S5 - HOPE SVM

December 15, 2020

1 Import data from DB.

[1]:
```
import pandas as pd
import numpy as np
```

[2]:
```
dfOrg = pd.read_csv('hope_dataset_cleaned.csv')

print(dfOrg.shape[0])
```

```
1243
```

[3]:
```
dfOrg.head(10)
```

[3]:
```
  pedido.data.attributes.age pedido.data.attributes.diagnostic_main  \
0                       75.0                     FISTULA PERITONEAL
1                       75.0                     FISTULA PERITONEAL
2                       75.0                     FISTULA PERITONEAL
3                       75.0                     FISTULA PERITONEAL
4                       75.0                     FISTULA PERITONEAL
5                       75.0                     FISTULA PERITONEAL
6                       75.0                     FISTULA PERITONEAL
7                       75.0                     FISTULA PERITONEAL
8                       75.0                     FISTULA PERITONEAL
9                       75.0                     FISTULA PERITONEAL

 pedido.data.attributes.gender  articulo  respuesta.articlesRevisedYear  \
0                         male  27395425                           2018
1                         male  28560554                           2018
2                         male  28641726                           2017
3                         male  26245344                           2016
4                         male  28942543                           2018
5                         male  24782153                           2014
6                         male  28002229                           2018
7                         male  27505109                           2017
8                         male  24850546                           2015
9                         male  29371050                           2019
```

```
   respuesta.articlesRevisedMonth  \
0                               1
1                               4
2                              12
3                              12
4                               6
5                               6
6                               9
7                               4
8                               1
9                               4

                                 respuesta.pubmed_keys  utilidad
0  Abdomen,Adenocarcinoma,Antiemetics,Blood Cultu…       1.0
1  Abdomen,Adenocarcinoma,Antiemetics,Blood Cultu…       NaN
2  Abdomen,Adenocarcinoma,Antiemetics,Blood Cultu…       NaN
3  Abdomen,Adenocarcinoma,Antiemetics,Blood Cultu…       NaN
4  Abdomen,Adenocarcinoma,Antiemetics,Blood Cultu…       NaN
5  Abdomen,Adenocarcinoma,Antiemetics,Blood Cultu…       NaN
6  Abdomen,Adenocarcinoma,Antiemetics,Blood Cultu…       NaN
7  Abdomen,Adenocarcinoma,Antiemetics,Blood Cultu…       NaN
8  Abdomen,Adenocarcinoma,Antiemetics,Blood Cultu…       NaN
9  Abdomen,Adenocarcinoma,Antiemetics,Blood Cultu…       NaN
```

2 Transform (factorice) from Categories to continuous atributes

Transform 'pedido.data.attributes.diagnostic_main' atribute

[4]:
```
dataDiagnosticMain, categoriesDiagnosticMain = pd.factorize(dfOrg['pedido.data.
 ↪attributes.diagnostic_main'])

dfOrg['pedido.data.attributes.diagnostic_main'] = dataDiagnosticMain
```

Transform 'gender' atribute

[5]:
```
dataGender, categoriesGender = pd.factorize(dfOrg['pedido.data.attributes.
 ↪gender'])

dfOrg['pedido.data.attributes.gender'] = dataGender
```

Transform 'respuesta.pubmed_keys' atribute

[6]:
```
categoriesORGPubMedKeys = dfOrg['respuesta.pubmed_keys'].value_counts()

print("total: " + str(categoriesORGPubMedKeys.size))
```

```
total: 80
```

[7]:
```
dataPubMedKeys, categoriesPubMedKeys = pd.factorize(dfOrg['respuesta.
↪pubmed_keys'])

dfOrg['respuesta.pubmed_keys'] = dataPubMedKeys
```

