S5 - HOPE SVM-V2

December 9, 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	pedido.data.attributes.gender	articulo	respuesta.articlesRevisedYear
0	75.0	FISTULA PERITONEAL	male	27395425	2018
1	75.0	FISTULA PERITONEAL	male	28560554	2018
2	75.0	FISTULA PERITONEAL	male	28641726	2017
3	75.0	FISTULA PERITONEAL	male	26245344	2016
4	75.0	FISTULA PERITONEAL	male	28942543	2018
5	75.0	FISTULA PERITONEAL	male	24782153	2014
6	75.0	FISTULA PERITONEAL	male	28002229	2018
7	75.0	FISTULA PERITONEAL	male	27505109	2017
8	75.0	FISTULA PERITONEAL	male	24850546	2015
9	75.0	FISTULA PERITONEAL	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
```

Expand pubmed_keys attribute

[4]:
```
dfOrg['respuesta.pubmed_keys'] = dfOrg['respuesta.pubmed_keys'].apply(lambda x :
 ↪ str(x).split(','))

dfOrg = dfOrg.explode('respuesta.pubmed_keys').reset_index(drop=True)

dfOrg.head(10)
```

[4]:
```
   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  27395425                          2018
2                          male  27395425                          2018
3                          male  27395425                          2018
4                          male  27395425                          2018
5                          male  27395425                          2018
6                          male  27395425                          2018
7                          male  27395425                          2018
8                          male  27395425                          2018
9                          male  27395425                          2018

   respuesta.articlesRevisedMonth respuesta.pubmed_keys  utilidad
0                               1                Abdomen       1.0
1                               1         Adenocarcinoma       1.0
2                               1            Antiemetics       1.0
3                               1          Blood Culture       1.0
4                               1              Catharsis       1.0
5                               1               Diuresis       1.0
6                               1                Fistula       1.0
7                               1            Gastrectomy       1.0
8                               1      Incisional Hernia       1.0
9                               1              Intestines       1.0
```

2 Transform (factorice) from Categories to continuous atributes

Transform 'pedido.data.attributes.diagnostic_main' atribute

[5]:
```
dataDiagnosticMain, categoriesDiagnosticMain = pd.factorize(dfOrg['pedido.data.
↪attributes.diagnostic_main'])

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

Transform 'gender' atribute

[6]:
```
dataGender, categoriesGender = pd.factorize(dfOrg['pedido.data.attributes.
↪gender'])

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

Transform 'respuesta.pubmed_keys' atribute

[7]:
```
categoriesORGPubMedKeys = dfOrg['respuesta.pubmed_keys'].value_counts()

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

```
total: 353
```

[8]:
```
dataPubMedKeys, categoriesPubMedKeys = pd.factorize(dfOrg['respuesta.
↪pubmed_keys'])

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

[9]:
```
dfOrg.head(10)
```

[9]:

	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	27395425	2018	
2	0	27395425	2018	
3	0	27395425	2018	
4	0	27395425	2018	
5	0	27395425	2018	
6	0	27395425	2018	
7	0	27395425	2018	
8	0	27395425	2018	
9	0	27395425	2018	

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

[10]:
```
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 => 14758
```

Remove row with age eq NaN

[11]:
```
dfOrg = dfOrg[pd.notnull(dfOrg['pedido.data.attributes.age'])]
```

3 Standardize the Data

Choosed “age”, “diagnostic_main”, “month” and “pubmed_keys” attributes (based on PCA_V3 study)

[12]:
```
from sklearn.preprocessing import StandardScaler

features = ["pedido.data.attributes.age",
    "pedido.data.attributes.diagnostic_main",
    "respuesta.articlesRevisedMonth",
    "respuesta.pubmed_keys",
    "utilidad"
]

# 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
```

[12]:

	pedido.data.attributes.age	pedido.data.attributes.diagnostic_main	\
0	1.285887	-1.503163	
1	1.285887	-1.503163	
2	1.285887	-1.503163	
3	1.285887	-1.503163	
4	1.285887	-1.503163	
…	…	…	
15583	-0.607930	-0.586347	
15584	-0.607930	-0.586347	
15585	-0.607930	-0.586347	
15586	-0.607930	-0.586347	
15587	-0.607930	-0.586347	

	respuesta.articlesRevisedMonth	respuesta.pubmed_keys	utilidad
0	-1.463658	-1.089722	1.0
1	-1.463658	-1.080463	1.0
2	-1.463658	-1.071203	1.0
3	-1.463658	-1.061944	1.0
4	-1.463658	-1.052684	1.0
…	…	…	…
15583	-1.178433	-0.330441	NaN
15584	-1.178433	-0.978608	NaN
15585	-1.178433	0.891817	NaN
15586	-1.178433	-0.876753	NaN
15587	-1.178433	0.901077	NaN

[15578 rows x 5 columns]

4 Separe data by utilidad is defined

[13]:
```
dfDataSetComplete = dfStandarized[pd.notnull(dfStandarized['utilidad'])]

print(dfDataSetComplete.shape[0])

