Recipes_v2

August 8, 2020

1 Comparación de diferentes algoritmos de clasificación

Por: Kevin Daniel Sánchez Díaz

[1]:
```
# Importar librerias principales
import pandas as pd
pd.set_option("display.max_columns", None)
import numpy as np
```

[2]:
```
# Leer csv
recipes = pd.read_csv("recipes.csv")

print("Data read into dataframe!") # takes about 30 secondsc
```

```
Data read into dataframe!
```

1.1 Transformar el DataFrame

[3]:
```
# fix name of the column displaying the cuisine
column_names = recipes.columns.values
column_names[0] = "cuisine"
recipes.columns = column_names

# convert cuisine names to lower case
recipes["cuisine"] = recipes["cuisine"].str.lower()

# make the cuisine names consistent
recipes.loc[recipes["cuisine"] == "austria", "cuisine"] = "austrian"
recipes.loc[recipes["cuisine"] == "belgium", "cuisine"] = "belgian"
recipes.loc[recipes["cuisine"] == "china", "cuisine"] = "chinese"
recipes.loc[recipes["cuisine"] == "canada", "cuisine"] = "canadian"
recipes.loc[recipes["cuisine"] == "netherlands", "cuisine"] = "dutch"
recipes.loc[recipes["cuisine"] == "france", "cuisine"] = "french"
recipes.loc[recipes["cuisine"] == "germany", "cuisine"] = "german"
recipes.loc[recipes["cuisine"] == "india", "cuisine"] = "indian"
recipes.loc[recipes["cuisine"] == "indonesia", "cuisine"] = "indonesian"
recipes.loc[recipes["cuisine"] == "iran", "cuisine"] = "iranian"
```

```
recipes.loc[recipes["cuisine"] == "italy", "cuisine"] = "italian"
recipes.loc[recipes["cuisine"] == "japan", "cuisine"] = "japanese"
recipes.loc[recipes["cuisine"] == "israel", "cuisine"] = "jewish"
recipes.loc[recipes["cuisine"] == "korea", "cuisine"] = "korean"
recipes.loc[recipes["cuisine"] == "lebanon", "cuisine"] = "lebanese"
recipes.loc[recipes["cuisine"] == "malaysia", "cuisine"] = "malaysian"
recipes.loc[recipes["cuisine"] == "mexico", "cuisine"] = "mexican"
recipes.loc[recipes["cuisine"] == "pakistan", "cuisine"] = "pakistani"
recipes.loc[recipes["cuisine"] == "philippines", "cuisine"] = "philippine"
recipes.loc[recipes["cuisine"] == "scandinavia", "cuisine"] = "scandinavian"
recipes.loc[recipes["cuisine"] == "spain", "cuisine"] = "spanish_portuguese"
recipes.loc[recipes["cuisine"] == "portugal", "cuisine"] = "spanish_portuguese"
recipes.loc[recipes["cuisine"] == "switzerland", "cuisine"] = "swiss"
recipes.loc[recipes["cuisine"] == "thailand", "cuisine"] = "thai"
recipes.loc[recipes["cuisine"] == "turkey", "cuisine"] = "turkish"
recipes.loc[recipes["cuisine"] == "vietnam", "cuisine"] = "vietnamese"
recipes.loc[recipes["cuisine"] == "uk-and-ireland", "cuisine"] = "uk-and-irish"
recipes.loc[recipes["cuisine"] == "irish", "cuisine"] = "uk-and-irish"


# remove data for cuisines with < 50 recipes:
recipes_counts = recipes["cuisine"].value_counts()
cuisines_indices = recipes_counts > 50

cuisines_to_keep = list(np.array(recipes_counts.index.values)[np.
 ↪array(cuisines_indices)])
recipes = recipes.loc[recipes["cuisine"].isin(cuisines_to_keep)]

# convert all Yes's to 1's and the No's to 0's
recipes = recipes.replace(to_replace="Yes", value=1)
recipes = recipes.replace(to_replace="No", value=0)
```

[4]:
```
# importar librerias de gráficos
%matplotlib inline

from sklearn.metrics import accuracy_score, confusion_matrix

import matplotlib.pyplot as plt

import graphviz

import itertools
```

[5]:
```
# Paises displonibles
recipes['cuisine'].value_counts().index
```

[5]:
```
Index(['american', 'italian', 'mexican', 'french', 'asian', 'east_asian',
       'korean', 'canadian', 'indian', 'western', 'chinese',
       'spanish_portuguese', 'uk-and-irish', 'southern_soulfood', 'jewish',
       'japanese', 'mediterranean', 'thai', 'german', 'scandinavian',
       'middleeastern', 'central_southamerican', 'eastern-europe', 'greek',
       'english_scottish', 'caribbean', 'cajun_creole',
       'easterneuropean_russian', 'moroccan', 'african', 'southwestern',
       'south-america', 'vietnamese', 'north-african'],
      dtype='object')
```

[6]:
```
# Paises que se analizaran
labels = ["korean", "japanese", "chinese", "thai", "indian"]
recipesF = recipes[recipes.cuisine.isin(labels)]
```

[7]:
```
# Crear variable X
X_recipes = recipesF.drop('cuisine', axis=1)
X_recipes.shape
```

[7]:
```
(2448, 383)
```

[8]:
```
# Crear variable y
y_recipes = recipesF['cuisine']
y_recipes.shape
```

[8]:
```
(2448,)
```

[9]:
```
# Separar las variables X y en entrenamiento y prueba
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_recipes, y_recipes,␣
 ↪test_size=0.2, random_state=42)
```

