Reporte Weka y Jupyter Notebook | Inteligencia Artificial

Marco Ferraro | B82957

Grabiel Revillat | B86524

Steven Nuñez | B95614

Enunciado

1. Vea el video de SVM usando Weka que se agregó.

2. Durante el tiempo de clase (o antes si lo desean) ustedes llevarán a cabo en grupo (con su equipo de trabajo) lo siguiente:

> a) Seleccionarán un archivo de datos .csv creado para la Tarea de Naive Bayes por alguno de los tres integrantes del grupo.
>
> b) Siguiendo la información del video, habilitarán LibSVM en Weka para crear un modelo con la colección de Trainning y Evaluarán sobre la de Testing que eligieron.
>
> c) Comparen los resultados de: Naive Bayes contra SVM (kernel lineal, radial y poli), ajusten los hiperparámetros vistos en la presentación de la clase del martes antes de evaluar sobre el data set de testing. Reporten tanto la calidad del modelo como los resultados de evaluación sobre el testing.

3. Usando Jupiter Notebook carguen los datos de training y usando el cuaderno visto el martes, modifíquenlo para evaluar el data set de testing con los mismos hiperparámetros que usaron con Weka.

4. Deben hacer una síntesis y un análisis de los resultados obtenidos en general.

Bayes y Máquinas de Soporte Vectorial en Weka

Para este trabajo se utiliza el set de datos de películas y su país de origen. El *target variable* es USA o Germany.

Para la primera parte de esta tarea, volvimos a probar el modelo de Bayes en los datos de prueba y entrenamiento.