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

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

```
830
14748
```

5 SVM

We check the number of results

[14]:
```
dfDataSetComplete.groupby('utilidad').size()
```

[14]:
```
utilidad
0.0    346
1.0    484
dtype: int64
```

Separe “utilidad” atribute from dataToTrain

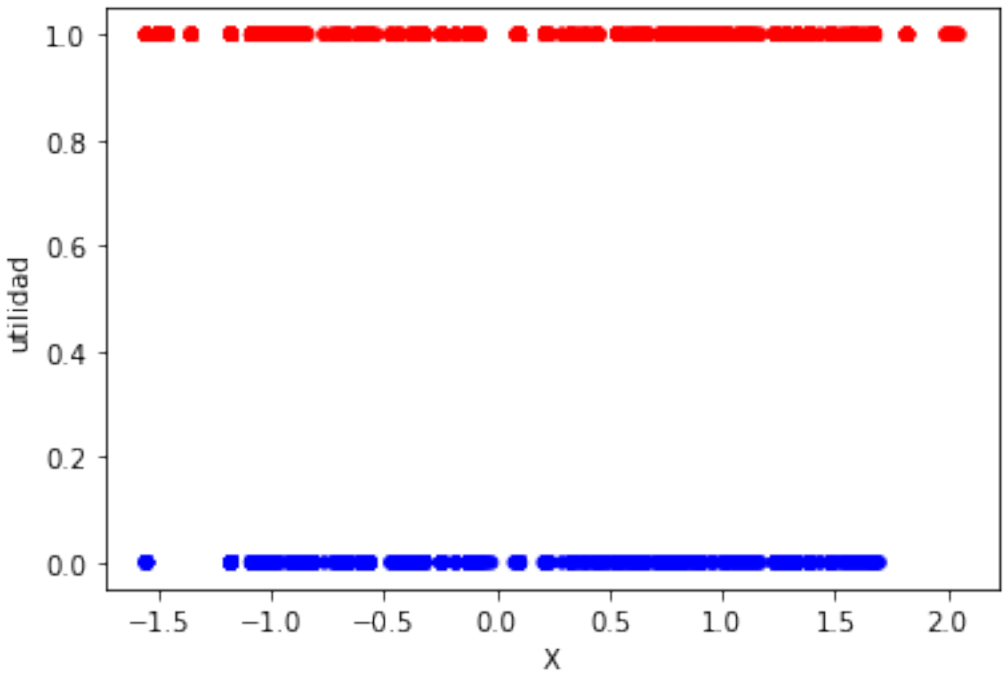
[15]:
```
X = np.array(dfDataSetComplete.drop(['utilidad'],1))
y = np.array(dfDataSetComplete['utilidad'])
X.shape
```