1.2 Modelado

[10]:
```
# Función que crea la matriz de confusión
evaluation = {}
def confusion(predictor, alg):
    test_cuisines = np.unique(y_test)
    bamboo_confusion_matrix = confusion_matrix(y_test, predictor, labels)
    cal = bamboo_confusion_matrix.diagonal().mean()
    print(alg+":",cal)
    evaluation[alg] = cal
    title = 'Bamboo Confusion Matrix ' + alg
    cmap = plt.cm.Blues

    plt.figure(figsize=(8, 6))
    bamboo_confusion_matrix = (
```

```
        bamboo_confusion_matrix.astype('float') / bamboo_confusion_matrix.
↪sum(axis=1)[:, np.newaxis]
        ) * 100

    plt.imshow(bamboo_confusion_matrix, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(test_cuisines))
    plt.xticks(tick_marks, test_cuisines)
    plt.yticks(tick_marks, test_cuisines)

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

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

    plt.show()
```

[11]:
```
# Importar los algoritmos de clasificación
from sklearn import tree
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.svm import SVC
```

[12]:
```
# Nombres de los algoritmos de clasificación que se utilizarán
names = ["DecisionTreeClassifier", "MLPClassifier", "KNeighborsClassifier",
         "GaussianProcessClassifier", "RandomForestClassifier",␣
↪"AdaBoostClassifier",
         "GaussianNB", "QuadraticDiscriminantAnalysis", "SVC Linear", "SVC␣
↪Poly"]
```

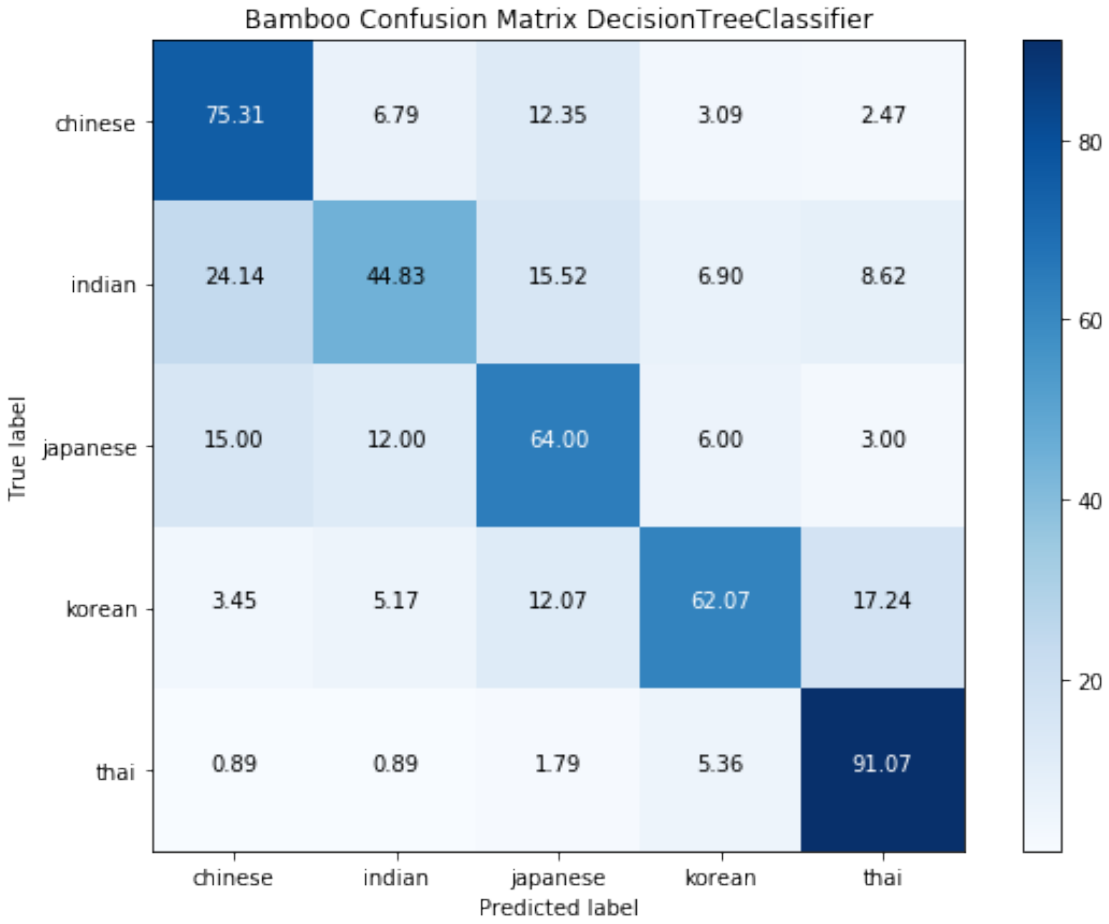
[13]:
```
# Instancias de los algoritmos a utilizar
classifiers = [
    tree.DecisionTreeClassifier(max_depth=33, random_state=42),
    MLPClassifier(alpha=1, max_iter=1000, random_state=42),
```

```
    KNeighborsClassifier(13),
    GaussianProcessClassifier(),
    RandomForestClassifier(max_depth=21, random_state=42),
    AdaBoostClassifier(random_state=42),
    GaussianNB(),
    QuadraticDiscriminantAnalysis(),
    SVC(kernel="linear", C=0.025),
    SVC(kernel="poly",gamma=2, C=1)
]
```

