GRUPO: Jesus Martin y Jorve Arevalo G13

Tercera parte. Recomendacion basada en filtrado colaborativo.

En esta tercera parte utilizaremos la librería SURPRISE Se puede consultar la documentacion en http://surpriselib.com/ (http://surpriselib.com/)

Para instalarla: conda install -c conda-forge scikit-surprise o pip install numpy pip install scikit-surprise

La librería SurPRISE (Simple Python RecommendatIon System Engine) tiene algoritmos de predición de ratings incluidos: baseline algorithms, neighborhood methods, matrix factorization-based (SVD, PMF, SVD++, NMF) y otros. También tiene predefinidas las medidas de similitud mas comunes sobre vectores (cosine, MSD, pearson…) Una de las cosas más utiles es que proporciona herramientas para evaluar, analizar y comparar el rendimiento de distitnos algoritmos. Lo que vamos a hacer en esta parte de la práctica es probar varios procedimientos de evaluación cruzada midiendo datos sobre errores entre el valor real (conocido) y la predicción del recomendador. Las siglas corresponden a las siguientes medidas:

MAE: Mean Absolute Error RMSE: Root mean square error (RMSE) MSE: mean square error is defined as the expected value of the square of the difference between the estimator and the parameter. -square root of the mean square error.

Vamos a ejecutar algunos recomendadores y evaluarlos para poder comentar los resultados.

In [1]:
```
conda install -c conda-forge scikit-surprise
```

```
Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... done

# All requested packages already installed.


Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
from collections import defaultdict
import numpy as np

from surprise import KNNBasic
from surprise import KNNWithMeans
from surprise import KNNWithZScore
from surprise import KNNBaseline

from surprise import Dataset
from surprise import accuracy
from surprise.model_selection import train_test_split
```

In [4]:
```
## Ejemplo getting started de la documentación de SURPRISE
##http://surpriselib.com/

from surprise import SVD
from surprise import Dataset
from surprise.model_selection import cross_validate

# Load the movielens-100k dataset (download it if needed).
data = Dataset.load_builtin('ml-100k')

# Use the famous SVD algorithm.
algo = SVD()

# Run 5-fold cross-validation and print results.
cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9345  0.9319  0.9424  0.9311  0.9343  0.9348  0.0040
MAE (testset)     0.7363  0.7362  0.7433  0.7346  0.7366  0.7374  0.0030
Fit time          4.98    4.81    5.12    5.34    4.82    5.02    0.20
Test time         0.13    0.10    0.26    0.13    0.13    0.15    0.05
```

Out[4]:
```
{'test_rmse': array([0.93446693, 0.93190033, 0.94235697, 0.93111614, 0.9343024
5]),
 'test_mae': array([0.73628176, 0.73616719, 0.74330182, 0.73455085, 0.7366320
5]),
 'fit_time': (4.98023247718811,
  4.812105894088745,
  5.1193273067474365,
  5.342425107955933,
  4.821113586425781),
 'test_time': (0.13358497619628906,
  0.10173249244689941,
  0.25531744956970215,
  0.13065147399902344,
  0.1296536922454834)}
```

In [5]:
```
# Evaluacion extracted from surprise:
# https://surprise.readthedocs.io/en/stable/FAQ.html#how-to-compute-precision-k-and-recall-k
def measures_at_k(predictions, k, th_recom, th_relev):
    '''Return precision and recall at k metrics for each user.'''

    # First map the predictions to each user.
    user_est_true = defaultdict(list)
    for uid, _, true_r, est, _ in predictions:
        user_est_true[uid].append((est, true_r))

    precisions = dict()
    recalls = dict()
    onehits = dict()
    mrr = dict()

    for uid, user_ratings in user_est_true.items():

        # Sort user ratings by estimated value
        user_ratings.sort(key=lambda x: x[0], reverse=True)

        # Number of relevant items
        n_rel = sum((true_r >= th_relev) for (_, true_r) in user_ratings)

        # Number of recommended items in top k
        n_rec_k = sum((est >= th_recom) for (est, _) in user_ratings[:k])

        # Number of relevant and recommended items in top k
        n_rel_and_rec_k = sum(((true_r >= th_relev) and (est >= th_recom))
                              for (est, true_r) in user_ratings[:k])

        # Precision@K: Proportion of recommended items that are relevant
        precisions[uid] = n_rel_and_rec_k / n_rec_k if n_rec_k != 0 else 0

        # Recall@K: Proportion of relevant items that are recommended
        recalls[uid] = n_rel_and_rec_k / n_rel if n_rel != 0 else 0


    return precisions, recalls
```