[15]:
```
(830, 4)
```

[16]:
```
from sklearn.model_selection import train_test_split

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

[17]:
```
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.articlesRevisedMonth",␣
↪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)
ax.set_xlabel('X')
```

[17]:
```
Text(0.5, 0, 'X')
```



A simple vista no podemos crear un hiperplano lineal (division 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" estan distribuidos sobre todo el plano de X. Exploraremos con los kernel methods, que metodo 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

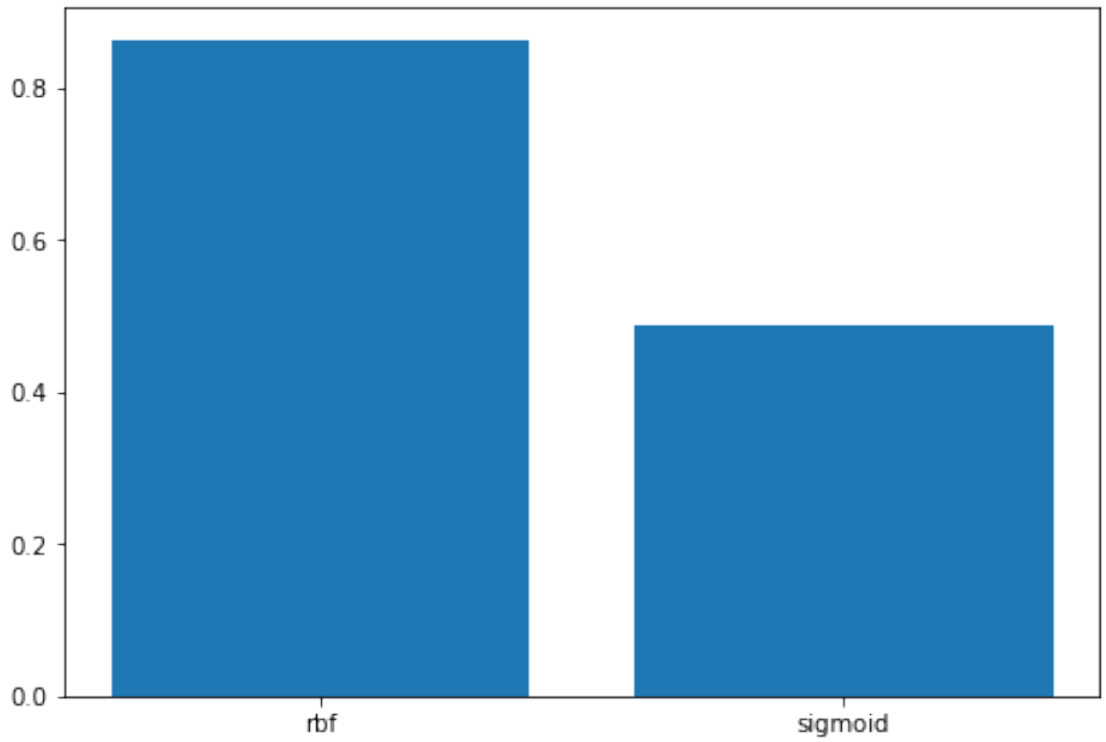
[18]:
```
from sklearn.model_selection import RandomizedSearchCV
from sklearn import svm
from scipy.stats import uniform as sp_rand
from time import time

clf = svm.SVC()

kernels = ['rbf', 'sigmoid']

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

```
    "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 0.6093668937683105 segundos
10) Precisión media: 76.21 +/- 3.94 con parámetros {'C': 360.85775701024386,
'gamma': 907.3825530117131, 'kernel': 'rbf'}
2) Precisión media: 82.15 +/- 3.10 con parámetros {'C': 317.80497788196783,
'gamma': 119.40946797705895, 'kernel': 'rbf'}
6) Precisión media: 76.69 +/- 4.23 con parámetros {'C': 87.01059367021347,
'gamma': 806.2789163354661, 'kernel': 'rbf'}
5) Precisión media: 77.98 +/- 3.45 con parámetros {'C': 149.54484171124932,
'gamma': 564.5199301320216, 'kernel': 'rbf'}
6) Precisión media: 76.69 +/- 4.23 con parámetros {'C': 118.97768424072203,
'gamma': 793.486590684318, 'kernel': 'rbf'}
6) Precisión media: 76.69 +/- 4.23 con parámetros {'C': 150.02147952417772,
'gamma': 737.8299797134209, 'kernel': 'rbf'}
6) Precisión media: 76.69 +/- 4.23 con parámetros {'C': 363.8370243112447,
'gamma': 848.9827977113308, 'kernel': 'rbf'}
3) Precisión media: 79.74 +/- 2.79 con parámetros {'C': 423.45666708016336,
'gamma': 277.59190556966644, 'kernel': 'rbf'}
1) Precisión media: 86.33 +/- 1.48 con parámetros {'C': 169.2424566596866,
'gamma': 21.685491125386175, 'kernel': 'rbf'}
```

```
4) Precisión media: 78.62 +/- 3.45 con parámetros {'C': 56.26738061340281,
'gamma': 313.7065587800348, 'kernel': 'rbf'}
El entrenamiento a durado 0.2546238899230957 segundos
7) Precisión media: 48.38 +/- 4.50 con parámetros {'C': 391.1677890243306,
'gamma': 497.20459077927774, 'kernel': 'sigmoid'}
7) Precisión media: 48.38 +/- 4.50 con parámetros {'C': 492.9377695700987,
'gamma': 453.0399489373676, 'kernel': 'sigmoid'}
4) Precisión media: 48.54 +/- 4.64 con parámetros {'C': 330.13037526527904,
'gamma': 724.4299745866858, 'kernel': 'sigmoid'}
1) Precisión media: 48.86 +/- 4.52 con parámetros {'C': 141.66423772199033,
'gamma': 964.081623637728, 'kernel': 'sigmoid'}
7) Precisión media: 48.38 +/- 4.50 con parámetros {'C': 277.89947393631587,
'gamma': 499.06800915767286, 'kernel': 'sigmoid'}
4) Precisión media: 48.54 +/- 4.64 con parámetros {'C': 339.4193364446046,
'gamma': 849.8383579899024, 'kernel': 'sigmoid'}
1) Precisión media: 48.86 +/- 4.52 con parámetros {'C': 8.782687504322816,
'gamma': 965.8092683063094, 'kernel': 'sigmoid'}
4) Precisión media: 48.54 +/- 4.64 con parámetros {'C': 232.84344873561824,
'gamma': 200.82390241386747, 'kernel': 'sigmoid'}
7) Precisión media: 48.38 +/- 4.50 con parámetros {'C': 340.8373827954047,
'gamma': 346.17465054874623, 'kernel': 'sigmoid'}
1) Precisión media: 48.86 +/- 4.52 con parámetros {'C': 491.55519979656776,
'gamma': 857.6069742445022, 'kernel': 'sigmoid'}
```

[19]:
```
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()
```



[20]:
```
pd.DataFrame(best_score)
```

[20]:

	kernel	score	params
0	rbf	0.863327	{'C': 169.2424566596866, 'gamma': 21.685491125…
1	sigmoid	0.488617	{'C': 141.66423772199033, 'gamma': 964.0816236…

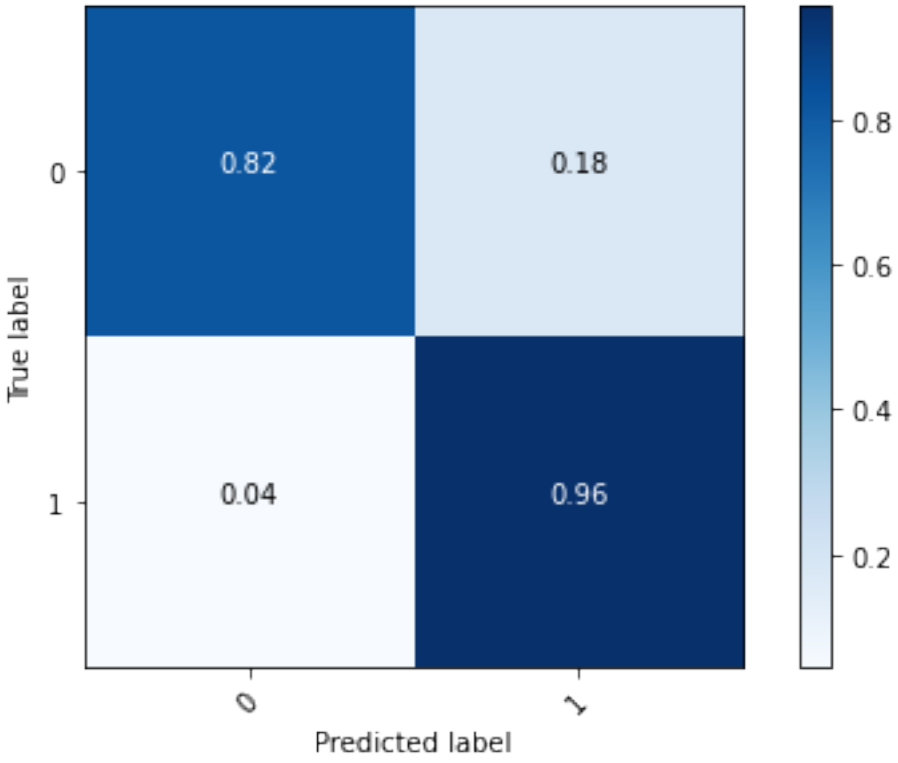
Vemos que el kernel que mejor se ajusta al modelo es el radial (rbf). Utilizaremos este kernel con sus correspondientes parámetros para entrenar el modelo predictivo

6.1 Evaluating the Algorithm

[21]:
```
from sklearn.metrics import accuracy_score

print("Valor óptimo para C: {}".format(best_score[0]['params']['C']))
print("Valor óptimo para gamma: {}".format(best_score[0]['params']["gamma"]))

modelSVM = svm.SVC(kernel=best_score[0]['kernel'],␣
 ↪gamma=best_score[0]['params']["gamma"], C=best_score[0]['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: 169.2424566596866
Valor óptimo para gamma: 21.685491125386175
Precisión en el conjunto de test: 89.42%
```

[22]:
```python
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

[23]:
```
result = modelSVM.predict(dfDataSetToPredict[["pedido.data.attributes.age",
    "pedido.data.attributes.diagnostic_main",
    "respuesta.articlesRevisedMonth",
    "respuesta.pubmed_keys"
]])

result
```

[23]:
```
array([1., 1., 1., …, 1., 1., 1.])
```

[ ]:
```

```