Recipes

August 7, 2020

[1]:

```python
import pandas as pd # import library to read data into dataframe
pd.set_option("display.max_columns", None)
import numpy as np # import numpy library
import re # import library for regular expression
import random # library for random number generation
```

[2]:

```python
recipes = pd.read_csv("recipes.csv")

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

```
Data read into dataframe!
```

[3]:

```python
# 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)
```

[5]:

```
# import decision trees scikit-learn libraries
%matplotlib inline

from sklearn.metrics import accuracy_score, confusion_matrix

import matplotlib.pyplot as plt

import graphviz

import itertools
```

[6]:

```
recipes['cuisine'].value_counts().index
```

[6]:

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

[7]:

```
labels = ["korean", "japanese", "chinese", "thai", "indian"]
recipesF = recipes[recipes.cuisine.isin(labels)]
```

[8]:

```
X_recipes = recipesF.drop('cuisine', axis=1)
X_recipes.shape
```

[8]:

```
(2448, 383)
```

[9]:

```
y_recipes = recipesF['cuisine']
y_recipes.shape
```

[9]:

```
(2448,)
```

[10]:

```
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)
```

[11]:

```
print(y_train.value_counts().shape)
print(y_test.value_counts().shape)
```

```
(5,)
(5,)
```

[12]:

```
evaluation = {}
```

[13]:

```
def confusion(predictor, alg):
    test_cuisines = np.unique(y_test)
    bamboo_confusion_matrix = confusion_matrix(y_test, predictor, labels)
    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()
```

1 DecisionTreeClassifier

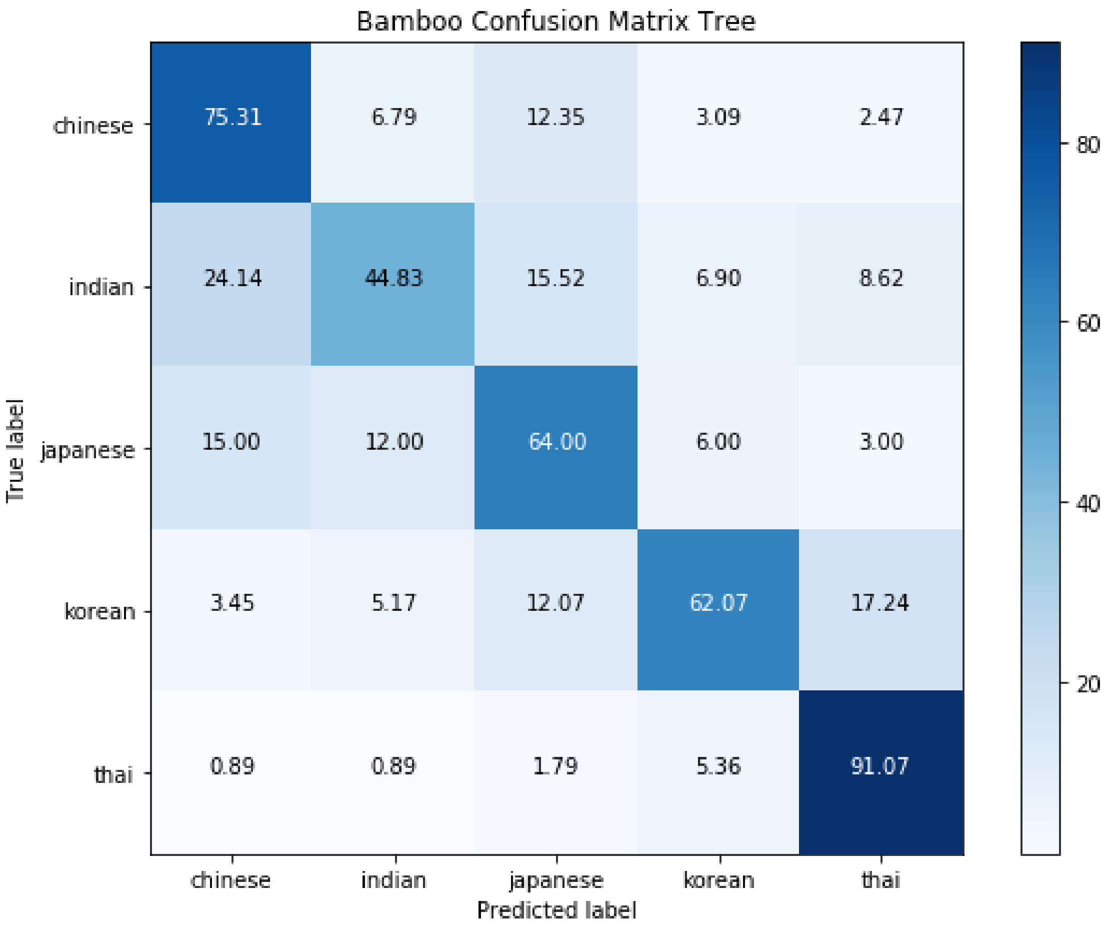
[14]:
```
from sklearn import tree

decisionTree = tree.DecisionTreeClassifier(max_depth=33, random_state=42)
decisionTree.fit(X_train, y_train)
```

[14]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=33, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=42, splitter='best')
```