In [6]:
```
def f1(precision, recall):
    """
        Funcion que calcula el f1 (media armónica) en funcion de precision y recall
    """
    denominador = precision + recall

    if denominador == 0:
        return 0
    else:
        return (2 * precision * recall) / denominador
```

In [7]:
```
def get_results(recommendations, k, knn):
    """
        Function to get the measures results
    """
    # threshold = 4 --> solo se tienen en cuenta peliculas que hayan
    # sido puntuadas con 4 o 5 estrellas
    precisions, recalls  = measures_at_k(recommendations, k, th_recom=4, th_relev
=1)

    # Measures can then be averaged over all users
    precision_result = sum(prec for prec in precisions.values()) / len(precisions
)
    recall_result = sum(rec for rec in recalls.values()) / len(recalls)
    # Media armónica
    f1_result = f1(precision_result, recall_result)
    # En este archivo se volcarán los resultados de las métricas. Tiene que exist
ir.
    #f = open("C:/Users/user/practicas/P2/results_user_cf.csv", 'a')
    f = open("C:/hlocal/results_user_cf.csv", 'a')
    f.write(str(k) + ',' + knn + "," + str(precision_result) + ',' + str(recall_r
esult) + ',' +  str(f1_result) +  '\n')
    f.close()
```

In [8]:
```
# Hemos cargado antes los datos de movieLens para 100K
# data = Dataset.load_builtin('ml-100k')
```

In [9]:
```
# creo dos conjuntos de datos, el training set y el evaluation set
# cada uno contendra la mitad de los datos
training_set, evaluation_set = train_test_split(data, test_size=.5)
```

In [10]:
```
# Ahora determino cual es el algoritmo que voy a usar de recomendacion
# en este caso voy a usar el algoritmo KNN para encontrar las similitudes entre i
tems
recommendation_algorithm = KNNBasic(k=100, sim_options={'name': 'pearson_baselin
e', 'user_based': True})
print(recommendation_algorithm)
```

```
<surprise.prediction_algorithms.knns.KNNBasic object at 0x000001BD2120ACA0>
```

In [11]:
```
# aplico el algoritmo sobre el training_set
recommendation_algorithm.fit(training_set)
print(recommendation_algorithm)
```

```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
<surprise.prediction_algorithms.knns.KNNBasic object at 0x000001BD2120ACA0>
```

In [12]:
```
# aplico el algoritmo sobre el evaluation set y obtengo las predicciones en las r
ecomendaciones
recommendations = recommendation_algorithm.test(evaluation_set)
#print(recommendations)
```

In [13]:
```
K = 10
for k in range(K):
    get_results(recommendations, k+1, "knn_basic")
```

In [14]:
```
#############################################################
# Hacer distintas pruebas con el resto de tipos KNN
recommendation_algorithm = KNNWithMeans(k=100, sim_options={'name': 'pearson_base
line', 'user_based': True})

# aplico el algoritmo sobre el training_set
recommendation_algorithm.fit(training_set)

# aplico el algoritmo sobre el evaluation set y obtengo las predicciones en las r
ecomendaciones
recommendations = recommendation_algorithm.test(evaluation_set)

K = 10
for k in range(K):
    get_results(recommendations, k+1, "knn_withmeans")
```

```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
```

In [15]:
```
#############################################################
# Hago lo mismo con el resto de tipos KNN
recommendation_algorithm = KNNWithZScore(k=100, sim_options={'name': 'pearson_bas
eline', 'user_based': True})

# aplico el algoritmo sobre el training_set
recommendation_algorithm.fit(training_set)

# aplico el algoritmo sobre el evaluation set y obtengo las predicciones en las r
ecomendaciones
recommendations = recommendation_algorithm.test(evaluation_set)

K = 10
for k in range(K):
    get_results(recommendations, k+1, "knn_withzscore")
```

```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
```

In [16]:
```
#############################################################
# Hago lo mismo con el resto de tipos KNN
recommendation_algorithm = KNNBaseline(k=100, sim_options={'name': 'pearson_basel
ine', 'user_based': True})

# aplico el algoritmo sobre el training_set
recommendation_algorithm.fit(training_set)

# aplico el algoritmo sobre el evaluation set y obtengo las predicciones en las r
ecomendaciones
recommendations = recommendation_algorithm.test(evaluation_set)

K = 10
for k in range(K):
    get_results(recommendations, k+1, "knn_baseline")
```

```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
```

Ejercicio: se pide ejecutar, comprender y escribir comentarios razonados sobre la evaluación de distintos recomendadores.