[8]:
```
dfOrg.head(10)
```

[8]:
```
   pedido.data.attributes.age  pedido.data.attributes.diagnostic_main  \
0                        75.0                                        0
1                        75.0                                        0
2                        75.0                                        0
3                        75.0                                        0
4                        75.0                                        0
5                        75.0                                        0
6                        75.0                                        0
7                        75.0                                        0
8                        75.0                                        0
9                        75.0                                        0

   pedido.data.attributes.gender  articulo  respuesta.articlesRevisedYear  \
0                              0  27395425                           2018
1                              0  28560554                           2018
2                              0  28641726                           2017
3                              0  26245344                           2016
4                              0  28942543                           2018
5                              0  24782153                           2014
6                              0  28002229                           2018
7                              0  27505109                           2017
8                              0  24850546                           2015
9                              0  29371050                           2019

   respuesta.articlesRevisedMonth  respuesta.pubmed_keys  utilidad
0                               1                      0       1.0
1                               4                      0       NaN
2                              12                      0       NaN
3                              12                      0       NaN
4                               6                      0       NaN
5                               6                      0       NaN
6                               9                      0       NaN
7                               4                      0       NaN
8                               1                      0       NaN
9                               4                      0       NaN
```

[9]:
```
print("age NaN => " + str(dfOrg[pd.isnull(dfOrg['pedido.data.attributes.age'])].
↪shape[0]))
print("diagnostic_main NaN => " + str(dfOrg[pd.isnull(dfOrg['pedido.data.
↪attributes.diagnostic_main'])].shape[0]))
```

```
print("gender NaN => " + str(dfOrg[pd.isnull(dfOrg['pedido.data.attributes.
↪gender'])].shape[0]))
print("articulo NaN => " + str(dfOrg[pd.isnull(dfOrg['articulo'])].shape[0]))
print("articlesRevisedYear NaN => " + str(dfOrg[pd.isnull(dfOrg['respuesta.
↪articlesRevisedYear'])].shape[0]))
print("articlesRevisedMonth NaN => " + str(dfOrg[pd.isnull(dfOrg['respuesta.
↪articlesRevisedMonth'])].shape[0]))
print("pubmed_keys NaN => " + str(dfOrg[pd.isnull(dfOrg['respuesta.
↪pubmed_keys'])].shape[0]))
print("utilidad NaN => " + str(dfOrg[pd.isnull(dfOrg['utilidad'])].shape[0]))
```

```
age NaN => 10
diagnostic_main NaN => 0
gender NaN => 0
articulo NaN => 0
articlesRevisedYear NaN => 0
articlesRevisedMonth NaN => 0
pubmed_keys NaN => 0
utilidad NaN => 1192
```

Remove row with age eq NaN

[10]:
```
dfOrg = dfOrg[pd.notnull(dfOrg['pedido.data.attributes.age'])]
```

3 Standardize the Data

Choosed "age", "diagnostic_main", "year", "pubmed_keys" and "articulo" attributes (based on PCA_V2 study)

[11]:
```
from sklearn.preprocessing import StandardScaler

features = ["pedido.data.attributes.age",
    "pedido.data.attributes.diagnostic_main",
    "respuesta.articlesRevisedYear",
    "respuesta.pubmed_keys",
    "articulo"]

# Separating out the features
x = dfOrg.loc[:, features].values

featuresTransformed = StandardScaler().fit_transform(x)

dfStandarized = pd.DataFrame(featuresTransformed, index=dfOrg.index,␣
↪columns=features)
dfStandarized['utilidad'] = dfOrg['utilidad']

dfStandarized
```