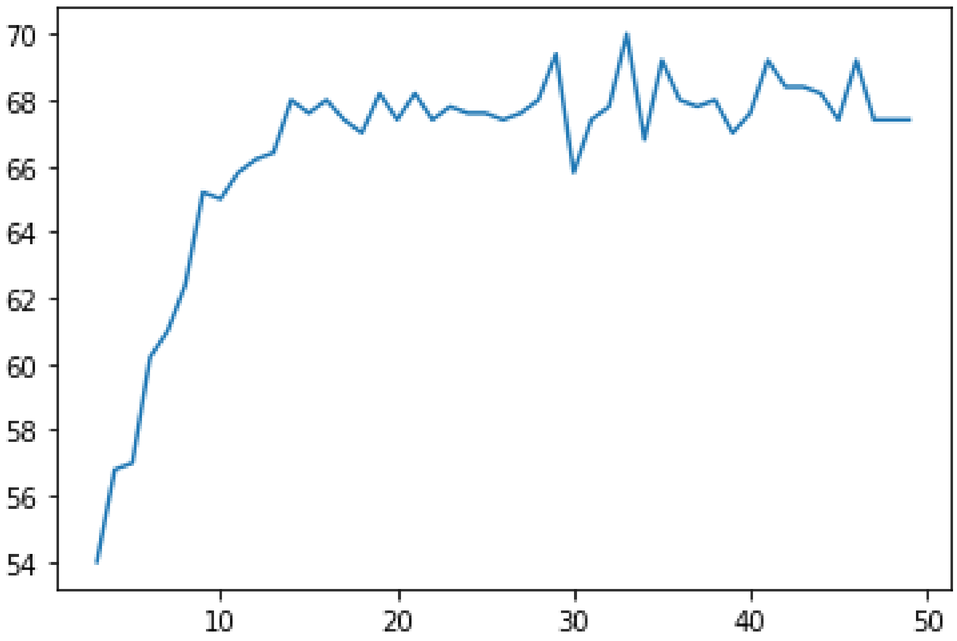
[15]:
```
recipes_pred_tree = decisionTree.predict(X_test)
```

[16]:
```
confusion(recipes_pred_tree, "Tree")
```



[18]:

```
evaluate = {}
for i in range(3,50):
    decisionTree = tree.DecisionTreeClassifier(max_depth=i, random_state=42)
    decisionTree.fit(X_train, y_train)
    recipes_pred = decisionTree.predict(X_test)
    bamboo_confusion_matrix = confusion_matrix(y_test, recipes_pred, labels)
    evaluate[i] = bamboo_confusion_matrix.diagonal().mean()
eva = pd.Series(evaluate)
eva.plot();
```



```
[20]: bamboo_confusion_matrix = confusion_matrix(y_test, recipes_pred_tree, labels)
      evaluation["DecisionTreeClassifier"] = bamboo_confusion_matrix.diagonal().mean()
```

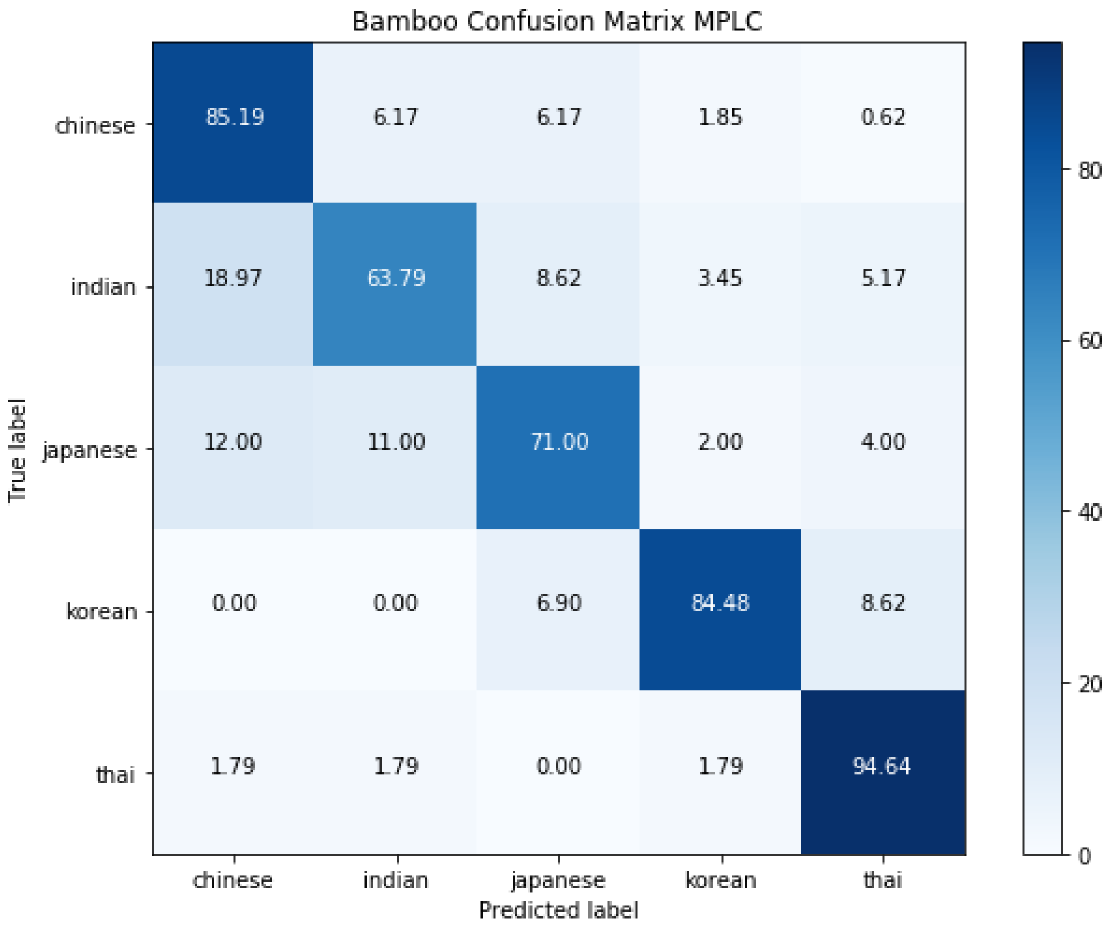
2 MLPClassifier

```
[21]: from sklearn.neural_network import MLPClassifier
      mplc = MLPClassifier(alpha=1, max_iter=1000, random_state=42)
      mplc.fit(X_train, y_train)
```

```
[21]: MLPClassifier(activation='relu', alpha=1, batch_size='auto', beta_1=0.9,
                    beta_2=0.999, early_stopping=False, epsilon=1e-08,
                    hidden_layer_sizes=(100,), learning_rate='constant',
                    learning_rate_init=0.001, max_fun=15000, max_iter=1000,
                    momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
                    power_t=0.5, random_state=42, shuffle=True, solver='adam',
                    tol=0.0001, validation_fraction=0.1, verbose=False,
                    warm_start=False)
```

```
[22]: recipes_pred_mplc = mplc.predict(X_test)
```

```
[23]: confusion(recipes_pred_mplc, "MPLC")
```



[24]:
```
bamboo_confusion_matrix = confusion_matrix(y_test, recipes_pred_mplc, labels)
evaluation["MLPClassifier"] = bamboo_confusion_matrix.diagonal().mean()
```

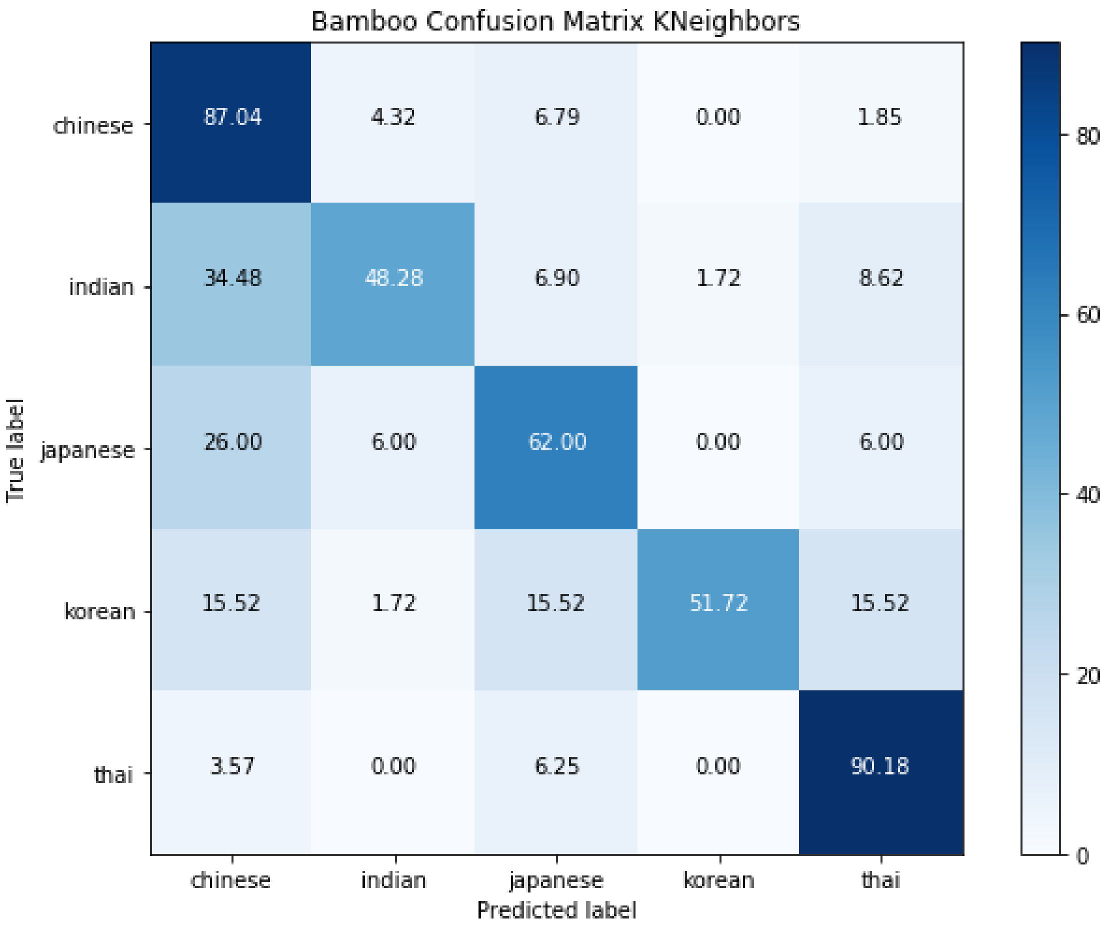
3 KNeighborsClassifier

[25]:
```
from sklearn.neighbors import KNeighborsClassifier
Kn = KNeighborsClassifier(13)
Kn.fit(X_train, y_train)
```

[25]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=13, p=2,
                     weights='uniform')
```

[26]:
```
recipes_pred_KN = Kn.predict(X_test)
```

[27]:
```
confusion(recipes_pred_KN, "KNeighbors")
```



[28]:

```
bamboo_confusion_matrix = confusion_matrix(y_test, recipes_pred_KN, labels)
evaluation["KNeighborsClassifier"] = bamboo_confusion_matrix.diagonal().mean()
```

4 GaussianProcessClassifier

[29]:

```
from sklearn.gaussian_process import GaussianProcessClassifier
gauss = GaussianProcessClassifier()
gauss.fit(X_train, y_train)
```

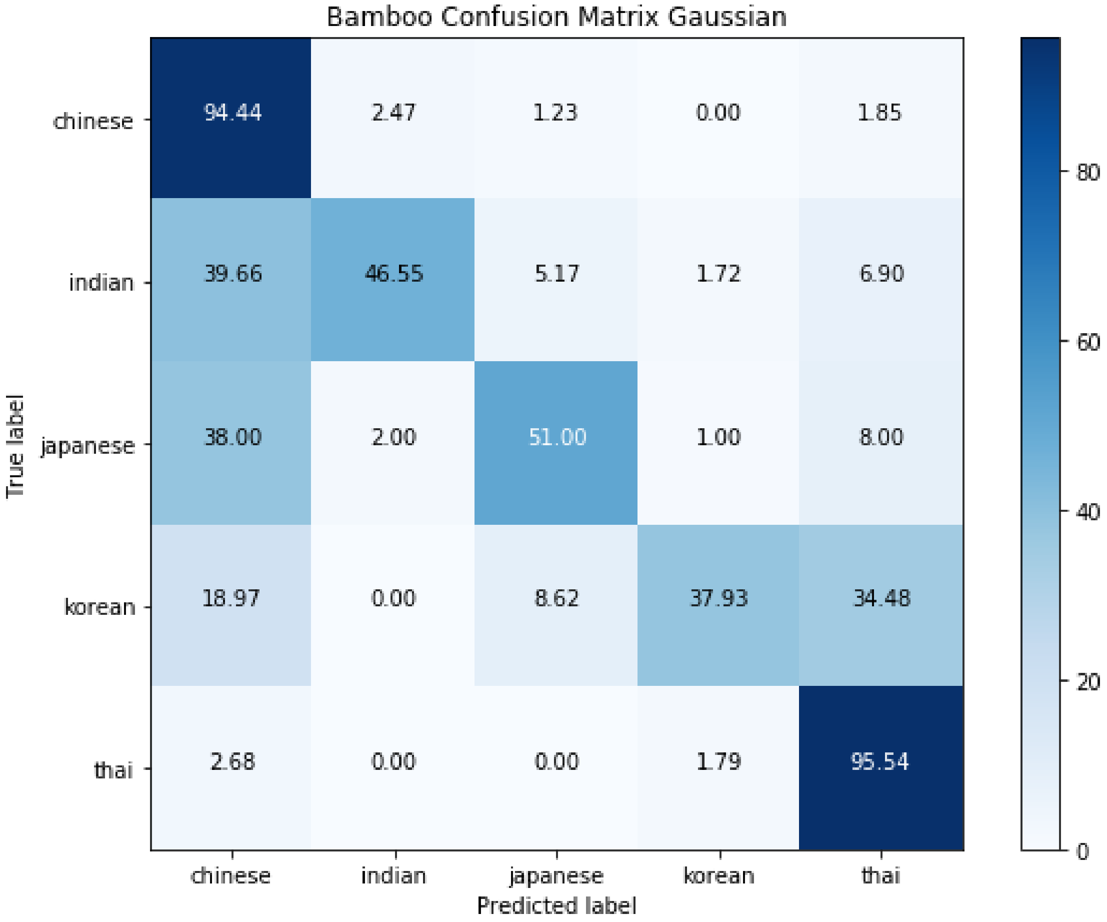
[29]:

```
GaussianProcessClassifier(copy_X_train=True, kernel=None, max_iter_predict=100,
                          multi_class='one_vs_rest', n_jobs=None,
                          n_restarts_optimizer=0, optimizer='fmin_l_bfgs_b',
                          random_state=None, warm_start=False)
```

[30]:

```
recipes_pred_gauss = gauss.predict(X_test)
```

[31]:
```
confusion(recipes_pred_gauss, "Gaussian")
```



[32]:
```
bamboo_confusion_matrix = confusion_matrix(y_test, recipes_pred_gauss, labels)
evaluation["GaussianProcessClassifier"] = bamboo_confusion_matrix.diagonal().
 ↪mean()
```

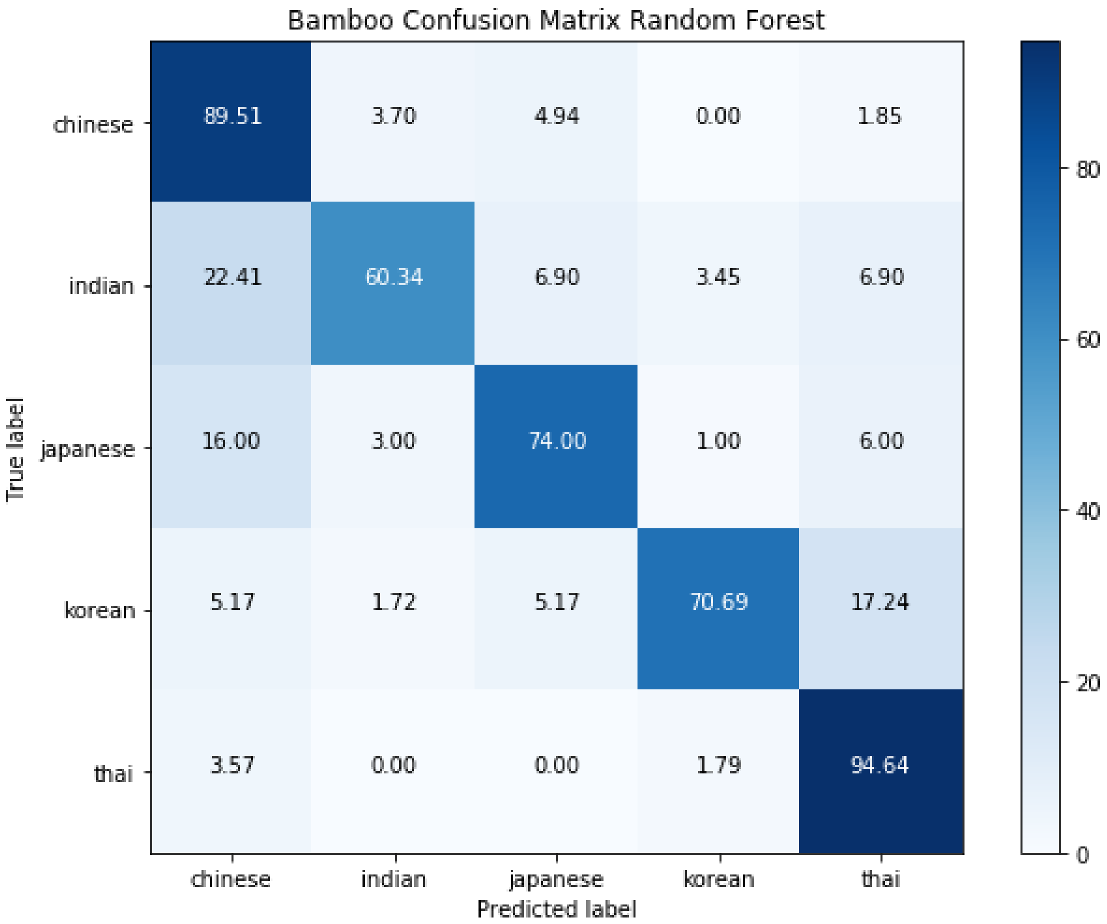
5 RandomForestClassifier

[43]:
```
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
forest = RandomForestClassifier(max_depth=21, random_state=42)
forest.fit(X_train, y_train)
```

[43]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=21, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=42, verbose=0,
                       warm_start=False)
```

```
[44]: recipes_pred_forest = forest.predict(X_test)
```

```
[45]: confusion(recipes_pred_forest, "Random Forest")
```



```
[46]: bamboo_confusion_matrix = confusion_matrix(y_test, recipes_pred_forest, labels)
      evaluation["RandomForestClassifier"] = bamboo_confusion_matrix.diagonal().mean()
```

6 AdaBoostClassifier

[48]:

```
ada = AdaBoostClassifier(random_state=42)
ada.fit(X_train, y_train)
```

[48]:

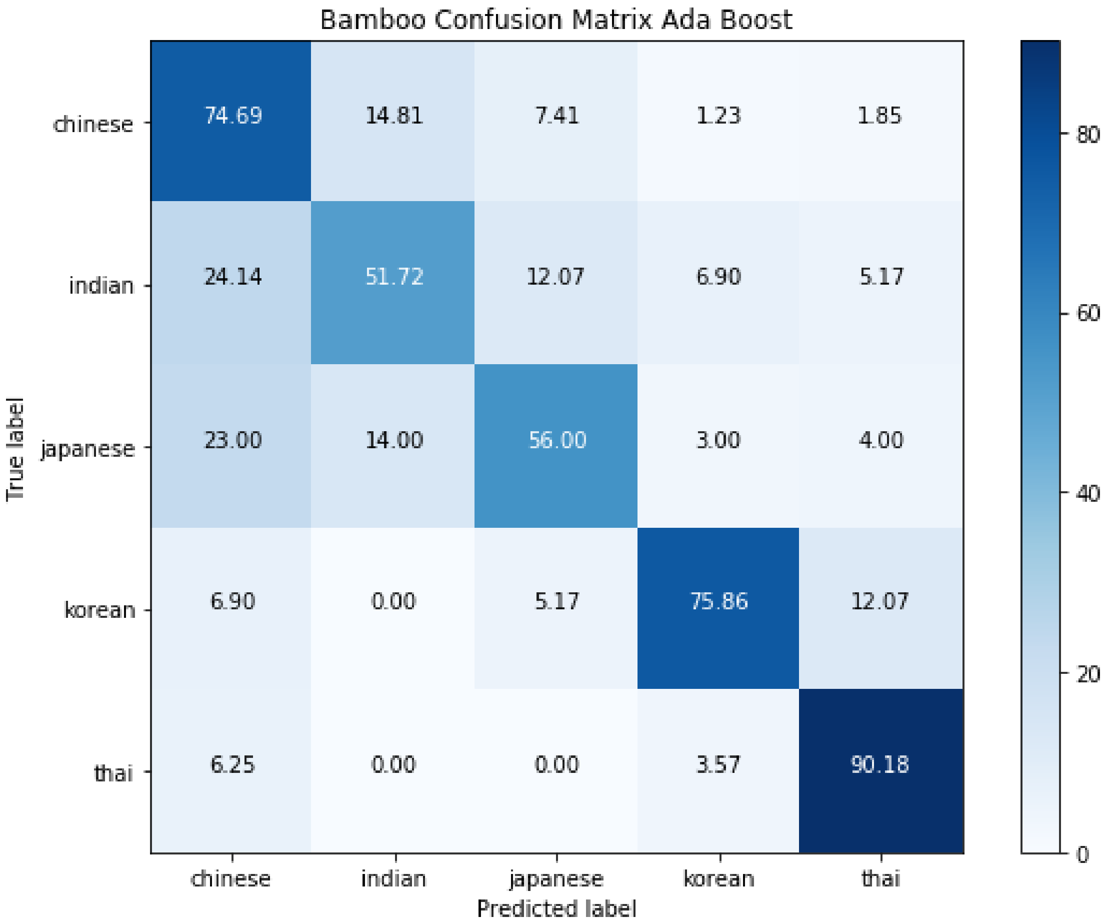
```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0,
                   n_estimators=50, random_state=42)
```

[49]:

```
recipes_pred_ada = ada.predict(X_test)
```

[51]:

```
confusion(recipes_pred_ada, "Ada Boost")
```



[52]:

```
bamboo_confusion_matrix = confusion_matrix(y_test, recipes_pred_ada, labels)
evaluation["AdaBoostClassifier"] = bamboo_confusion_matrix.diagonal().mean()
```

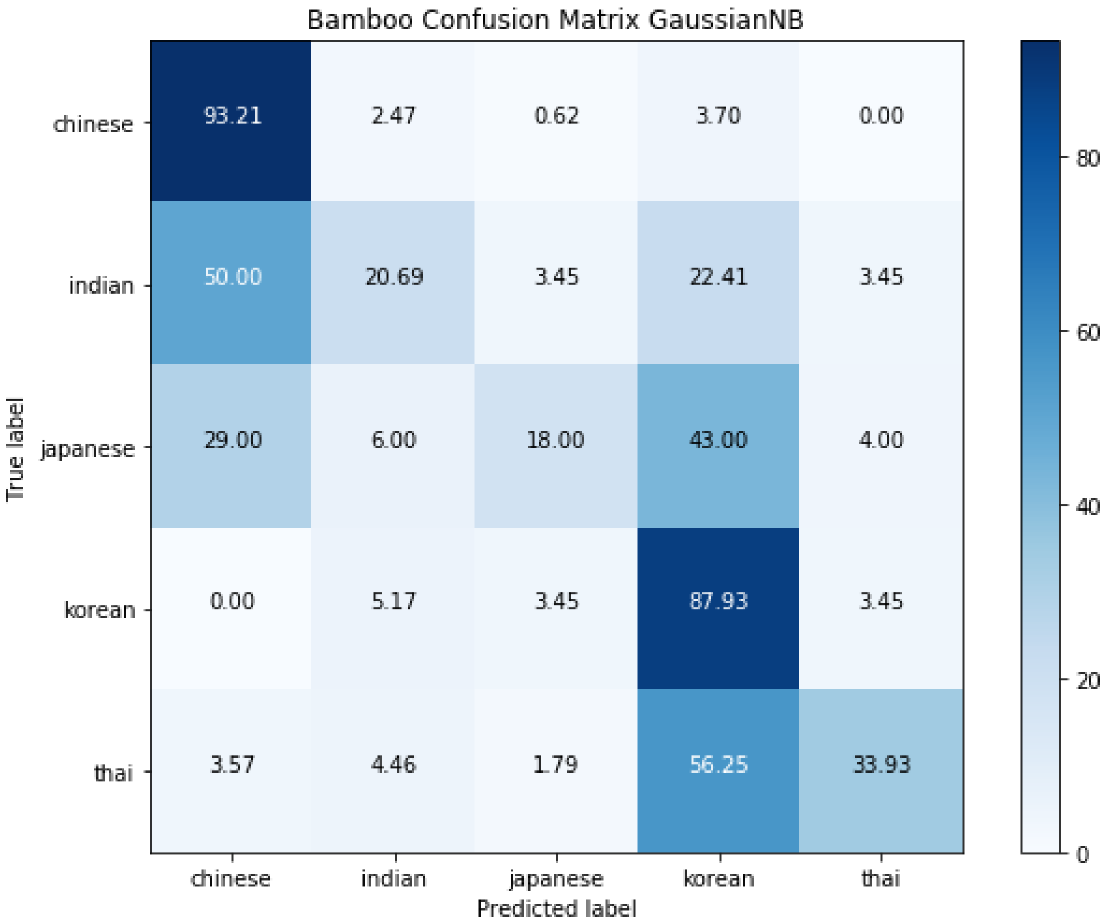
7 GaussianNB

[53]:
```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train, y_train)
```

[53]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

[54]:
```
recipes_pred_NB = nb.predict(X_test)
```

[56]:
```
confusion(recipes_pred_NB, "GaussianNB")
```



[58]:
```
bamboo_confusion_matrix = confusion_matrix(y_test, recipes_pred_NB, labels)
evaluation["GaussianNB"] = bamboo_confusion_matrix.diagonal().mean()
```

8 QuadraticDiscriminantAnalysis

[59]:

```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
QDA = QuadraticDiscriminantAnalysis()
QDA.fit(X_train, y_train)
```

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

[59]:

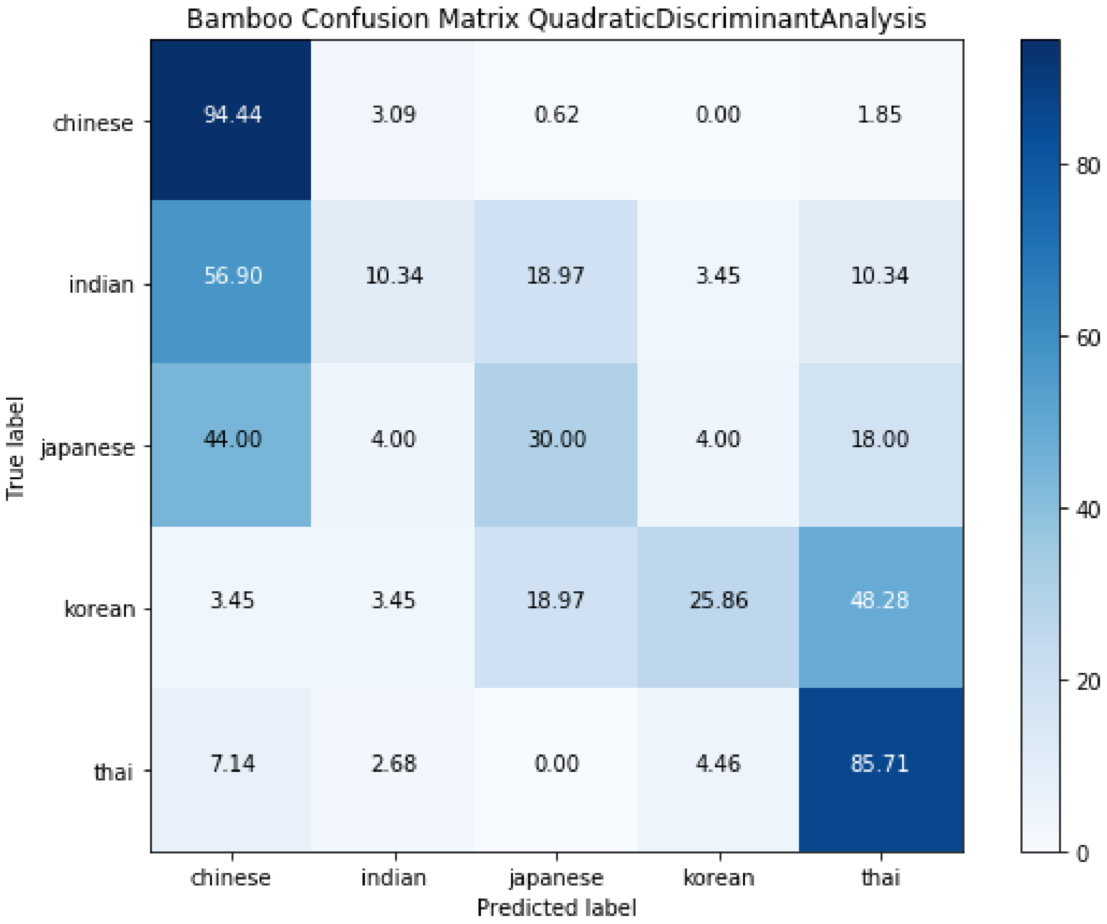
```
QuadraticDiscriminantAnalysis(priors=None, reg_param=0.0,
                              store_covariance=False, tol=0.0001)
```

[60]:

```
recipes_pred_QDA = QDA.predict(X_test)
```

[61]:

```
confusion(recipes_pred_QDA, "QuadraticDiscriminantAnalysis")
```



```
[74]: bamboo_confusion_matrix = confusion_matrix(y_test, recipes_pred_QDA, labels)
      evaluation["QuadraticDiscriminantAnalysis"] = bamboo_confusion_matrix.
       ↪diagonal().mean()
```

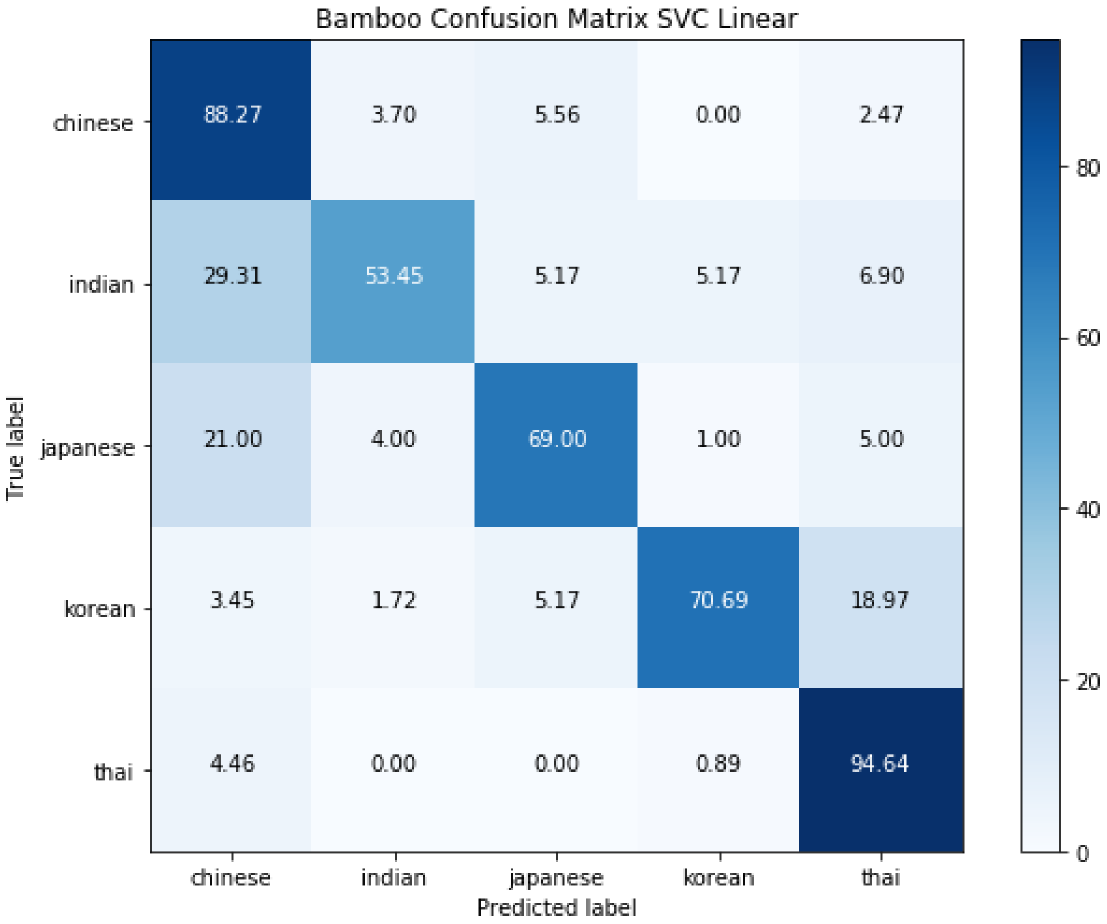
9 SVC Linear

```
[63]: from sklearn.svm import SVC
      svc = SVC(kernel="linear", C=0.025)
      svc.fit(X_train, y_train)
```

```
[63]: SVC(C=0.025, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
          decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
          max_iter=-1, probability=False, random_state=None, shrinking=True,
          tol=0.001, verbose=False)
```

```
[64]: recipes_pred_svc = svc.predict(X_test)
```

```
[65]: confusion(recipes_pred_svc, "SVC Linear")
```



[73]:

```
bamboo_confusion_matrix = confusion_matrix(y_test, recipes_pred_svc, labels)
evaluation["SVC Linear"] = bamboo_confusion_matrix.diagonal().mean()
```

10 RBF SVC

[70]:

```
Rsvc = SVC(kernel="poly",gamma=2, C=1)
Rsvc.fit(X_train, y_train)
```

[70]:

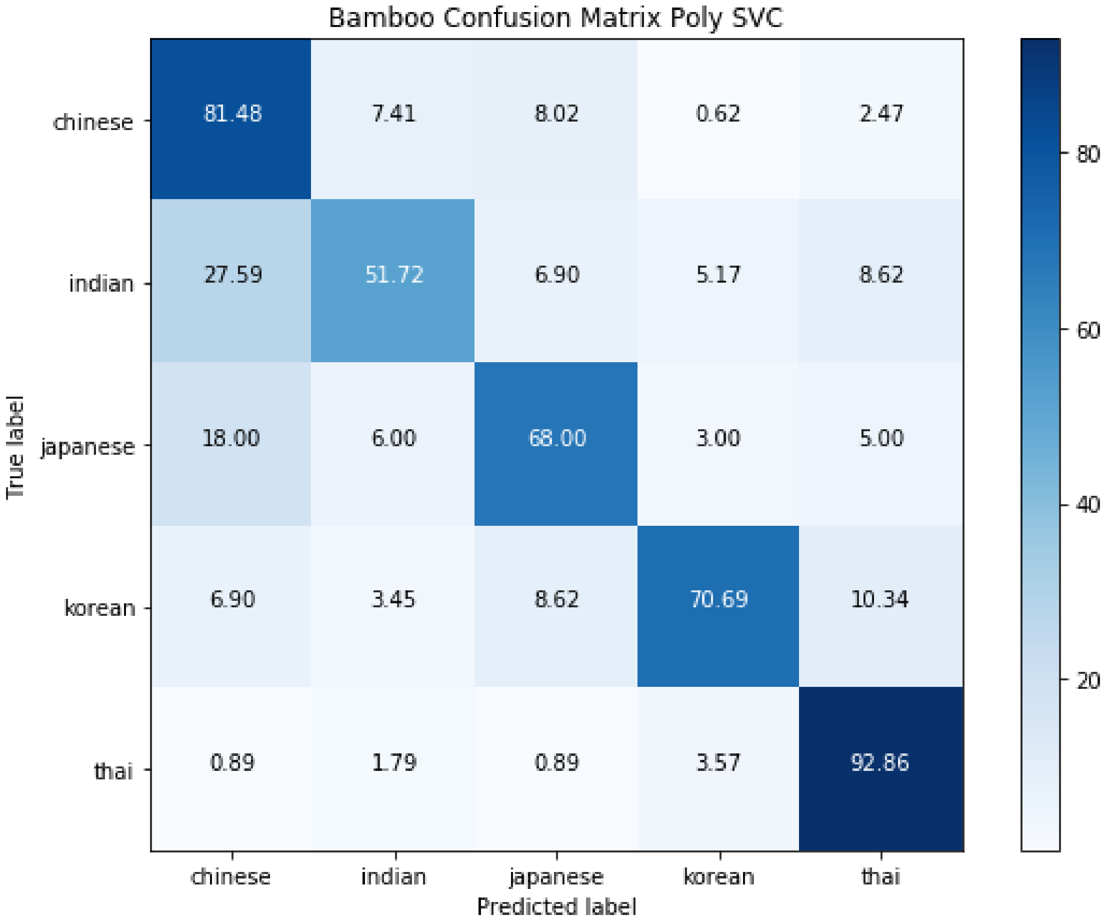
```
SVC(C=1, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=2, kernel='poly',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

[71]:

```
recipes_pred_Rsvc = Rsvc.predict(X_test)
```

[75]:

```
confusion(recipes_pred_Rsvc, "Poly SVC")
```



[76]:
```
bamboo_confusion_matrix = confusion_matrix(y_test, recipes_pred_Rsvc, labels)
evaluation["SVC Poly"] = bamboo_confusion_matrix.diagonal().mean()
```

[80]:
```
comparison = pd.Series(evaluation)
```

[83]:
```
comparison
```

[83]:
```
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
dtype: float64
```

[ ]:
```

```