.best_params_)
```

```
Accuracy on train 94.30%
Accuracy on val 94.51%
{'logreg__C': 1.0, 'logreg__penalty': 'l1', 'std': StandardScaler()}
```

[64]:
```
y_pred_logreg = gs_logreg.predict(X_val)
print(classification_report(y_val, y_pred_logreg))
```

```
              precision    recall  f1-score   support

           0       0.92      0.97      0.95      5437
           1       0.97      0.92      0.94      5403

    accuracy                           0.95     10840
   macro avg       0.95      0.95      0.95     10840
weighted avg       0.95      0.95      0.95     10840
```

[65]:
```
plot_confusion_matrix(gs_logreg, X_val, y_val, y_train)
```



[66]:
```
y_pred_logreg = gs_logreg.predict(X_val)
print(classification_report(y_val, y_pred_logreg))
```

```
              precision    recall  f1-score   support

           0       0.92      0.97      0.95      5437
           1       0.97      0.92      0.94      5403

    accuracy                           0.95     10840
   macro avg       0.95      0.95      0.95     10840
weighted avg       0.95      0.95      0.95     10840
```

[67]:
```
model_conf_interval(gs_logreg, X_val, y_val)
```

[67]: (0.9406632369044758, 0.9492428020804828)

2.11 SVM

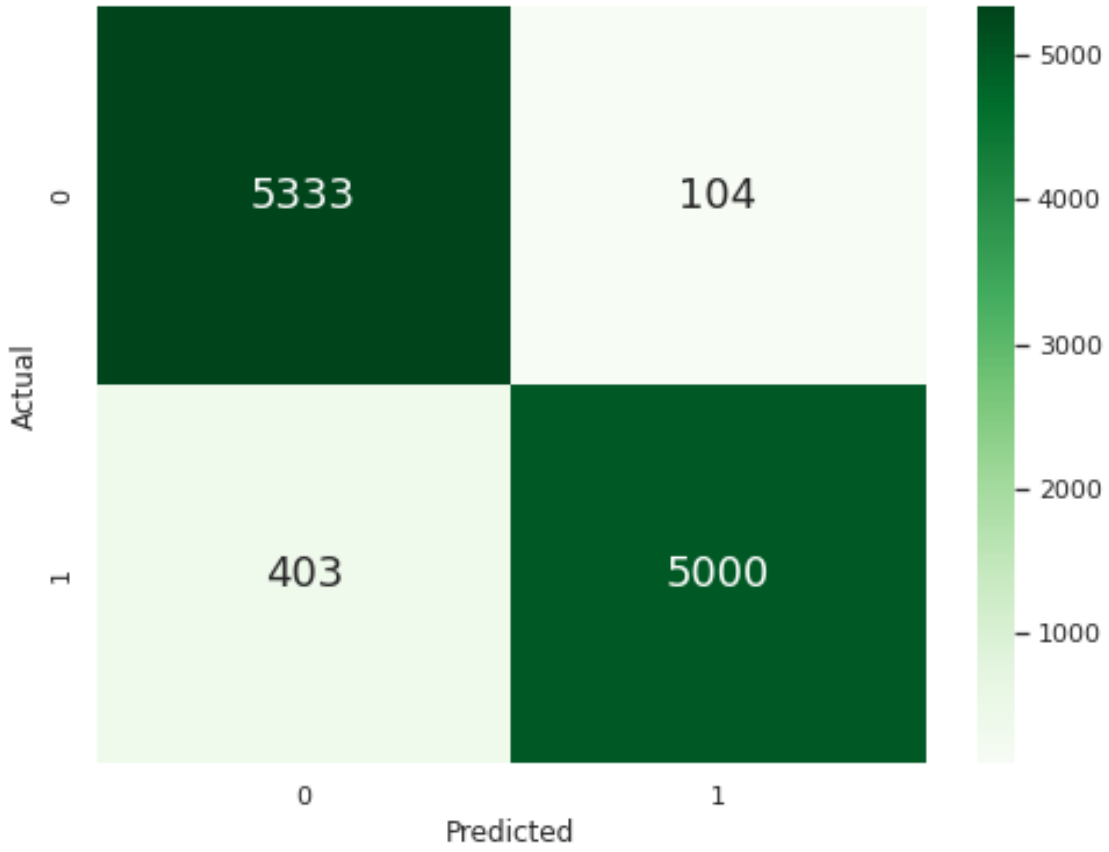
[68]:
```
scale = [None, StandardScaler()]
grid = [
    {
        "std" : scale,
        "svm__C": np.logspace(-2, 2, 5)
    },
]

model_svm = Pipeline([
    ('std', None),
    ('svm', SVC())
])

gs_svm = GridSearchCV(model_svm, grid, cv=skf)
gs_svm.fit(X_train, y_train)

print('Accuracy on train {:.2f}%'.format(gs_svm.score(X_train, y_train)*100))
print('Accuracy on val {:.2f}%'.format(gs_svm.score(X_val, y_val)*100))
print(gs_svm.best_params_)
```

```
Accuracy on train 95.24%
Accuracy on val 95.32%
{'std': StandardScaler(), 'svm__C': 100.0}
```

[69]:
```
y_pred_svm = gs_svm.predict(X_val)
print(classification_report(y_val, y_pred_svm))
```

```
              precision    recall  f1-score   support

           0       0.93      0.98      0.95      5437
           1       0.98      0.93      0.95      5403

    accuracy                           0.95     10840
   macro avg       0.95      0.95      0.95     10840
weighted avg       0.95      0.95      0.95     10840
```

[70]:
```
plot_confusion_matrix(gs_svm, X_val, y_val, y_train)
```



[71]:
```
y_pred_svm = gs_svm.predict(X_val)
print(classification_report(y_val, y_pred_svm))
```

```
              precision    recall  f1-score   support

           0       0.93      0.98      0.95      5437
           1       0.98      0.93      0.95      5403

    accuracy                           0.95     10840
   macro avg       0.95      0.95      0.95     10840
weighted avg       0.95      0.95      0.95     10840
```

[72]:
```
model_conf_interval(gs_svm, X_val, y_val)
```

[72]: (0.9490908313072256, 0.9570456183369406)

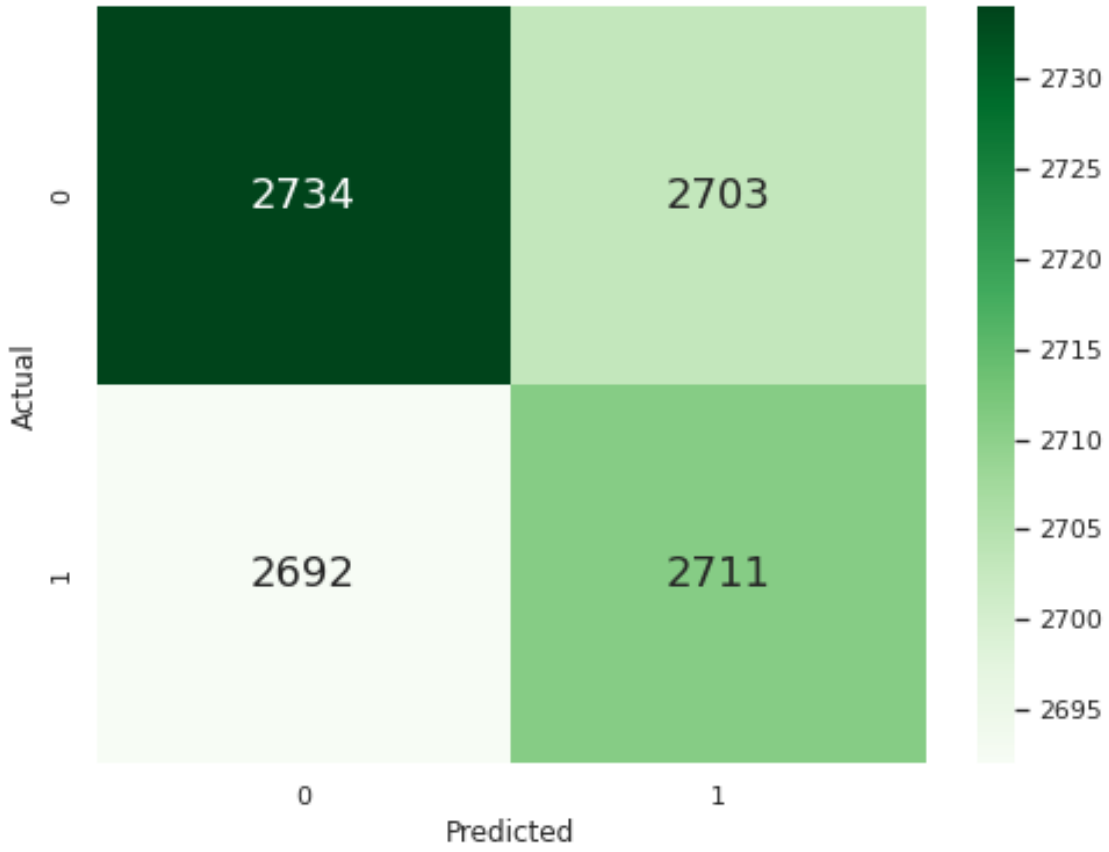
2.12 Modello Random

Generiamo un modello randomico da confrontare con quelli già eseguiti, possibile grazie al modulo `DummyClassifier` con `strategy=uniform` della libreria `scikit-learn`

[73]:
```
scale = [None, StandardScaler()]
grid = [
    {
        "std" : scale,
    },
]