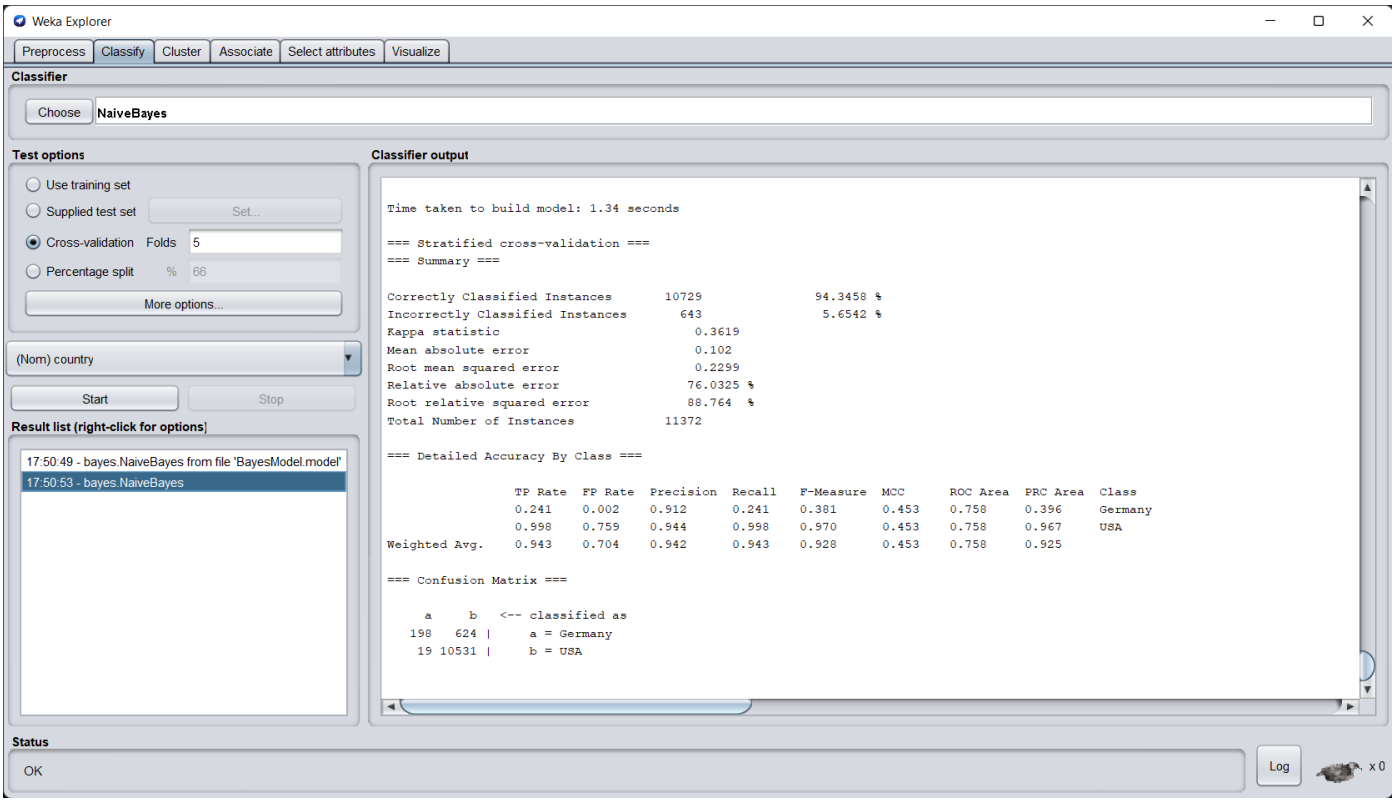

Ilustración 1 Resultados Training con 5 Cross Folds Bayes

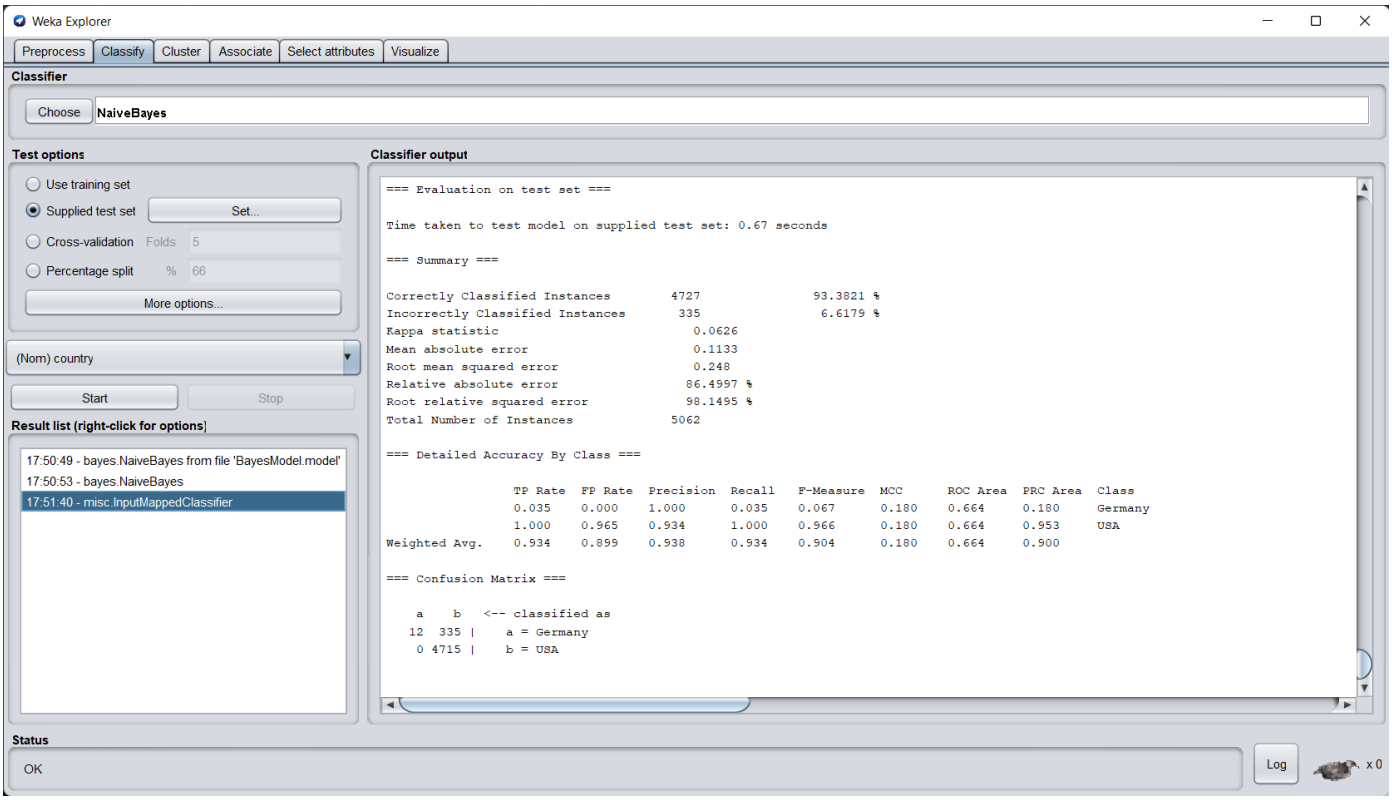

Ilustración 2 Resultados Testing con el modelo de Bayes de Ilustración 1

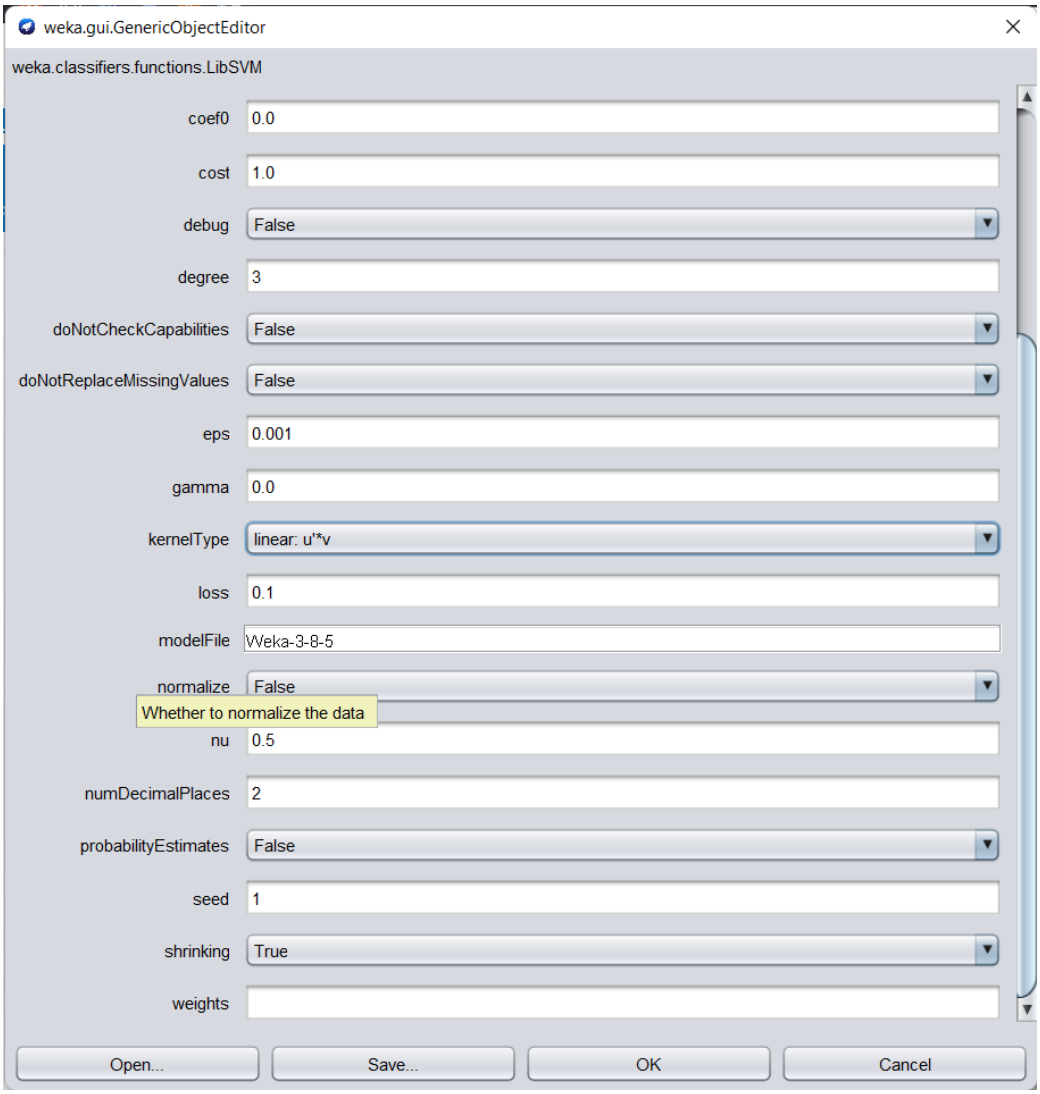

Ilustración 3 Definición SVM con kernel lineal

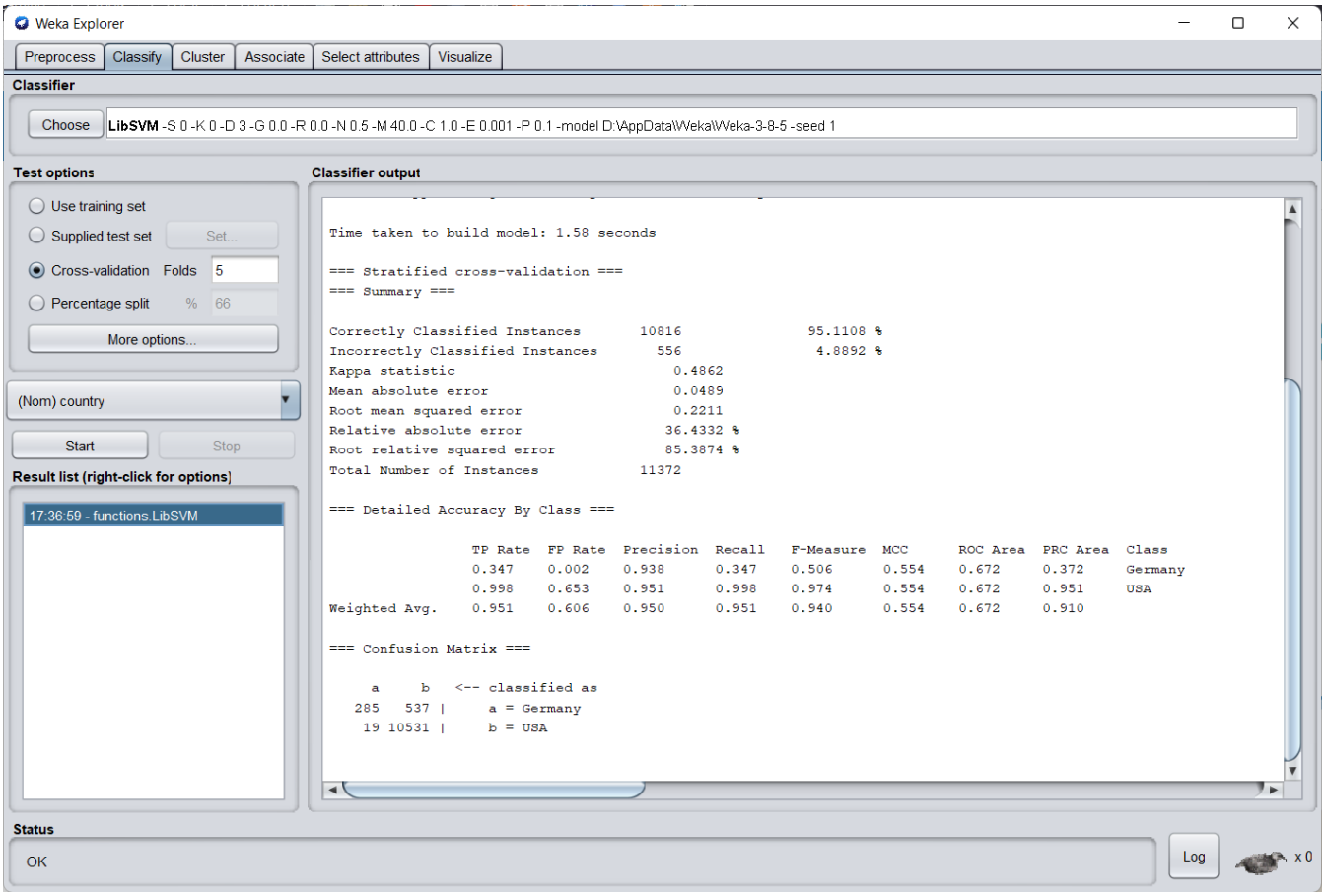

Ilustración 4 Resultados Training con 5 Cross Folds kernel lineal

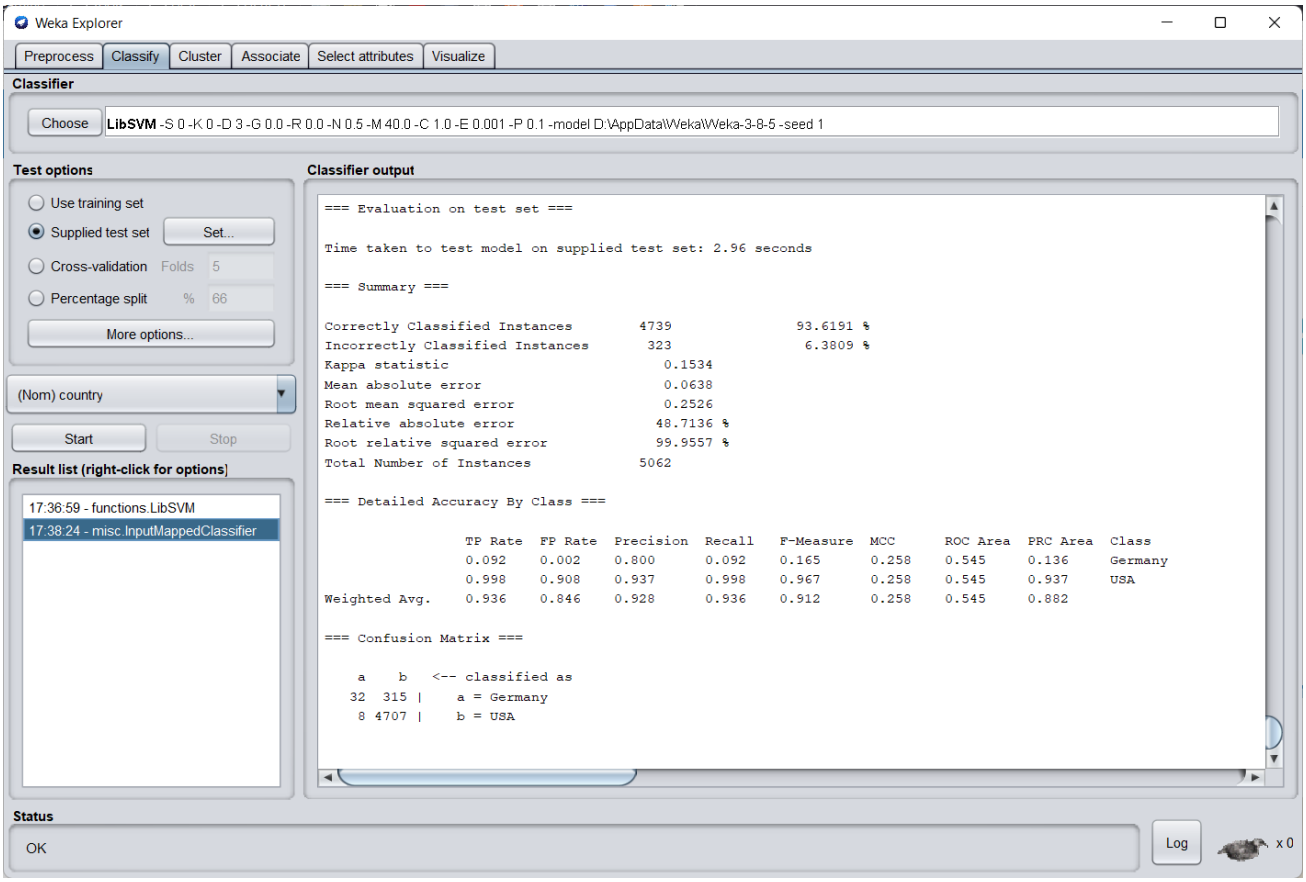

Ilustración 5 Resultados Testing kernel lineal

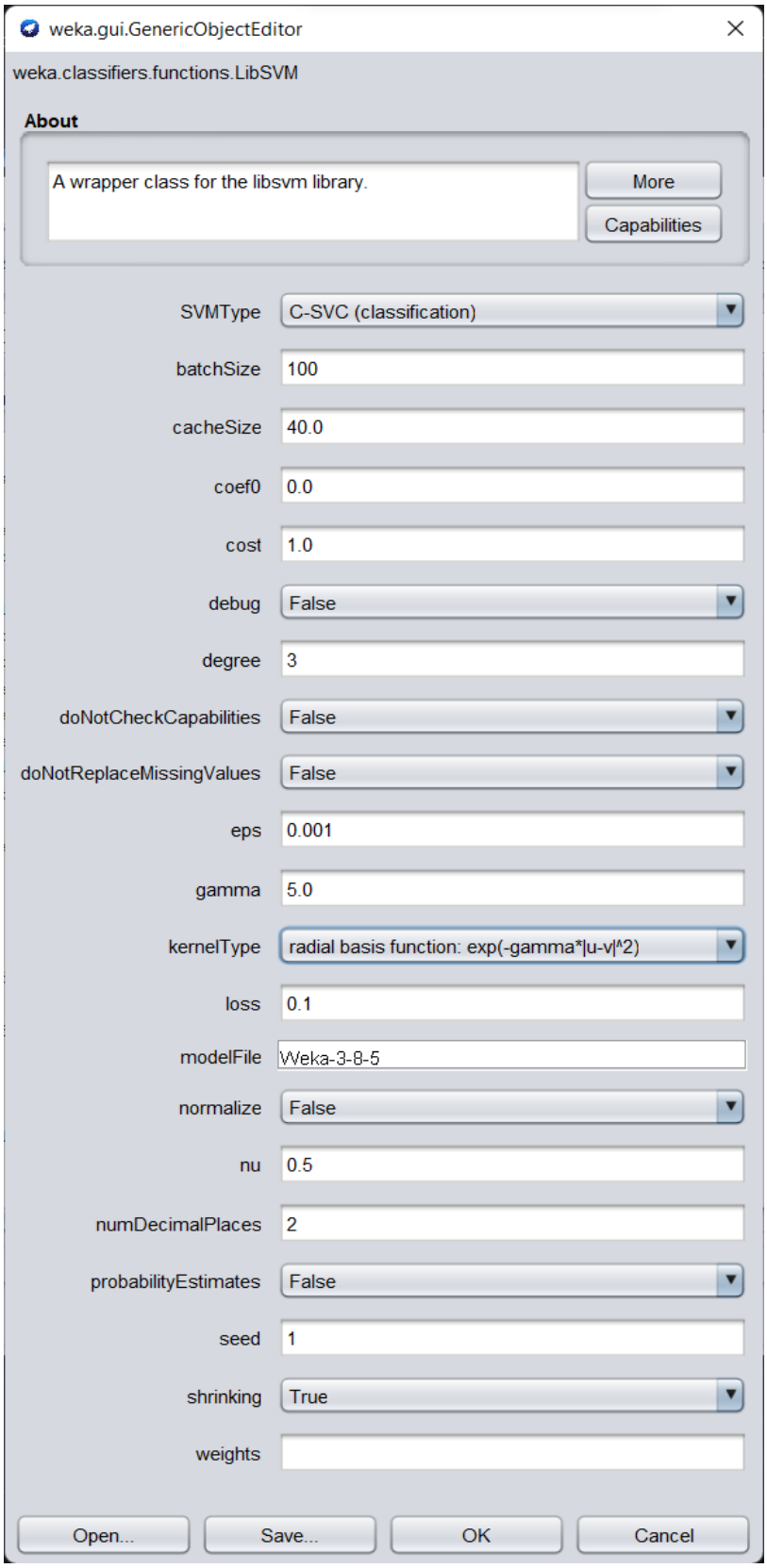

Ilustración 6 Definición SVM con kernel radial y gamma 5

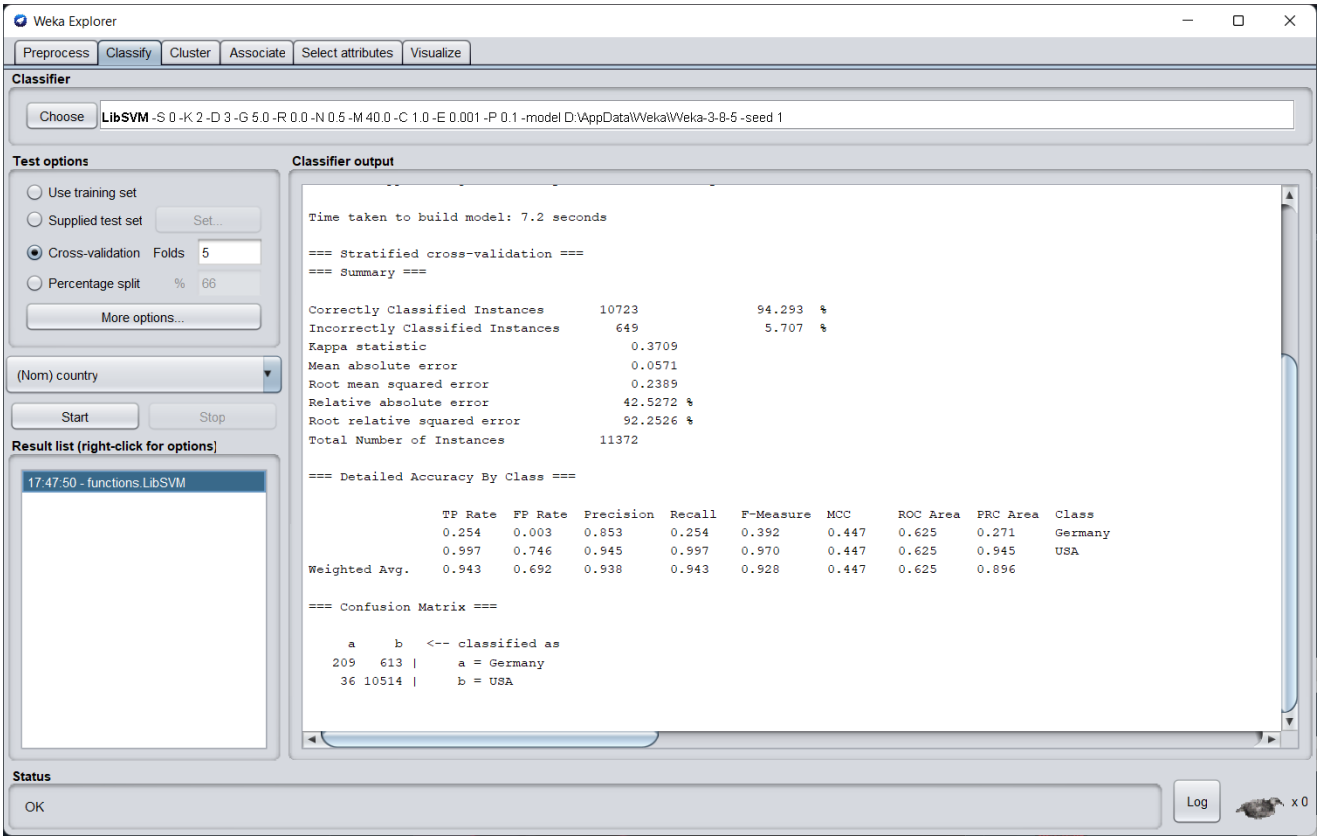

Ilustración 7 Resultados Training con 5 Cross Folds kernel radial

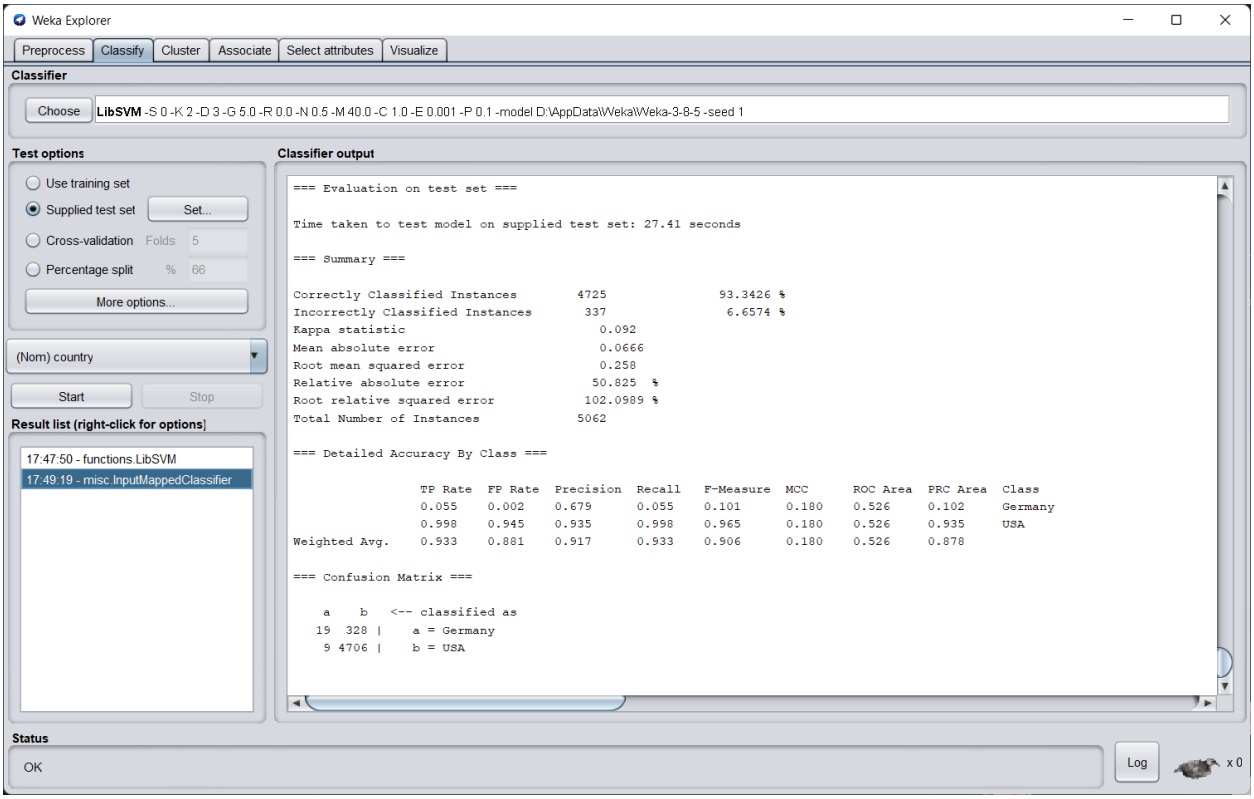

Ilustración 8 Resultados Testing con kernel radial

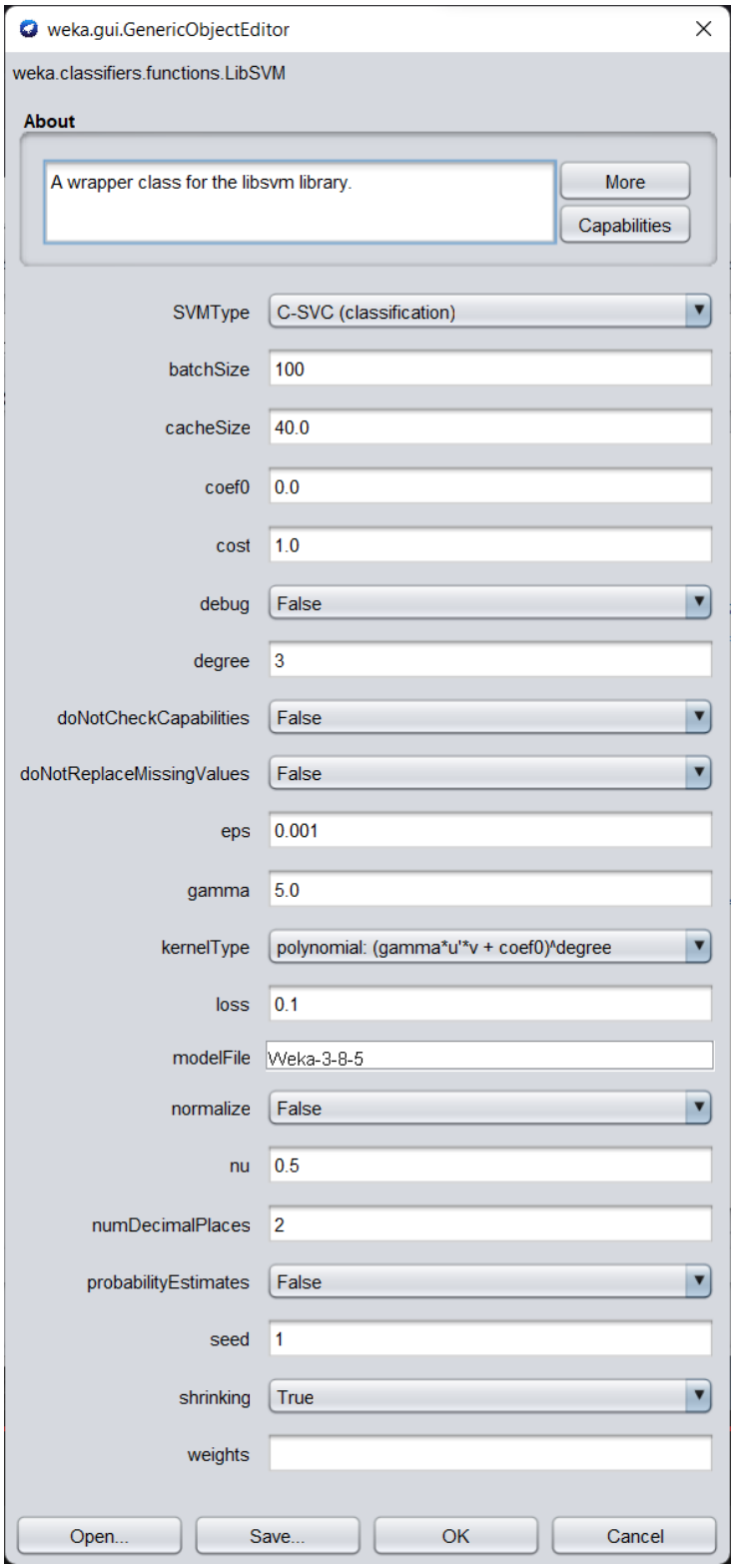

Ilustración 9 Definicion SVM con kernel polinomial y gamma 5

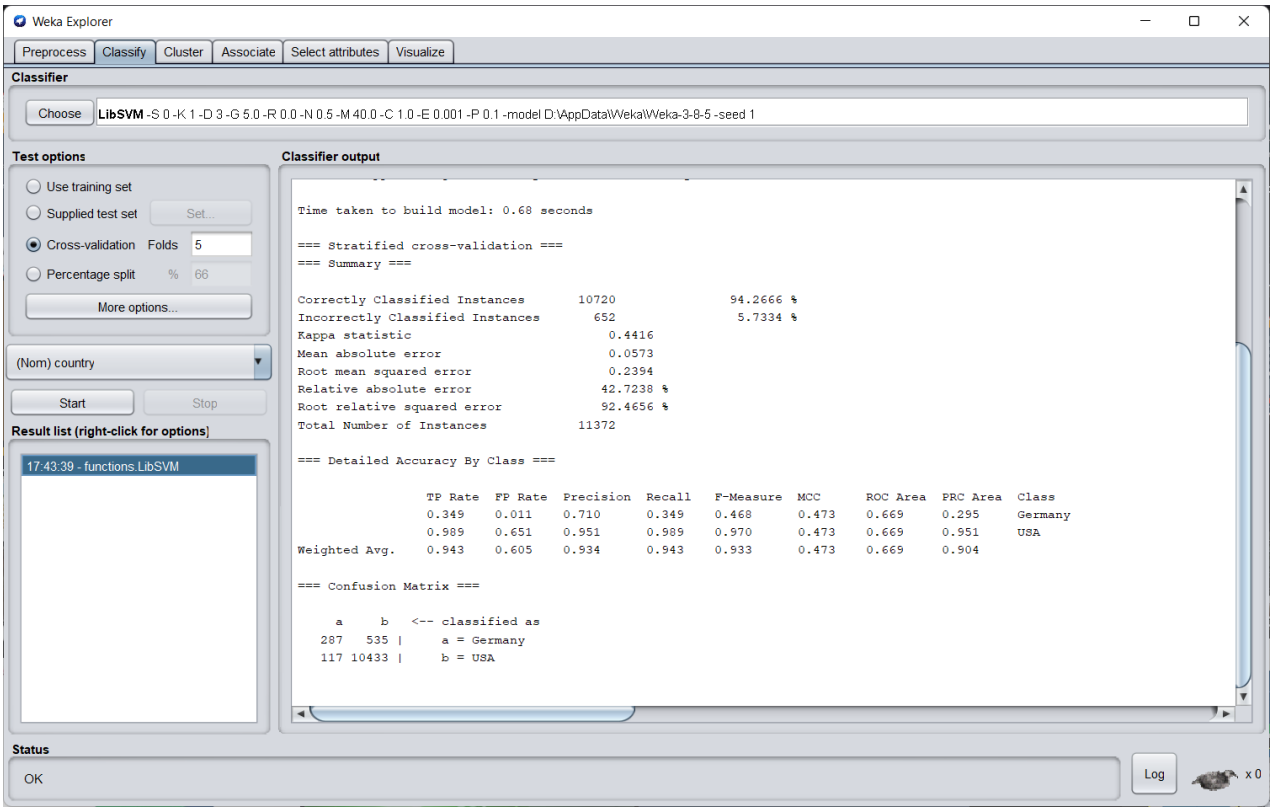

Ilustración 10 Resultados training con kernel polinomial

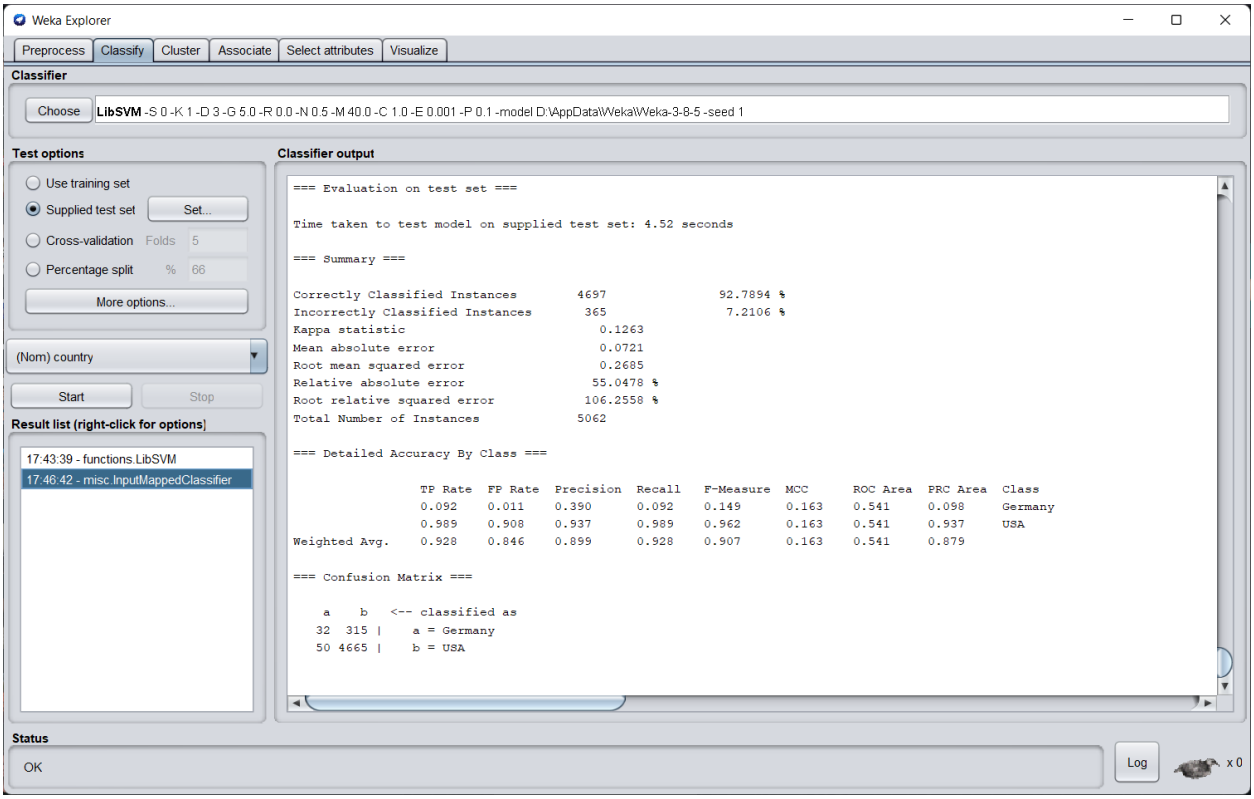

Ilustración 11 Resultados Testing con kernel polinomial

```
In [14]: # Ligar el dataset con el clasificador
         SVM = svm.SVC(C=1, kernel='linear', degree=3, gamma='auto')
         SVM.fit(Train_X_Tfidf,Train_Y)
         # Hacer las predicciones con el conjunto de prueba
         predictions_SVM = SVM.predict(Test_X_Tfidf)
```

Ilustración 12 Definicion kernel lineal en Python

```
In [18]: # Ligar el dataset con el clasificador
         SVM = svm.SVC(C=1, kernel='rbf', degree=3, gamma='scale')
         SVM.fit(Train_X_Tfidf,Train_Y)
         # Hacer las predicciones con el conjunto de prueba
         predictions_SVM = SVM.predict(Test_X_Tfidf)
```

Ilustración 13 Definicion kernal radial en Python

In [16]:
```
# Ligar el dataset con el clasificador
SVM = svm.SVC(C=1, kernel='poly', degree=3, gamma='scale')
SVM.fit(Train_X_Tfidf,Train_Y)
# Hacer las predicciones con el conjunto de prueba
predictions_SVM = SVM.predict(Test_X_Tfidf)
```

Ilustración 14 Definicion kernel polinomial en Python

In [15]:
```
print("SVM Accuracy Score -> ", skm.accuracy_score(Test_Y, predictions_SVM)*100)
print("SVM Recall ", etiqueta0, skm.recall_score(Test_Y, predictions_SVM, pos_label=0))
print("SVM Recall ", etiqueta1, skm.recall_score(Test_Y, predictions_SVM, pos_label=1))
print("SVM Precision  ", etiqueta0, skm.precision_score(Test_Y, predictions_SVM,pos_label=0))
print("SVM Precision  ", etiqueta1, skm.precision_score(Test_Y, predictions_SVM,pos_label=1))
print("SVM F  ", etiqueta0, skm.f1_score(Test_Y, predictions_SVM,pos_label=0))
print("SVM F  ", etiqueta1, skm.f1_score(Test_Y, predictions_SVM,pos_label=1))
print("SVM Confusion Table: \n", skm.confusion_matrix(Test_Y, predictions_SVM))
```

```
SVM Accuracy Score ->  94.15250888976689
SVM Recall  Germany 0.15273775216138327
SVM Recall  USA 0.999575821845175
SVM Precision   Germany 0.9636363636363636
SVM Precision   USA 0.9412822049131216
SVM F   Germany 0.26368159203980096
SVM F   USA 0.9695535897963382
SVM Confusion Table:
 [[  53  294]
 [   2 4713]]
```

Ilustración 15 Resultados kernel lineal en Python

In [19]:
```
print("SVM Accuracy Score -> ", skm.accuracy_score(Test_Y, predictions_SVM)*100)
print("SVM Recall ", etiqueta0, skm.recall_score(Test_Y, predictions_SVM, pos_label=0))
print("SVM Recall ", etiqueta1, skm.recall_score(Test_Y, predictions_SVM, pos_label=1))
print("SVM Precision  ", etiqueta0, skm.precision_score(Test_Y, predictions_SVM,pos_label=0))
print("SVM Precision  ", etiqueta1, skm.precision_score(Test_Y, predictions_SVM,pos_label=1))
print("SVM F  ", etiqueta0, skm.f1_score(Test_Y, predictions_SVM,pos_label=0))
print("SVM F  ", etiqueta1, skm.f1_score(Test_Y, predictions_SVM,pos_label=1))
print("SVM Confusion Table: \n", skm.confusion_matrix(Test_Y, predictions_SVM))
```

```
SVM Accuracy Score ->  93.71789806400632
SVM Recall  Germany 0.0893371757925072
SVM Recall  USA 0.999575821845175
SVM Precision   Germany 0.9393939393939394
SVM Precision   USA 0.9371644462119706
SVM F   Germany 0.16315789473684209
SVM F   USA 0.9673645320197044
SVM Confusion Table:
 [[  31  316]
 [   2 4713]]
```

Ilustración 16 Resultados kernel radial en Python

```
In [17]: print("SVM Accuracy Score -> ", skm.accuracy_score(Test_Y, predictions_SVM)*100)
         print("SVM Recall ", etiqueta0, skm.recall_score(Test_Y, predictions_SVM, pos_label=0))
         print("SVM Recall ", etiqueta1, skm.recall_score(Test_Y, predictions_SVM, pos_label=1))
         print("SVM Precision  ", etiqueta0, skm.precision_score(Test_Y, predictions_SVM,pos_label=0))
         print("SVM Precision  ", etiqueta1, skm.precision_score(Test_Y, predictions_SVM,pos_label=1))
         print("SVM F  ", etiqueta0, skm.f1_score(Test_Y, predictions_SVM,pos_label=0))
         print("SVM F  ", etiqueta1, skm.f1_score(Test_Y, predictions_SVM,pos_label=1))
         print("SVM Confusion Table: \n", skm.confusion_matrix(Test_Y, predictions_SVM))

SVM Accuracy Score ->  93.16475701303833
SVM Recall  Germany 0.005763688760806916
SVM Recall  USA 0.9997879109225875
SVM Precision   Germany 0.6666666666666666
SVM Precision   USA 0.9318047044870528
SVM F   Germany 0.011428571428571429
SVM F   USA 0.9645999590750972
SVM Confusion Table:
 [[   2  345]
 [   1 4714]]
```

Ilustración 17 Resultados kernel polinomial en Python

Conclusiones

Para analizar los resultados hay que tener claro que las maquinas de Soporte Vectorial en Jupyter y en Weka tienen que analizarse sin relacionar los resultados entre estas mismas, ya que, al ser Máquinas de Soporte Vectorial diferentes representan una variable de bloqueo.

Análisis Weka

En los rendimientos del modelo, entre las Maquinas de Soporte Vectorial vemos que la que tiene mejor rendimiento es la de kernel lineal, ya que tiene más precisión para predecir las *X* que pertenecen a **Germany.** Dicho esto, hay algo peculiar; por alguna razon, el modelo de Naive Bayes tiene precisión del 100 en el *target variable* **Germany.** Podemos decir que es por la naturaleza de los datos, pero es algo que llama poderosamente la atención, y mas porque acierta todos los *True Negative*.

Análisis Jupyter

Para trabajar esta parte, se modificó el código de Python para respetar la separación de los datos de prueba y de entrenamiento. No se separan con los métodos de *sci-kit learn*, si no que se importan como csv aparte (el notebook viene adjunto con esta tarea). Dentro de los resultados que genera *sci-kit learn* vemos que el kernel lineal tiene un mejor resultado, pero marginalmente. De conclusión, para analizar datos con 1 *X* y *1 target variable* con maquinas de soporte vectorial recomendamos utilizar el kerneal líneal, ya que no es tan complejo y parece que su umbral de decisión es más preciso que los otros kernel.