[11]:

```
      pedido.data.attributes.age  pedido.data.attributes.diagnostic_main  \
0                       1.443474                               -1.360638
1                       1.443474                               -1.360638
2                       1.443474                               -1.360638
3                       1.443474                               -1.360638
4                       1.443474                               -1.360638
…                              …                                       …
1238                   -0.429381                               -0.580827
1239                   -0.429381                               -0.580827
1240                   -0.429381                               -0.580827
1241                   -0.429381                               -0.580827
1242                   -0.429381                               -0.580827

      respuesta.articlesRevisedYear  respuesta.pubmed_keys  articulo  utilidad
0                          0.643671              -1.650220 -0.221939       1.0
1                          0.643671              -1.650220  0.137839       NaN
2                          0.224418              -1.650220  0.162904       NaN
3                         -0.194835              -1.650220 -0.577070       NaN
4                          0.643671              -1.650220  0.255793       NaN
…                                 …                      …         …         …
1238                      -0.194835               1.520816  0.574852       NaN
1239                       1.062924               1.520816 -0.540973       NaN
1240                      -0.614089               1.520816  0.801912       NaN
1241                       1.062924               1.520816 -0.056202       NaN
1242                      -0.614089               1.520816 -2.782199       NaN

[1233 rows x 6 columns]
```

4 Separe data by utilidad is defined

[12]:

```
dfDataSetComplete = dfStandarized[pd.notnull(dfStandarized['utilidad'])]

print(dfDataSetComplete.shape[0])

dfDataSetToPredict = dfStandarized[pd.isnull(dfStandarized['utilidad'])]

print(dfDataSetToPredict.shape[0])
```

```
51
1182
```

5 SVM

We check the number of results

[13]:

```
dfDataSetComplete.groupby('utilidad').size()
```

[13]:
```
utilidad
0.0    21
1.0    30
dtype: int64
```

Separe “utilidad” atribute from data to train

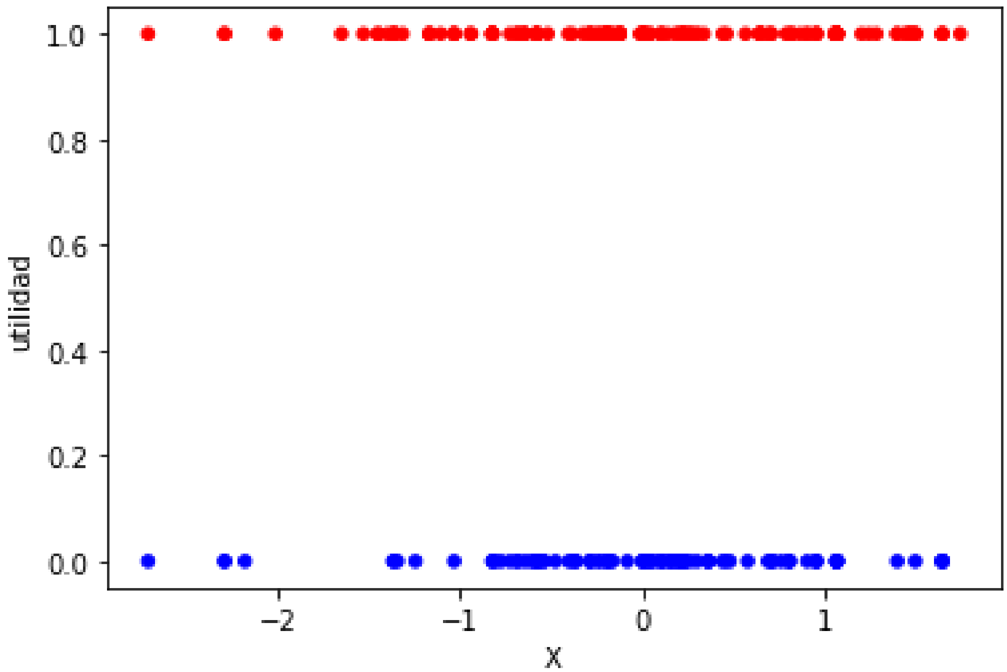
[14]:
```
X = np.array(dfDataSetComplete.drop(['utilidad'],1))
y = np.array(dfDataSetComplete['utilidad'])
X.shape
```