Prueba otros algoritmos de predicción de ratings basados en la estimación de los vecinos más próximos y realiza evaluaciones de su comportamiento. Comenta los resultados.¶ Puedes consultar la documentación en https://surprise.readthedocs.io/en/stable/knn_inspired.html# (https://surprise.readthedocs.io/en/stable/knn_inspired.html#)

In [18]:
```
from surprise import Dataset
from surprise.model_selection import cross_validate

prueba = KNNBasic(k=100, sim_options={'name': 'pearson_baseline', 'user_based': T
rue})


# Run 5-fold cross-validation and print results.
cross_validate(prueba, data, measures=['RMSE', 'MAE', 'MSE'], cv=10, verbose=True
)
```

```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.0073  1.0011  0.9930  1.0016  0.9978  1.0001  0.0047
MAE (testset)     0.7947  0.7924  0.7872  0.7923  0.7941  0.7921  0.0027
Fit time          1.79    1.69    1.74    2.17    1.69    1.82    0.18
Test time         4.16    4.65    4.49    4.57    4.68    4.51    0.19
```

Out[18]:
```
{'test_rmse': array([1.00730126, 1.0011089 , 0.99295073, 1.0015932 , 0.9977607
]),
 'test_mae': array([0.79467113, 0.79243139, 0.78715273, 0.79230279, 0.7940849
8]),
 'fit_time': (1.7920794486999512,
  1.6866772174835205,
  1.7432670593261719,
  2.174572229385376,
  1.6905815601348877),
 'test_time': (4.1642985343933105,
  4.648779392242432,
  4.487090110778809,
  4.56973934173584,
  4.679753541946411)}
```

In [19]:
```
from surprise import Dataset
from surprise.model_selection import cross_validate

prueba = KNNWithMeans(k=100, sim_options={'name': 'pearson_baseline', 'user_base
d': True})


# Run 5-fold cross-validation and print results.
cross_validate(prueba, data, measures=['RMSE', 'MAE', 'MSE'], cv=10, verbose=True
)
```

```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9314  0.9413  0.9437  0.9445  0.9294  0.9381  0.0064
MAE (testset)     0.7233  0.7333  0.7359  0.7372  0.7287  0.7317  0.0051
Fit time          1.82    2.28    1.81    2.45    1.75    2.02    0.29
Test time         4.58    4.74    5.02    5.14    5.01    4.90    0.21
```

Out[19]:
```
{'test_rmse': array([0.93141264, 0.94131789, 0.9437245 , 0.94451686, 0.9293577
5]),
 'test_mae': array([0.72331874, 0.73326945, 0.73592225, 0.7371603 , 0.7286807
5]),
 'fit_time': (1.8191378116607666,
  2.2847301959991455,
  1.8120849132537842,
  2.4504234790802,
  1.752119779586792),
 'test_time': (4.580800771713257,
  4.740548133850098,
  5.017637729644775,
  5.140260696411133,
  5.011681318283081)}
```

In [21]:
```
from surprise import Dataset
from surprise.model_selection import cross_validate

prueba = KNNWithZScore(k=100, sim_options={'name': 'pearson_baseline', 'user_base
d': True})


# Run 5-fold cross-validation and print results.
cross_validate(prueba, data, measures=['RMSE', 'MAE', 'MSE'], cv=10, verbose=True
)
```

```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE, MSE of algorithm KNNWithZScore on 10 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Fold 6  Fold 7  Fold 8
Fold 9  Fold 10 Mean    Std
RMSE (testset)    0.9119  0.9350  0.9321  0.9290  0.9269  0.9503  0.9276  0.9311
0.9294  0.9276  0.9301  0.0089
MAE (testset)     0.7095  0.7233  0.7257  0.7221  0.7225  0.7388  0.7209  0.7235
0.7233  0.7237  0.7233  0.0067
MSE (testset)     0.8315  0.8743  0.8688  0.8631  0.8592  0.9030  0.8605  0.8670
0.8638  0.8604  0.8652  0.0166
Fit time          2.18    2.00    2.02    2.01    2.03    2.01    1.99    2.01
2.07    3.14    2.15    0.34
Test time         3.32    3.79    3.26    3.51    3.32    3.28    3.44    3.32
3.20    2.14    3.26    0.40
```