model_random = Pipeline([
    ('std', None),
    ('rnd', DummyClassifier(strategy='uniform', random_state=42))
], )

gs_random = GridSearchCV(model_random, grid, cv=skf)
gs_random.fit(X_train, y_train);

print('Accuracy on train {:.2f}%'.format(gs_random.score(X_train, y_train)*100))
print('Accuracy on val {:.2f}%'.format(gs_random.score(X_val, y_val)*100))
print(gs_random.best_params_)
```

```
Accuracy on train 50.20%
Accuracy on val 50.23%
{'std': None}
```

[74]:
```
plot_confusion_matrix(gs_random, X_val, y_val, y_train)
```



[75]:
```
y_pred_random = gs_random.predict(X_val)
print(classification_report(y_val, y_pred_random))
```

```
              precision    recall  f1-score   support

           0       0.50      0.50      0.50      5437
           1       0.50      0.50      0.50      5403

    accuracy                           0.50     10840
   macro avg       0.50      0.50      0.50     10840
weighted avg       0.50      0.50      0.50     10840
```

[76]:
```
model_conf_interval(gs_random, X_val, y_val)
```

[76]: (0.4928947567448444, 0.5117161553742487)

2.13 Confronto tra i modelli

[77]:
```
def diff_interval(a1, a2, N1, N2, Z):
    sigma1 = a1 * (1 - a1) / N1
    sigma2 = a2 * (1 - a2) / N2
    sigma = np.sqrt(sigma1 + sigma2)
    diff = abs(a1 - a2)
    Z_sigma = Z * sigma
    return (diff - Z_sigma, diff + Z_sigma)


def model_diff_interval(m1, m2, X, y, level=0.95):
    from scipy.stats import norm
    a1 = m1.score(X, y)
    a2 = m2.score(X, y)
    N1 = len(X)
    N2 = len(X)
    Z = norm.ppf((1 + level) / 2)
    return diff_interval(a1, a2, N1, N2, Z)
```

[78]:
```
model_diff_interval(gs, gs_unb, X_val, y_val)
```

[78]:
```
(0.053998643636976484, 0.07201611651062512)
```

[79]:
```
model_diff_interval(gs, gs_logreg, X_val, y_val)
```

[79]:
```
(0.039737899201385346, 0.05398903806798741)
```

[80]:
```
model_diff_interval(gs, gs_svm, X_val, y_val)
```

[80]:
```
(0.04803969606252485, 0.06192340356847152)
```

[81]:
```
model_diff_interval(gs, gs_random, X_val, y_val)
```

[81]:
```
(0.38494175484075516, 0.40694016397843297)
```

[82]:
```
model_diff_interval(gs_logreg, gs_svm, X_val, y_val)
```

[82]:
```
(0.00227142733635392, 0.013964735025269698)
```

[83]:
```
model_diff_interval(gs_logreg, gs_random, X_val, y_val)
```

[83]:
```
(0.4324614821971627, 0.4531473738913982)
```

[84]:
```
model_diff_interval(gs_svm, gs_random, X_val, y_val)
```

[84]:
```
(0.44070526289473066, 0.46113975555545383)
```

Da questo confronto si evince che il modello migliore per classificare il nostro dataset è **Perceptron** con un accuratezza del ~**96%**, seppure con una precisione inferiore rispetto agli altri metodi (che nonostante hanno un accuratezza molto simile ma una precisione maggiore)

I modelli con miglior accuratezza sono: 1. Perceptron, 96% 2. SVM, 95% 3. Regressione Logistica, 94%

Grazie a questi modelli possiamo classificare le candidate pulsar, possiamo inoltre scartare quelle che non raggiungono certe caratteristiche e concentrarci su quelle che invece sono molto vicine dall'essere pulsar

Grazie a questi modelli viene ridotto notevolmente il lavoro manuale che l'uomo deve svolgere in quanto si registrano un numero di candidati sempre maggiore.