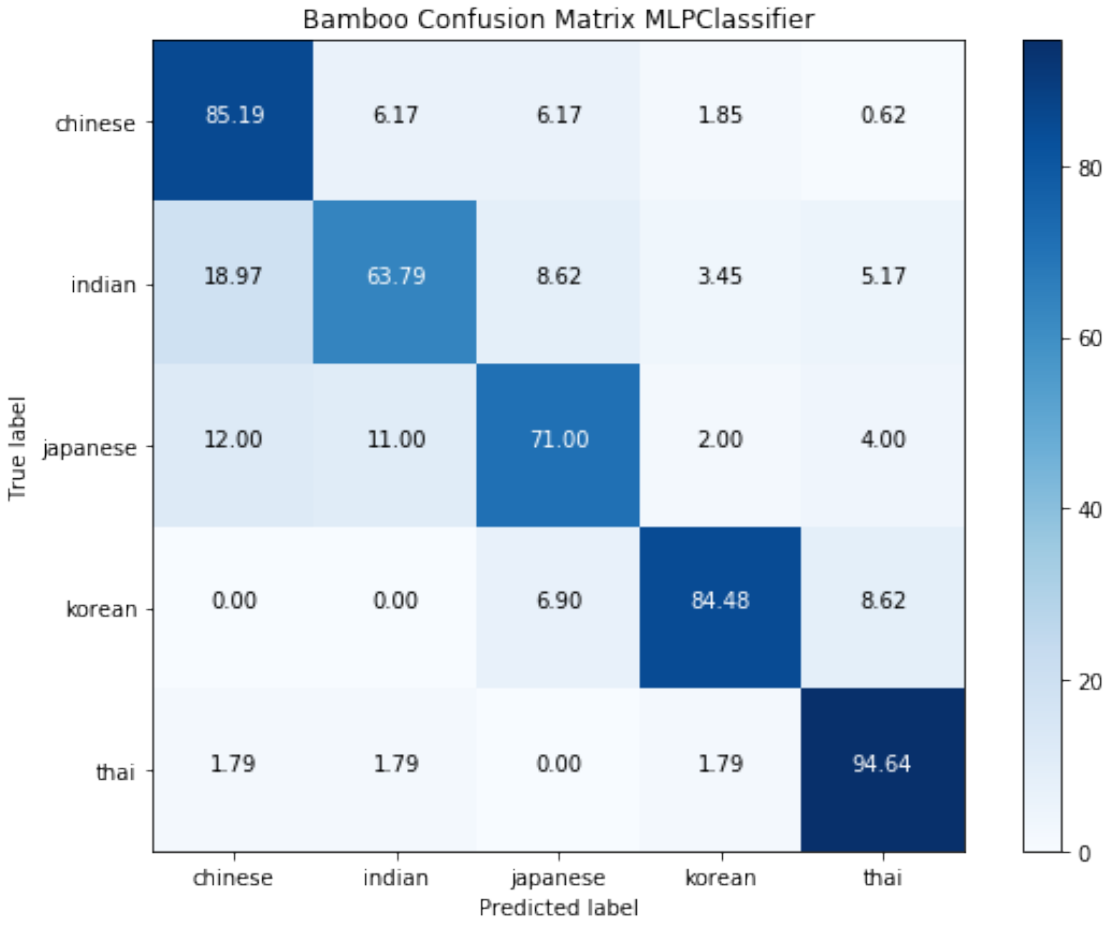
1.3 Comparación

[14]:
```
# Impresión de todas las matrices de confusión y de la calificación de cada␣
 ↪algoritmo
for name, clf in zip(names, classifiers):
    clf.fit(X_train, y_train)
    %timeit clf.predict(X_test)
    pred = clf.predict(X_test)
    confusion(pred, name)
    print("-------------------------------------------------------------------")
```

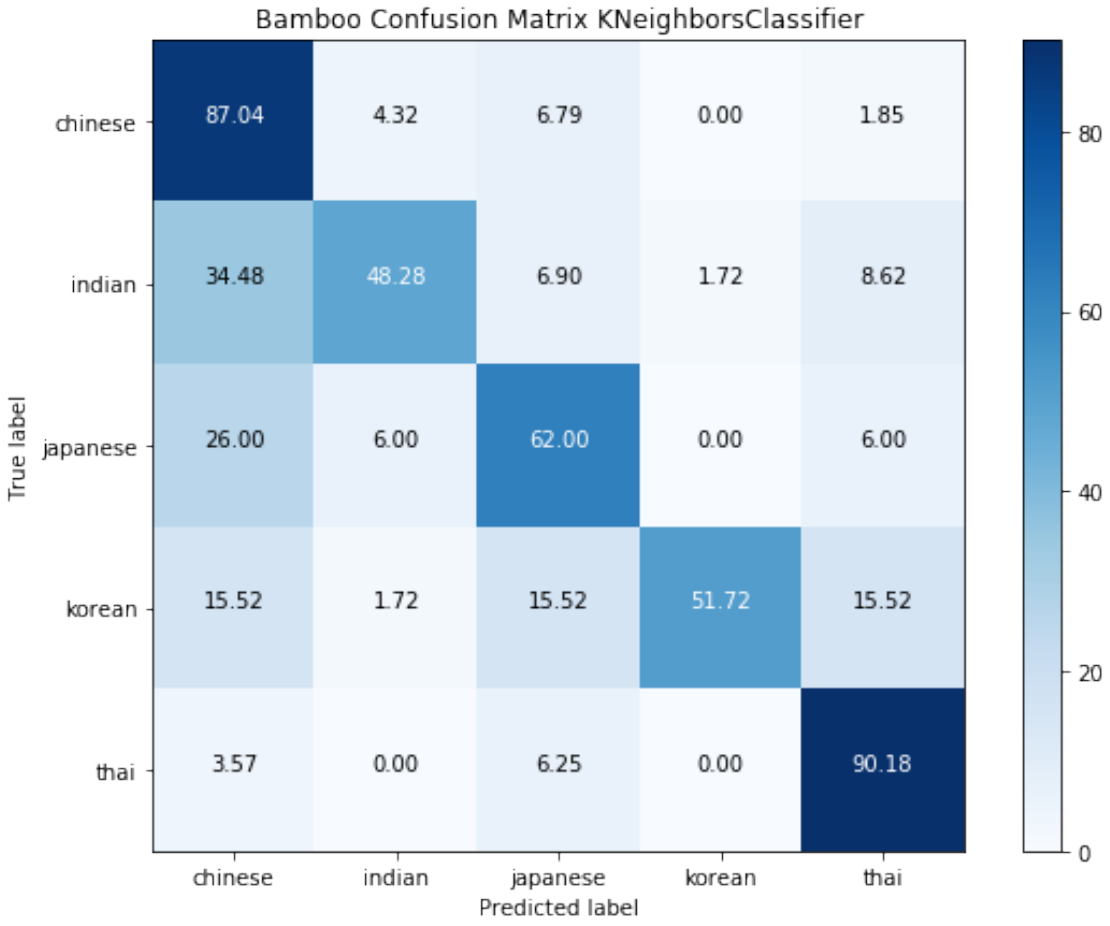
```
5.1 ms ± 236 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
DecisionTreeClassifier: 70.0
```



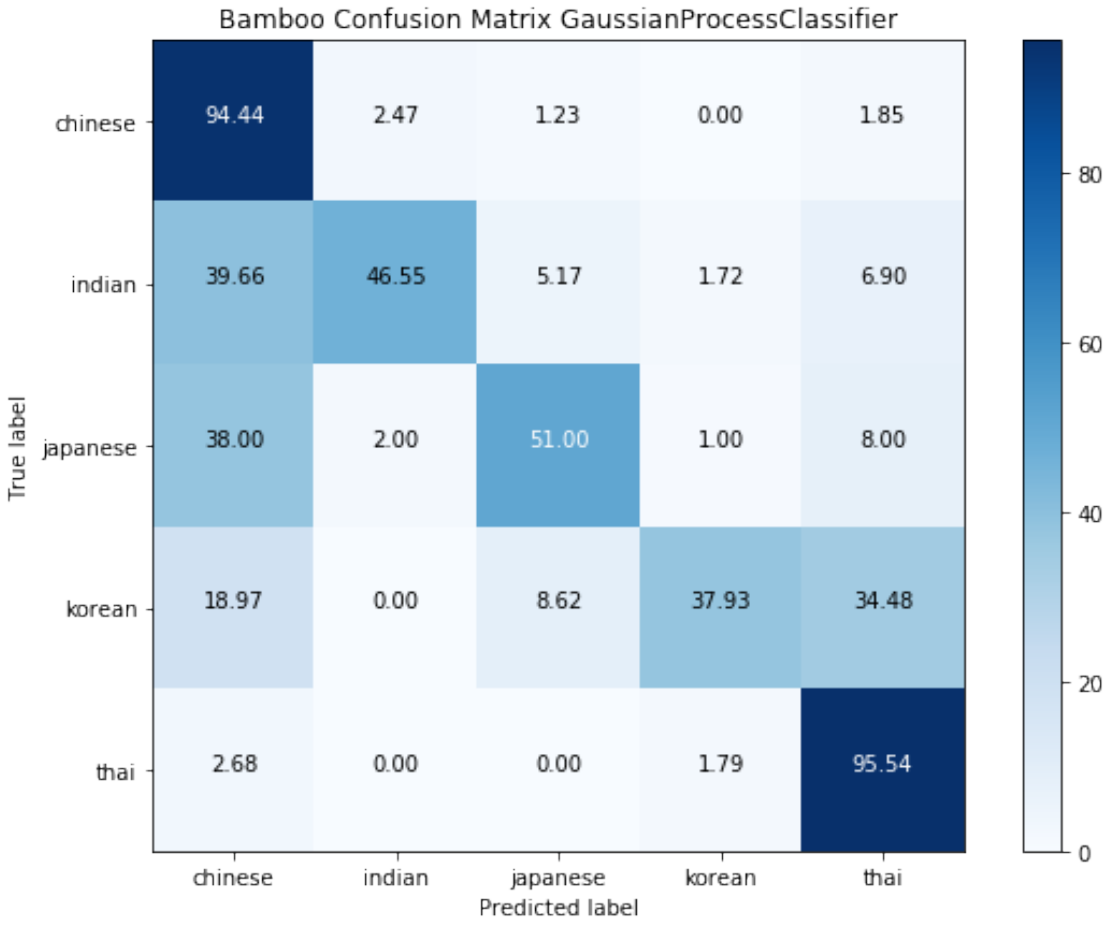
```
23.2 ms ± 6.81 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
MLPClassifier: 80.2
```