Out[21]:

```
{'test_rmse': array([0.91188726, 0.93503703, 0.93208583, 0.92903157, 0.92692358,
       0.95028222, 0.92761649, 0.93114852, 0.92938843, 0.9275548 ]),
 'test_mae': array([0.70945147, 0.72327468, 0.72571297, 0.72214484, 0.72252321,
       0.73878922, 0.72091296, 0.72345803, 0.72326841, 0.72372332]),
 'test_mse': array([0.83153837, 0.87429425, 0.868784  , 0.86309967, 0.85918733,
       0.9030363 , 0.86047236, 0.86703757, 0.86376285, 0.86035791]),
 'fit_time': (2.1751861572265625,
  1.9990522861480713,
  2.0156517028808594,
  2.0147502422332764,
  2.0285372734069824,
  2.013579845428467,
  1.990257978439331,
  2.0057456493377686,
  2.072604179382324,
  3.1427173614501953),
 'test_time': (3.3181307315826416,
  3.785884380340576,
  3.2592873573303223,
  3.5078816413879395,
  3.3243298530578613,
  3.277240514755249,
  3.4366257190704346,
  3.318516969680786,
  3.199486017227173,
  2.143355131149292)}
```

In [22]:
```
from surprise import Dataset
from surprise.model_selection import cross_validate

prueba = KNNBaseline(k=100, sim_options={'name': 'pearson_baseline', 'user_based'
: True})


# Run 5-fold cross-validation and print results.
cross_validate(prueba, data, measures=['RMSE', 'MAE', 'MSE'], cv=10, verbose=True
)
```

```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE, MSE of algorithm KNNBaseline on 10 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Fold 6  Fold 7  Fold 8  
Fold 9  Fold 10 Mean    Std     
RMSE (testset)    0.9006  0.9149  0.9014  0.9136  0.9225  0.9164  0.9222  0.9212  
0.9158  0.9218  0.9150  0.0077  
MAE (testset)     0.7086  0.7149  0.7087  0.7146  0.7214  0.7168  0.7189  0.7241  
0.7180  0.7215  0.7168  0.0049  
MSE (testset)     0.8111  0.8370  0.8126  0.8346  0.8509  0.8399  0.8504  0.8486  
0.8386  0.8496  0.8373  0.0140  
Fit time          2.04    2.05    2.09    2.07    2.08    3.10    3.06    2.66    
2.17    2.18    2.35    0.40    
Test time         3.68    4.01    3.72    4.40    5.23    2.82    3.22    4.51    
4.12    3.81    3.95    0.64    
```

Out[22]:

```
{'test_rmse': array([0.90063565, 0.91487225, 0.90143737, 0.91358254, 0.92245739,
       0.91643551, 0.92215765, 0.9212075 , 0.91576619, 0.92175799]),
 'test_mae': array([0.70859484, 0.71494999, 0.70867138, 0.71464864, 0.72141067,
       0.71676147, 0.71892904, 0.72409623, 0.71799223, 0.72145466]),
 'test_mse': array([0.81114457, 0.83699124, 0.81258933, 0.83463305, 0.85092764,
       0.83985404, 0.85037473, 0.84862326, 0.83862772, 0.8496378 ]),
 'fit_time': (2.04227352142334,
  2.0513291358947754,
  2.0943620204925537,
  2.073352098464966,
  2.081934690475464,
  3.10170841217041,
  3.0565133094787598,
  2.659560441970825,
  2.1715192794799805,
  2.1832115650177),
 'test_time': (3.6753504276275635,
  4.012025594711304,
  3.724048376083374,
  4.402460098266602,
  5.231021881103516,
  2.815519332885742,
  3.2164506912231445,
  4.5070600509643555,
  4.116001844406128,
  3.811185598373413)}
```


-MSE,MAE,RMSE:

- El Error Cuadratico Medio(MSE) es la media del error que el Sistema Recomendador comete, al cuadrado, por lo que esta medida penaliza errores de prediccion mayores. Al ser una medida de error, cuanto menos sea el MSE, más eficaz es el Sistema recomendador.
- El Error Absoluto Medio (MAE) es el error que el Sistema Recomendado comete al realizar las predicciones de las valoraciones de preferencia. Por lo tanto, cuanto menor es el MAE de un Sistema Recomendador, mejor es su eficacia.
- RMSE representa básicamente la desviación estándar de las diferencias entre los valores estimados y los valores reales. Se define como la raíz cuadrada de la diferencia entre la predicción del rating y su valor real al cuadrado.