[14]:
```
(51, 5)
```

[15]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

[16]:
```
ax = dfDataSetComplete.plot.scatter(x="pedido.data.attributes.age",␣
 ↪y="utilidad", c=dfDataSetComplete.utilidad.map({0:'b', 1:'r'}))
dfDataSetComplete.plot.scatter(x="pedido.data.attributes.diagnostic_main",␣
 ↪y="utilidad", c=dfDataSetComplete.utilidad.map({0:'b', 1:'r'}), ax=ax)
dfDataSetComplete.plot.scatter(x="respuesta.articlesRevisedYear", y="utilidad",␣
 ↪c=dfDataSetComplete.utilidad.map({0:'b', 1:'r'}), ax=ax)
dfDataSetComplete.plot.scatter(x="respuesta.pubmed_keys", y="utilidad",␣
 ↪c=dfDataSetComplete.utilidad.map({0:'b', 1:'r'}), ax=ax)
dfDataSetComplete.plot.scatter(x="articulo", y="utilidad", c=dfDataSetComplete.
 ↪utilidad.map({0:'b', 1:'r'}), ax=ax)
ax.set_xlabel('X')
```

[16]:
```
Text(0.5, 0, 'X')
```



A simple vista no podemos crear un hiperplano lineal (división de valores) que nos ayude a poder clasificar los valores del campo utilidad en base los atributos del data set, ya que los resultados del campo "utilidad" están distribuidos sobre todo el plano de X. Exploraremos con los kernel methods, que método nos ayuda a poder crear el hiperplano más optimo para la clasificación.

6 Exploring Hiper Parameters

· C: El valor de penalización de los errores en la clasificación. Indica el compromiso entre obtener el hiperplano con el margen más grande posible y clasificar el máximo número de ejemplos correctamente. Probaremos valores aleatorios distribuidos uniformemente entre 1 y 500.

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

[17]:
```
from sklearn.model_selection import RandomizedSearchCV
from sklearn import svm
from scipy.stats import uniform as sp_rand
from time import time

clf = svm.SVC()

kernels = ['poly', 'rbf', 'sigmoid']

param_dist = {
    "C":     sp_rand(loc=1, scale=500),
```

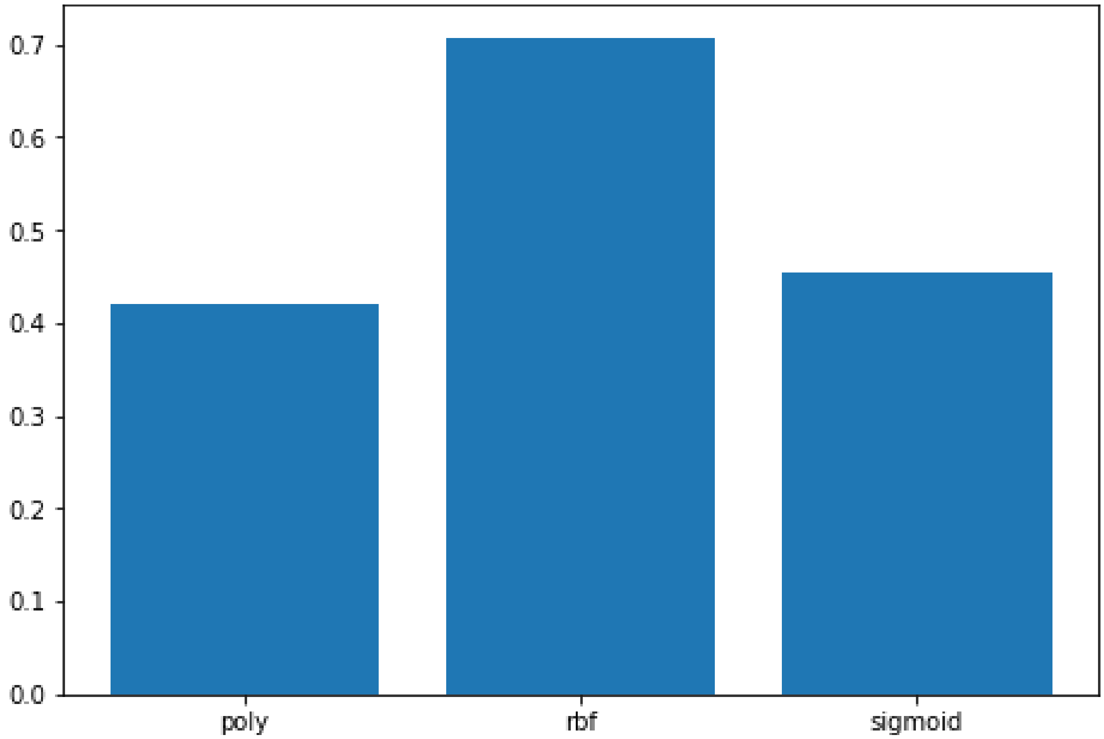
```python
    "gamma": sp_rand(loc=1e-3, scale=1e3)
}

best_score = []

for k in kernels:
    param_dist['kernel'] = [k]
    n_iter_search = 10
    random_search = RandomizedSearchCV(clf, param_distributions=param_dist,␣
 ↪n_iter=n_iter_search, cv=4)

    start = time()
    random_search.fit(X_train, y_train)
    end = time()
    print("El entrenamiento a durado {} segundos".format(end - start))

    means = random_search.cv_results_["mean_test_score"]
    stds = random_search.cv_results_["std_test_score"]
    params = random_search.cv_results_['params']
    ranks = random_search.cv_results_['rank_test_score']
    best_score.append({'kernel':k,'score':random_search.best_score_, 'params':␣
 ↪random_search.best_params_})

    for rank, mean, std, pms in zip(ranks, means, stds, params):
        print("{}) Precisión media: {:.2f} +/- {:.2f} con parámetros {}".
 ↪format(rank, mean*100, std*100, pms))
```

```
El entrenamiento a durado 0.11058187484741211 segundos
1) Precisión media: 41.94 +/- 6.10 con parámetros {'C': 53.821075955066966,
'gamma': 425.9568020864903, 'kernel': 'poly'}
1) Precisión media: 41.94 +/- 6.10 con parámetros {'C': 67.54459482731284,
'gamma': 296.2062003214801, 'kernel': 'poly'}
1) Precisión media: 41.94 +/- 6.10 con parámetros {'C': 258.8045148248579,
'gamma': 789.4708117751253, 'kernel': 'poly'}
1) Precisión media: 41.94 +/- 6.10 con parámetros {'C': 298.11432709769304,
'gamma': 899.3173986957249, 'kernel': 'poly'}
1) Precisión media: 41.94 +/- 6.10 con parámetros {'C': 162.96844404448342,
'gamma': 135.31069594836018, 'kernel': 'poly'}
1) Precisión media: 41.94 +/- 6.10 con parámetros {'C': 398.06387221894306,
'gamma': 292.3118987051191, 'kernel': 'poly'}
1) Precisión media: 41.94 +/- 6.10 con parámetros {'C': 417.60824273412493,
'gamma': 183.70307024299845, 'kernel': 'poly'}
1) Precisión media: 41.94 +/- 6.10 con parámetros {'C': 265.265522466685,
'gamma': 49.707115701978026, 'kernel': 'poly'}
1) Precisión media: 41.94 +/- 6.10 con parámetros {'C': 78.92440360685087,
'gamma': 5.205532851343702, 'kernel': 'poly'}
```

```
1) Precisión media: 41.94 +/- 6.10 con parámetros {'C': 197.21348727675948,
'gamma': 355.55660778888614, 'kernel': 'poly'}
El entrenamiento a durado 0.07101178169250488 segundos
1) Precisión media: 70.83 +/- 9.57 con parámetros {'C': 89.593529467132,
'gamma': 484.37015474111286, 'kernel': 'rbf'}
1) Precisión media: 70.83 +/- 9.57 con parámetros {'C': 497.59938994676645,
'gamma': 63.60278434380348, 'kernel': 'rbf'}
1) Precisión media: 70.83 +/- 9.57 con parámetros {'C': 483.27875318239387,
'gamma': 590.2235607411753, 'kernel': 'rbf'}
1) Precisión media: 70.83 +/- 9.57 con parámetros {'C': 264.48700283625533,
'gamma': 468.1097115939347, 'kernel': 'rbf'}
1) Precisión media: 70.83 +/- 9.57 con parámetros {'C': 496.06159670665454,
'gamma': 89.75433828951704, 'kernel': 'rbf'}
1) Precisión media: 70.83 +/- 9.57 con parámetros {'C': 363.2747281986112,
'gamma': 847.0321634798003, 'kernel': 'rbf'}
1) Precisión media: 70.83 +/- 9.57 con parámetros {'C': 359.18141558786346,
'gamma': 131.64813262565332, 'kernel': 'rbf'}
1) Precisión media: 70.83 +/- 9.57 con parámetros {'C': 9.456648497428777,
'gamma': 524.3334538162322, 'kernel': 'rbf'}
1) Precisión media: 70.83 +/- 9.57 con parámetros {'C': 297.07954461197625,
'gamma': 60.36143190912177, 'kernel': 'rbf'}
1) Precisión media: 70.83 +/- 9.57 con parámetros {'C': 57.23447571702056,
'gamma': 532.4247947478979, 'kernel': 'rbf'}
El entrenamiento a durado 0.07169866561889648 segundos
8) Precisión media: 36.94 +/- 12.46 con parámetros {'C': 197.43743465989655,
'gamma': 781.2845981501584, 'kernel': 'sigmoid'}
3) Precisión media: 42.50 +/- 14.41 con parámetros {'C': 367.04331987384484,
'gamma': 331.9711266948303, 'kernel': 'sigmoid'}
5) Precisión media: 39.72 +/- 9.82 con parámetros {'C': 168.45931989146058,
'gamma': 571.9701867062855, 'kernel': 'sigmoid'}
1) Precisión media: 45.28 +/- 10.87 con parámetros {'C': 188.88474765880682,
'gamma': 986.3697466130012, 'kernel': 'sigmoid'}
8) Precisión media: 36.94 +/- 12.46 con parámetros {'C': 275.00446105231833,
'gamma': 762.047106254711, 'kernel': 'sigmoid'}
3) Precisión media: 42.50 +/- 14.41 con parámetros {'C': 399.27158051487226,
'gamma': 122.36535948601588, 'kernel': 'sigmoid'}
1) Precisión media: 45.28 +/- 10.87 con parámetros {'C': 97.33222256043095,
'gamma': 958.1955604856994, 'kernel': 'sigmoid'}
8) Precisión media: 36.94 +/- 20.05 con parámetros {'C': 496.8476787318492,
'gamma': 297.8384011289445, 'kernel': 'sigmoid'}
6) Precisión media: 37.22 +/- 18.68 con parámetros {'C': 234.11407995020528,
'gamma': 86.15628167589459, 'kernel': 'sigmoid'}
6) Precisión media: 37.22 +/- 18.68 con parámetros {'C': 21.10773037681829,
'gamma': 41.07446304056991, 'kernel': 'sigmoid'}
```

[18]:
```
import matplotlib.pyplot as plt
```

```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar([dic['kernel'] for dic in best_score],[dic['score'] for dic in␣
 ↪best_score])
plt.show()
```



[19]:
```
pd.DataFrame(best_score)
```

[19]:

	kernel	score	params
0	poly	0.419444	{'C': 53.821075955066966, 'gamma': 425.9568020…
1	rbf	0.708333	{'C': 89.593529467132, 'gamma': 484.3701547411…
2	sigmoid	0.452778	{'C': 188.88474765880682, 'gamma': 986.3697466…

Vemos que el kernel que mejor se ajusta al modelo es el radial (rbf). Utilizaremos este kernel con sus correspondientes parámetros para entrenar el modelo predictivo.

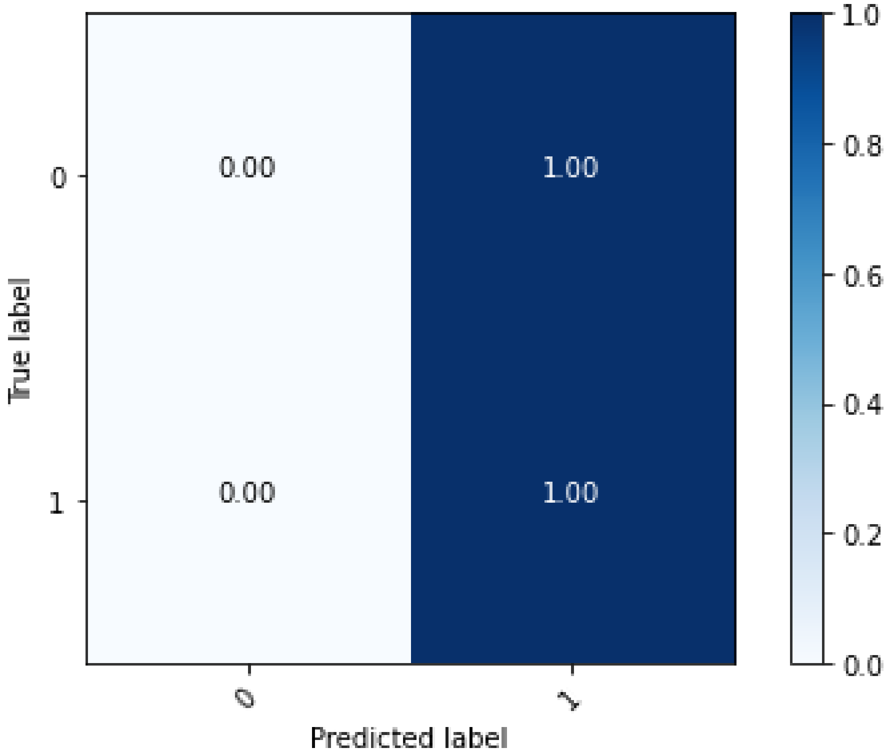
6.1 Evaluating the Algorithm

[20]:
```
from sklearn.metrics import accuracy_score

print("Valor óptimo para C: {}".format(best_score[1]['params']['C']))
print("Valor óptimo para gamma: {}".format(best_score[1]['params']["gamma"]))
```

```
modelSVM = svm.SVC(kernel=best_score[1]['kernel'],␣
↪gamma=best_score[1]['params']["gamma"], C=best_score[1]['params']['C']).
↪fit(X_train, y_train)
predict = modelSVM.predict(X_test)

accuracy = np.true_divide(np.sum(predict == y_test), predict.shape[0])*100
print("Precisión en el conjunto de test: {:.2f}%".format(accuracy))
```

```
Valor óptimo para C: 89.593529467132
Valor óptimo para gamma: 484.37015474111286
Precisión en el conjunto de test: 53.85%
```

[21]:
```
from sklearn.metrics import confusion_matrix
import itertools


cnf_matrix = confusion_matrix(y_test, predict)

def plot_confusion_matrix(cm, classes):
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    cmap=plt.cm.Blues

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], ".2f"),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

n_classes=["0","1"]
plot_confusion_matrix(cnf_matrix, classes=n_classes)
```



7 Run Prediction

[22]:
```
result = modelSVM.predict(dfDataSetToPredict[[
    "pedido.data.attributes.age",
    "pedido.data.attributes.diagnostic_main",
    "respuesta.articlesRevisedYear",
    "respuesta.pubmed_keys",
    "articulo"
]])

result
```

[22]:
```
array([1., 1., 1., …, 1., 1., 1.])
```