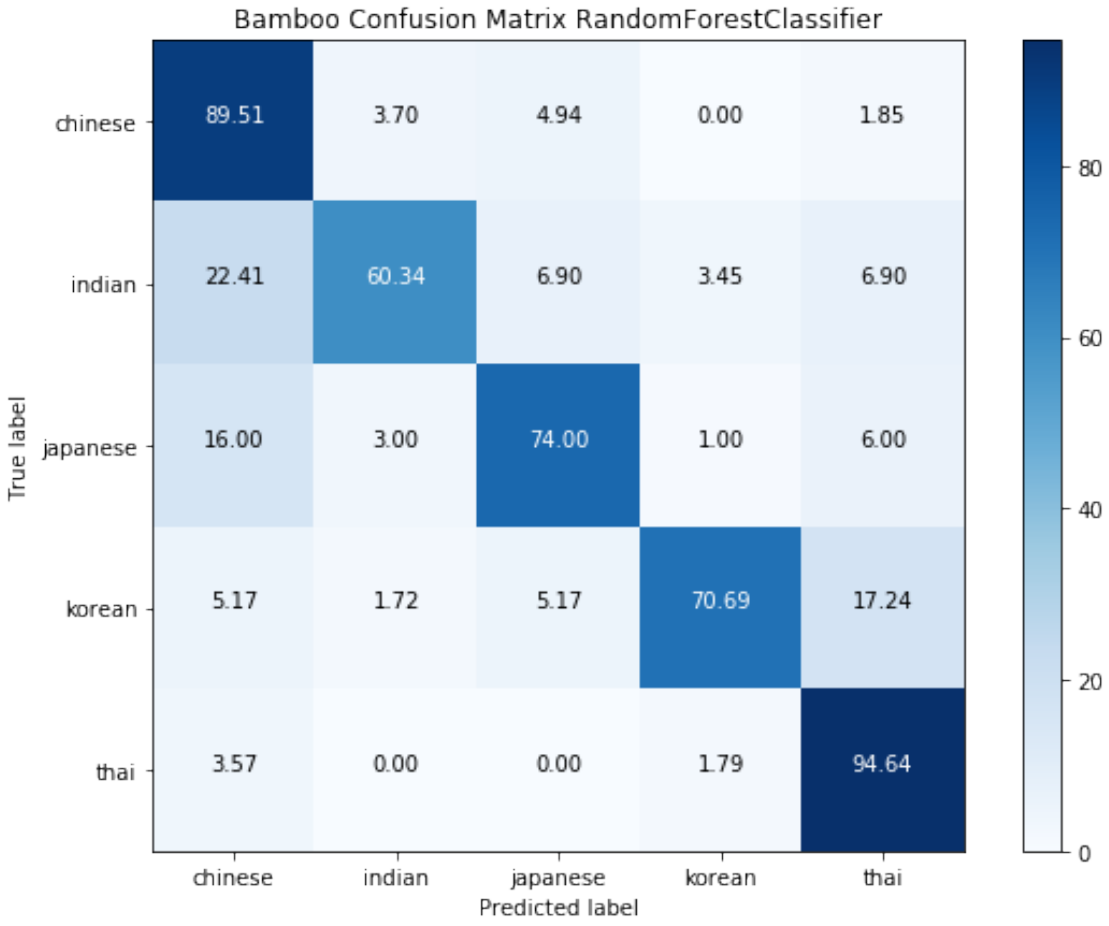
```
3.66 s ± 150 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
KNeighborsClassifier: 72.4
```



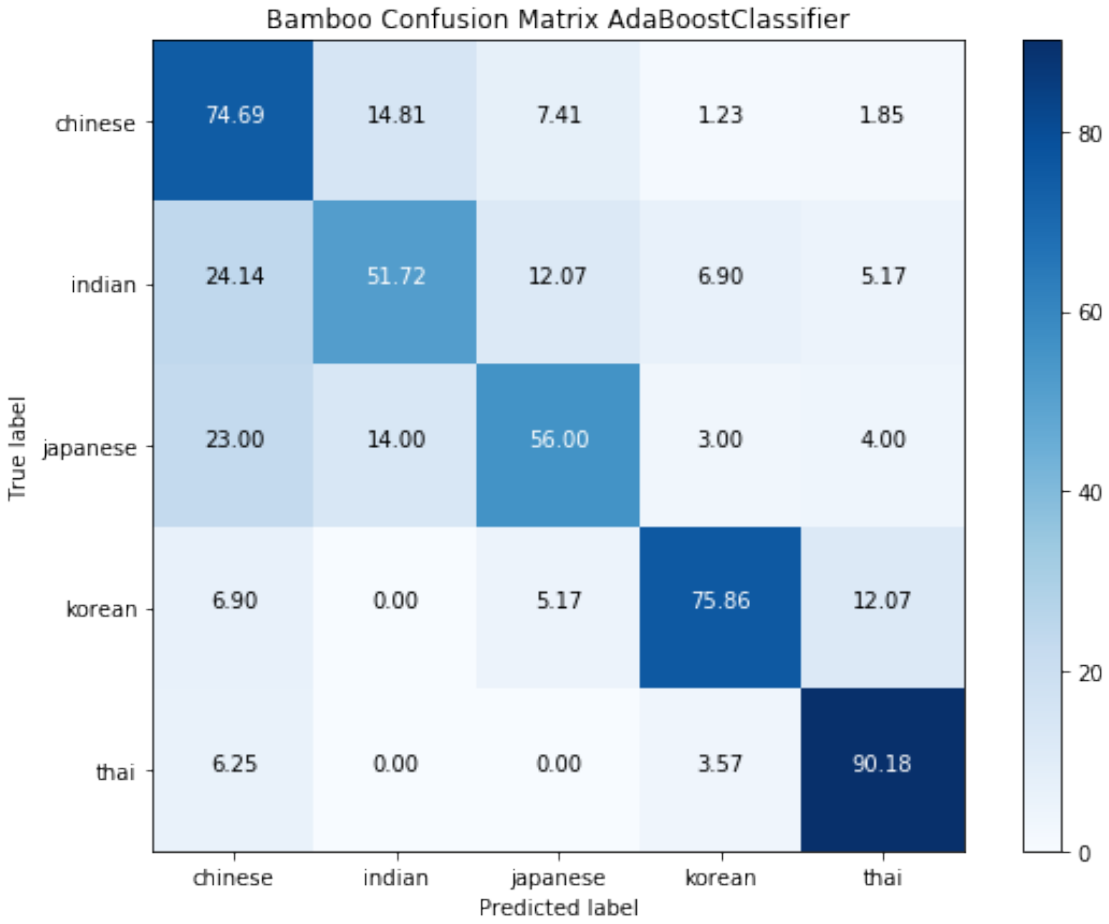
```
24.6 s ± 2.97 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
GaussianProcessClassifier: 72.0
```



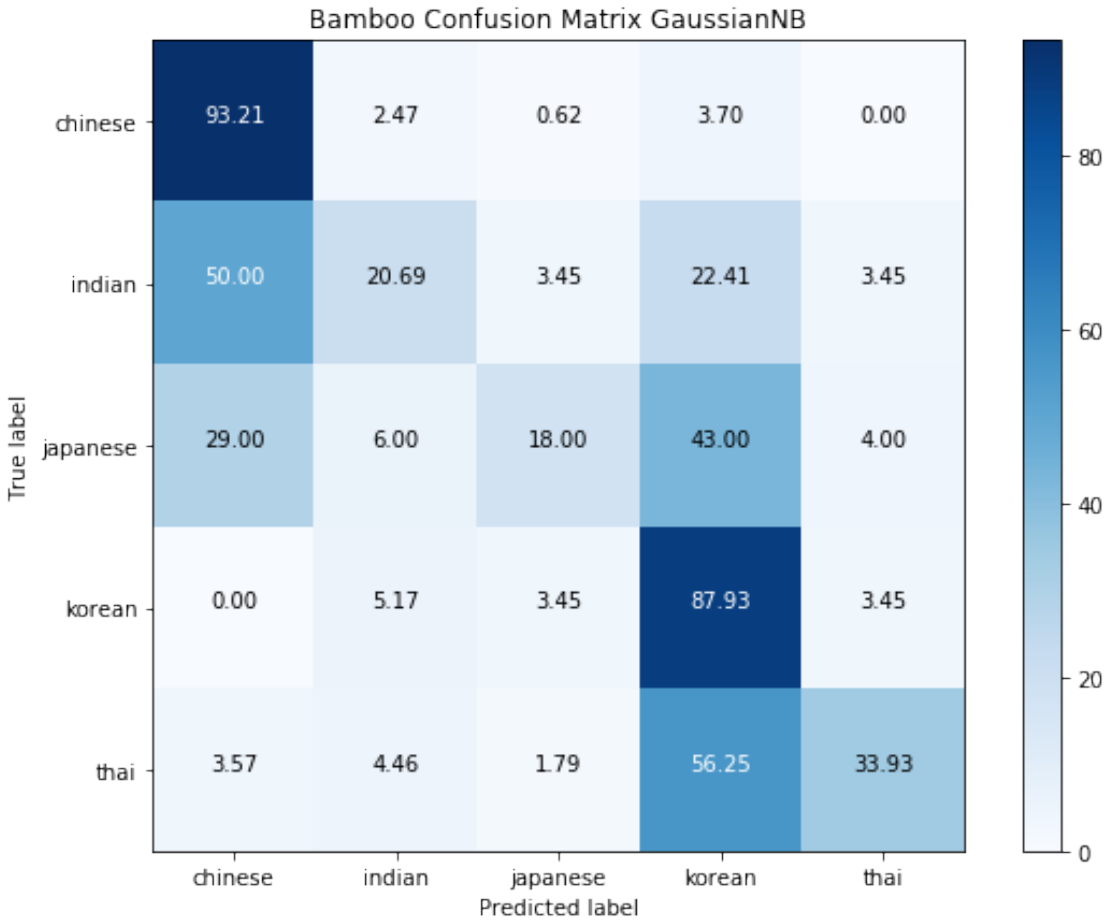

```
------------------------------------------------------------------
76.2 ms ± 495 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
RandomForestClassifier: 80.2
```



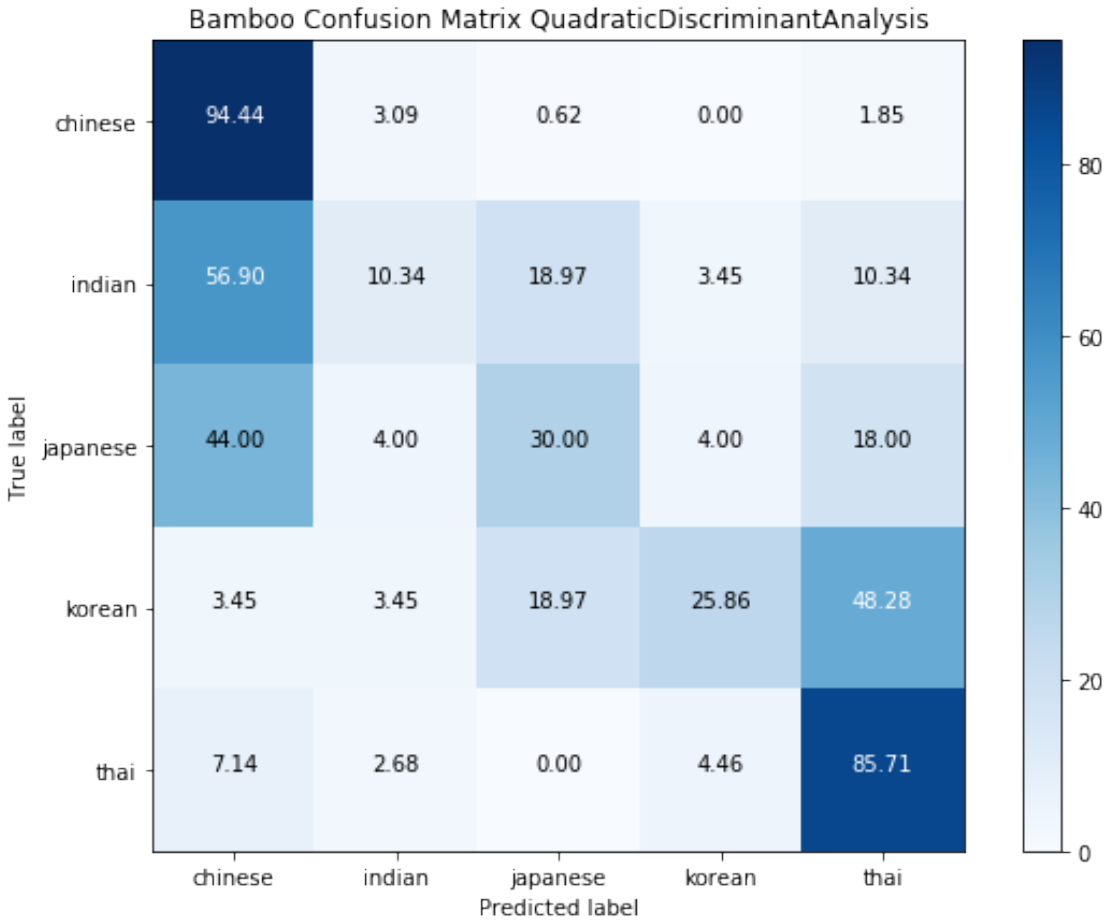
```
132 ms ± 3.79 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
AdaBoostClassifier: 70.4
```



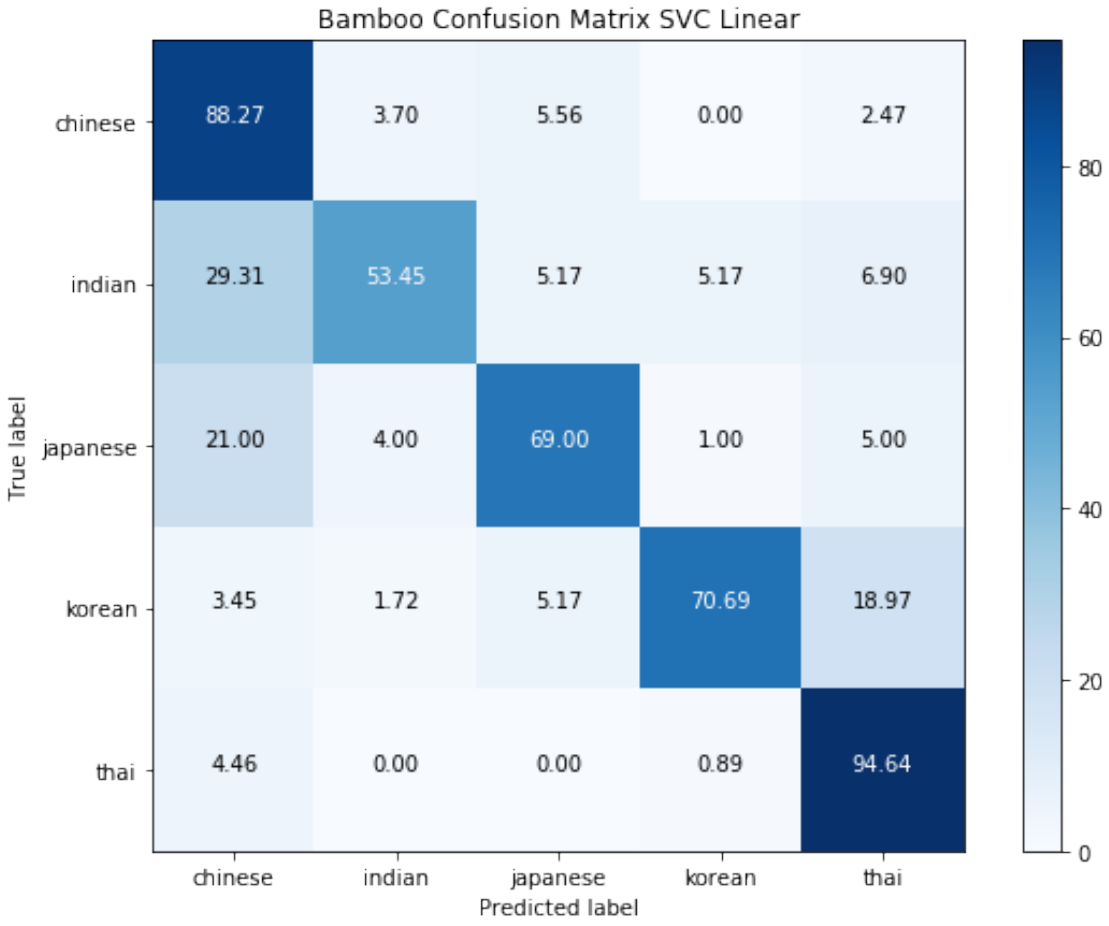

```
------------------------------------------------------------------
31.3 ms ± 1.33 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
GaussianNB: 54.0
```



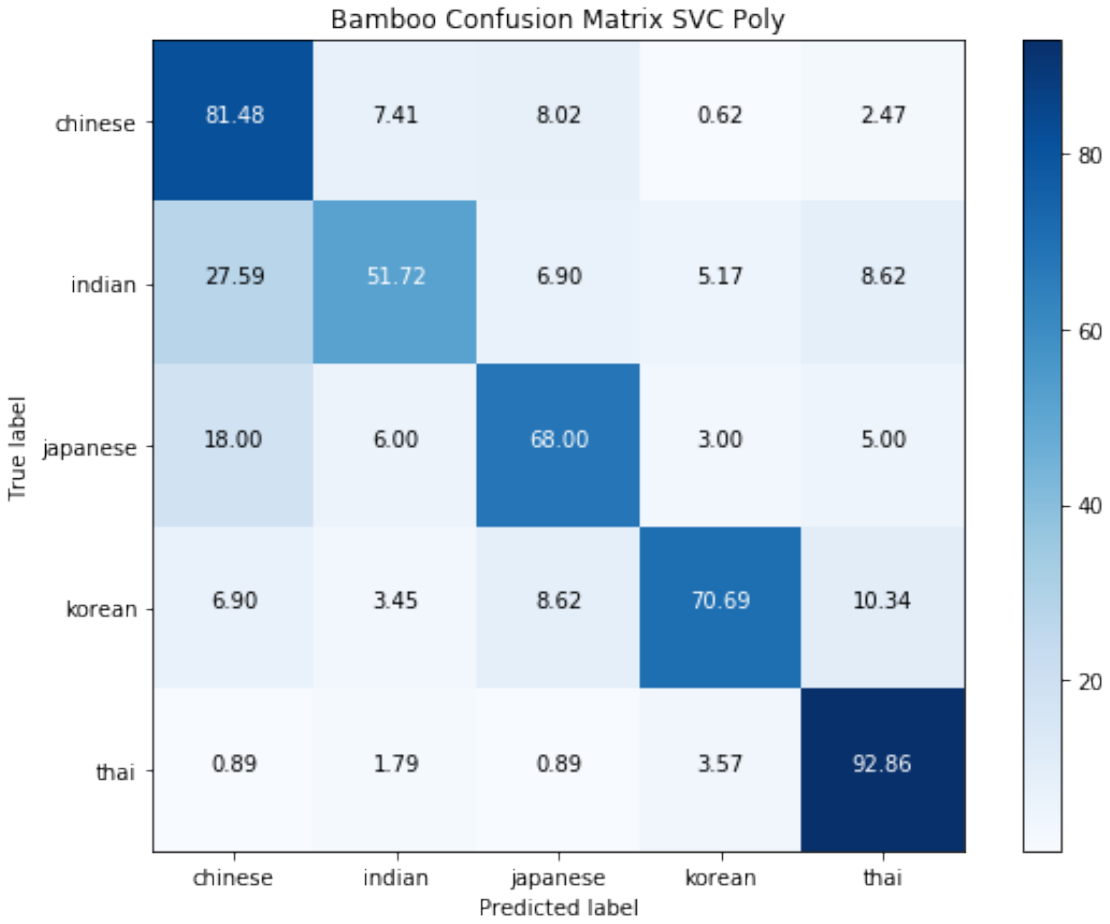
```
/home/kevin/anaconda3/lib/python3.7/site-
packages/sklearn/discriminant_analysis.py:691: UserWarning: Variables are
collinear
  warnings.warn("Variables are collinear")

201 ms ± 9.96 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
QuadraticDiscriminantAnalysis: 60.0
```




```
--------------------------------------------------------------------
599 ms ± 18.3 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
SVC Linear: 78.0
```



```
527 ms ± 45 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
SVC Poly: 75.0
```



---

[15]:
```
# Calificacion de cada algoritmo
evaluation
```

[15]:
```
{'DecisionTreeClassifier': 70.0,
 'MLPClassifier': 80.2,
 'KNeighborsClassifier': 72.4,
 'GaussianProcessClassifier': 72.0,
 'RandomForestClassifier': 80.2,
 'AdaBoostClassifier': 70.4,
 'GaussianNB': 54.0,
 'QuadraticDiscriminantAnalysis': 60.0,
 'SVC Linear': 78.0,
 'SVC Poly': 75.0}
```