Aunque las métricas de precisión estadística han ayudado a medir con éxito diversos campos de los sistemas de recomendación, las recomendaciones más precisas a veces no son las más útiles para los usuarios. Por lo tanto, se deben explorar métricas capaces de evaluar si los objetos recomendados por el sistema son realmente relevantes para el usuario.

Los Recomendadores colaborativos se basan en comparar usuarios entre si, y comprobar su similitud entre sus preferencias(ratings). En este caso usamos el filtrado colaborativo tradicional. No tenemos ningun tipo de informacion semantica del usuario, ya que solo contamos con su rating numericos.

La evaluacion con las medidas de precision, Recall y media armonica(error medio):

- la precisión es una medida de la relevancia de los resultados, sera igual para los k elementos ya que es la media de la precision de todos los elementos(peliculas).
- Recall es una medida de cuántos resultados verdaderamente relevantes se devuelven, podemos ver que en k = 1 tiene el valor mas bajo y en k = 10 el valor mas alto
- Usamos la media armónica porque castiga los valores extremos. La idea es proporcionar una única métrica que pondera las dos proporciones (precisión y recuperación) de manera equilibrada, requiriendo que ambas tengan un valor más alto para que aumente el valor de la puntuación F1. Por ejemplo, una precisión de 0.01 y una recuperación de 1.0 daría:
- media aritmética de (0.01 + 1.0) /2=0.505,
- media armonica F1 (fórmula anterior) de 2 *(0.01* 1.0) / (0.01 + 1.0) = ~ 0.02. Esto se debe a que la puntuación F1 es mucho más sensible a que una de las dos entradas tenga un valor bajo (0,01 aquí). Lo que lo hace genial si quieres equilibrar los dos.

Una alta precisión se relaciona con una tasa baja de falsos positivos y la alta recuperación se relaciona con una baja tasa de falsos negativos. Un sistema con alta recuperación pero baja precisión arroja muchos resultados, pero la mayoría de las etiquetas predichas son incorrectas. Un sistema con alta precisión pero poca recuperación es todo lo contrario, arrojando muy pocos resultados, pero la mayoría de las etiquetas predichas son correctas. Un sistema ideal con alta precisión y alta recuperación devolverá muchos resultados, con todos los resultados etiquetados correctamente.

Para la evaluacion hacemos k (10) grupos: *En KNNBasic nos muestra unos valores de: precision = 0.945(todos los casos); recall = (1ºcaso)0.0355 - 0.1629(10º caso) ; media Armonica = 0.0684(1ºcaso) - 0.2884(10ºcaso)

*En KNNWithMeans nos muestra unos valores de: precision = 0.844(todos los casos); recall = (1ºcaso)0.0315 - 0.195(10º caso) ; media Armonica = 0.0607(1ºcaso) - 0.3176(10ºcaso)

*En KNNWithScore nos muestra unos valores de: precision = 0.8366(todos los casos); recall = (1ºcaso)0.03095 - 0.1937(10º caso) ; media Armonica = 0.05969(1ºcaso) - 0.3146(10ºcaso)

*En KNNBaseline nos muestra unos valores de: precision = 0.9183(todos los casos); recall = (1ºcaso)0.0352 - 0.1888(10º caso) ; media Armonica = 0.06791(1ºcaso) - 0.3132(10ºcaso)

Todos tienen mas o menos datos parecidos, pero podemos ver que KNNBasic es el mas preciso, y despues le sigue KNNBaseline.

En Recall tambien tienen datos parecidos, pero los que mejores resultados relevantes tienen son KNNWithMeans seguido de KNNWithScore.

En F1(media armonica) los que tienen mejores valores compensando en precision y recall en orden de mejor a peor: KNNWithMeans, KNNWithScore, KNNBaseline y por ultimo KNNBasic.

In [ ]: