

Proyecto: Expedia Hotel

Recommendations

<https://www.kaggle.com/c/expedia-hotel-recommendations>

(<https://www.kaggle.com/c/expedia-hotel-recommendations>)

Linmar Piña

06 Julio 2016

1. Objetivo

Predecir el resultado de la reservación (cluster hotel) para un evento-usuario dado, en base a su búsqueda; es decir los atributos de búsqueda asociados a ese evento en particular.

2. Datos

A continuación vemos una breve descripción de cada una de las columnas que contienen los archivos

- train.csv - Conjunto de entrenamiento / test.csv - Conjunto de prueba

Column name	Description	Data type
date_time	Timestamp	string
site_name	ID of the Expedia point of sale (i.e. Expedia.com, Expedia.co.uk, Expedia.co.jp, ...)	int
posa_continent	ID of continent associated with site_name	int
user_location_country	The ID of the country the customer is located	int
user_location_region	The ID of the region the customer is located	int
user_location_city	The ID of the city the customer is located	int
orig_destination_distance	Physical distance between a hotel and a customer at the time of search. A null means the distance could not be calculated	double
user_id	ID of user	int
is_mobile	1 when a user connected from a mobile device, 0 otherwise	tinyint
is_package	1 if the click/booking was generated as a part of a package (i.e. combined with a flight), 0 otherwise	int
channel	ID of a marketing channel	int
srch_ci	Checkin date	string
srch_co	Checkout date	string
srch_adults_cnt	The number of adults specified in the hotel room	int
srch_children_cnt	The number of (extra occupancy) children specified in the hotel room	int
srch_rm_cnt	The number of hotel rooms specified in the search	int
srch_destination_id	ID of the destination where the hotel search was performed	int
srch_destination_type_id	Type of destination	int
hotel_continent	Hotel continent	int
hotel_country	Hotel country	int
hotel_market	Hotel market	int
is_booking	1 if a booking, 0 if a click	tinyint
cnt	Numer of similar events in the context of the same user session	bigint
hotel_cluster	ID of a hotel cluster	int

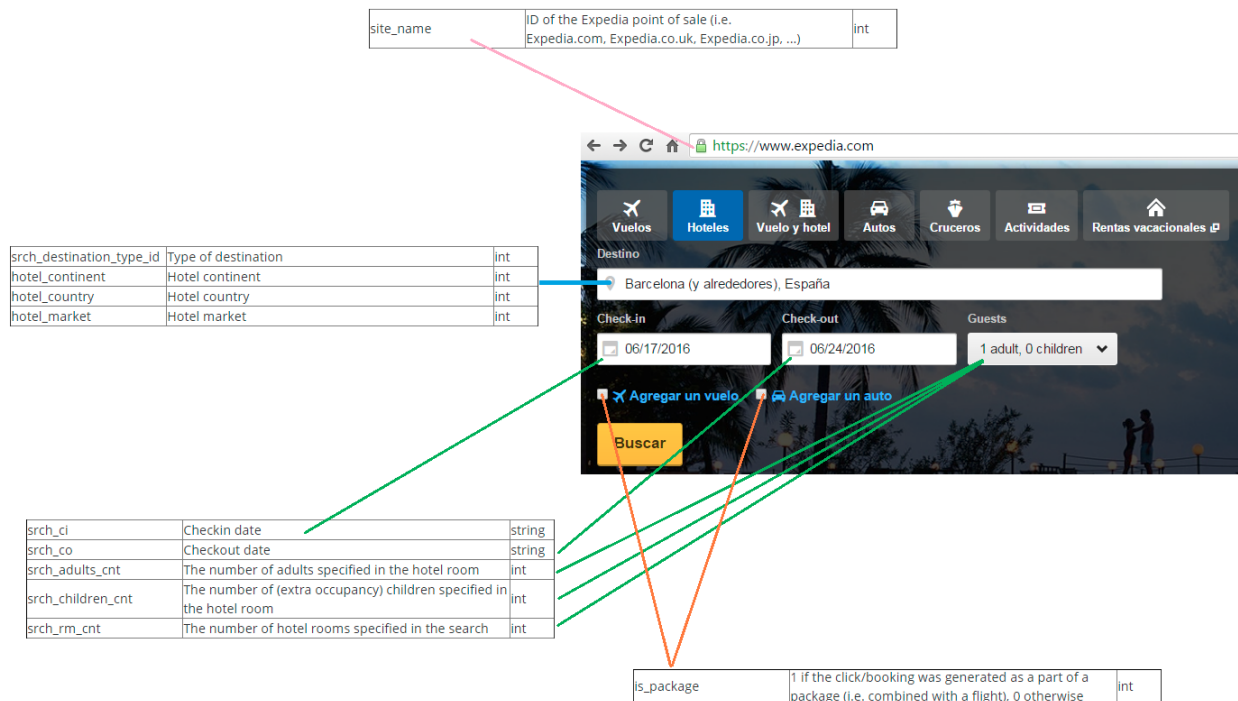
El archivo test.csv (conjunto de prueba) no contiene las ultimas tres columnas, correspondientes a is_booking, cnt, y hotel_cluster.

- destinations.csv - Atributos dominantes (tendencias) en la busqueda de hoteles

Column name	Description	Data type
srch_destination_id	ID of the destination where the hotel search was performed	int
d1-d149	latent description of search regions	double

3. Sobre Expedia

Expedia es una compañía online de viajes, dado que los datos se componen de información de los eventos en los que los usuarios reservan hoteles a través de las paginas de Expedia, es bueno ver un poco como es y como se corresponden los campos de búsqueda con la data que Expedia nos proporciona. Como podemos ver,



los atributos que son determinados por el usuario; es decir su sesión en el sitio de Expedia, son:

user_location_country	The ID of the country the customer is located	int
user_location_region	The ID of the region the customer is located	int
user_location_city	The ID of the city the customer is located	int
channel	ID of a marketing channel	int
is_mobile	1 when a user connected from a mobile device, 0 otherwise	tinyint
is_booking	1 if a booking, 0 if a click	tinyint
cnt	Numer of similar events in the context of the same user session	bigint

finalmente, de las columnas que quedan por determinar

date_time	Timestamp	string
posa_continent	ID of continent associated with site_name	int
orig_destination_distance	Physical distance between a hotel and a customer at the time of search. A null means the distance could not be calculated	double
user_id	ID of user	int
srch_destination_id	ID of the destination where the hotel search was performed	int
hotel_cluster	ID of a hotel cluster	int

vemos que son identificadores asignados por el sitio apoyado en los campos anteriores.

4. Explorando la data

Ahora que nos hemos relacionado un poco con los datos, podemos hacer un poco de exploración, comenzamos leyendo los datos:

Pandas es una librería que proporciona alto rendimiento para trabajar con datos y estructuras de datos, provee herramientas de facil uso para el analisis en el lenguaje de programación Python.

<http://pandas.pydata.org/> (<http://pandas.pydata.org/>)

```
import pandas as pd
```

Para importar la data, usamos la función de pandas, **pd.read_csv()**, y asignamos a una variable.

http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html)

```
destinations = pd.read_csv("destinations.csv")
test = pd.read_csv("test.csv")
train = pd.read_csv("train.csv")
```

Mostramos la cantidad de elementos y variables, con **DataFrame.shape** el cual retorna una lista ordenada que representa las dimensiones de la DataFrame.

```
train.shape  
test.shape
```

Exploramos las primeras 5 líneas de la data, con **DataFrame.head()**.

```
train.head(5)  
test.head(5)
```

Observaciones a simple vista:

- Los datos en *test.csv* son posteriores a *train.csv*, los primeros son del 2015 y los ultimos de los años 2013 y 2014.
- La columna **is_booking** del archivo *test.csv* contiene solo eventos de reserva, dado que contiene solos 1.

5. Exploramos **hotel_cluster**, lo que esperamos predecir.

Vamos a predecir en que **hotel_cluster** un usuario reservará despues de una busqueda determinada.

La forma en que será evaluado es, mediante 5 predicciones para cada fila (evento de usuario), el orden de aparición será desde el mas probable a menos probable, acertamos si: primero, el **hotel_cluster** correspondiente al evento esta entre las 5 predicciones que nuestro modelo hace, y segundo, que tan cerca este de la primera posición.

Ahora que sabemos lo que queremos hacer, es oportuno explorar **hotel_cluster**. Lo hacemos en principio con la funciones de pandas **DataFrame.value_count()** el cual retorna un objeto que contiene el recuento de los valores únicos en orden descendente de modo que el primer elemento es el elemento más frecuencia. Excluye los valores NA por defecto. **DataFrame.describe()** cuya salida depende de los tipo de dato solicitado, nuestro caso es numérico, e incluirá una especie de resumen estadístico.

```
#Valores unicos y cuantas veces se repiten
train["hotel_cluster"].value_counts()

#Lo guardamos en la variable m
m=train["hotel_cluster"].value_counts()

#Al aplicar .describe() sobre m, vemos que solo hay 100 valores unicos
m.describe()

#Lo cual confirmamos al ver que max=99 y min=
train["hotel_cluster"].describe()
```

Aunque la salida se trunca, muestra que el número de hoteles en cada agrupación está uniformemente distribuida y no parece haber relación entre el número de grupos y el número de elementos. El maximo valor es 99 y el minimo 0, lo que nos dice que este archivo solo contiene 100 grupos de hoteles.

6. Exploramos ahora los **user_ids** de *train.csv* y *test.csv*

Vamos a ver si los ids de usuario en el archivo *test.csv* estan contenidos en *train.csv*, lo que haremos es lo siguiente:

- Crear un subconjunto de todos los valores unicos de **user_id** de *test.csv*
- Crear un subconjunto de todos los valores unicos de **user_id** de *train.csv*
- Averiguar cuantos **user_id** de *test.csv* estan en **user_id** de *train.csv*
- Ver si el recuento coincide con el numero total de identificadores de usuario de *test.csv*

```
#.unique() devuelve los valores unicos, los cuales almacenamos en una variable
test_ids=set(test.user_id.unique())
train_ids=set(train.user_id.unique())
#Devuelve el numero de elementos comunes de test_ids y train_ids
intersection_count=len(test_ids & train_ids)
#Compara el numero de elementos comunes de test_ids y train_ids, con los elemen
tos de test_ids
intersection_count==len(test_ids)
```

Lo que confirma nuestra hipotesis que los ids de usuario en el archivo *test.csv* estan contenidos en *train.csv*.

7. Reducción de los datos de Kaggle

Los datos en *train.csv* contienen 37 millones de filas, al momento de probar diferentes métodos será

muy difícil trabajar con la data completa, queremos un conjunto pequeño de datos representativo que nos permita iterar rápidamente. Hacemos esto, primero tomando una muestra aleatoria de las filas de nuestra data, y seleccionamos nuestro nuevo conjunto de datos de prueba y entrenamiento, del conjunto de prueba original. Conservaremos la etiqueta **hotel_cluster** para cada fila, esto nos permite calcular la precisión para cada técnica.

7.1 Añadimos los campos de mes y año

Dado que el archivo *test.csv* se diferencia del archivo *train.csv* en parte por el campo de la fecha **date_time**, lo primero que haremos es añadir los campos mes y año a *train*, que nos permita segmentar nuestros datos en dos conjuntos de la misma forma. Esto lo hacemos, así:

- Convertimos la columna **date_time** en *train.csv* de ser un objeto a un valor *datetime* que es un formato manejable, con **.to_datetime()**.
- Extraemos el año con **DateFrame.dt.year** y mes con **DateFrame.dt.month** de **date_time** y las asignamos a sus propias columnas, como sigue:

```
#Convertimos un objeto en un valor Datetime
train["date_time"]=pd.to_datetime(train["date_time"])

#Creamos columnas independientes de año y mes
train["year"]=train["date_time"].dt.year
train["month"]=train["date_time"].dt.month
```

Note que aun permanece en la columna *date_time*, y a esta se le añaden dos columnas mas, año y mes.

7.2 Recolectamos 10000 usuarios

Debido a que los **user_id** en *test.csv* son un subconjunto de los **user_id** en *train.csv*, haremos un muestreo aleatorio de manera que se preserven los datos. Lo haremos seleccionando a azar un determinado número de usuarios únicos (**.unique()**) en *train*. Para esto:

```
#Este módulo implementa generadores de números pseudo-aleatorios para varias di
stribuciones.

import random

#Selecciona los usuarios no repetidos de train.csv
unique_user=train.user_id.unique()

#Añade a sel_user_ids 10000 elementos elegidos aleatoriamente de unique_user
sel_user_ids=[unique_user[i] for i in sorted(random.sample(range(len(unique_use
r)),10000))]

#Seleccionamos los 10000 con toda la informacion del resto de las columnas con
la funcion .isin()
sel_train = train[train.user_id.isin(sel_user_ids)]
```

Así tenemos los 10000 **user_id** con todas sus características (demás columnas).

7.3 Recolectamos los nuevos conjuntos de datos de entrenamiento y prueba

Ahora tenemos que crear la nueva data de entrenamiento y de prueba, las que llamaremos t1 y t2

```
#Data de entrenamiento
#Todos aquellos usuarios que sean del año 2013 y 2014, para este último desde en
ero hasta julio, de los 10000 que seleccionamos.
t1 = sel_train[((sel_train.year == 2013) | ((sel_train.year == 2014) & (sel_tra
in.month < 8)))]

#Data de prueba
#Todos aquellos usuarios que sean del año 2014, desde julio en adelante, de los 1
0000 que seleccionamos.
t2 = sel_train[((sel_train.year == 2014) & (sel_train.month >= 8))]
```

Para permanecer con las características de los datos originales, en proporción a sus años (recordamos que train está en 2013 y 2014, mientras que test 2015), entonces t1 contiene a todos los datos que tienen año 2013 y 2014 hasta julio, los datos que son del 2014 de julio en adelante están en t2

7.4 Removemos los eventos de solo click en t2 (new test)

El archivo *test.csv* original solo contiene reservas, por lo que removemos de t2 los eventos de solo clicks. Esto es cuando **is_booking** es 0 representa un click y 1 representa un booking, así:


```
#Mantenemos solamente en t2 a los is_booking=1
t2=t2[t2.is_booking==1]
```

8. Un algoritmo simple

La tecnica mas sencilla por la que podemos comenzar, es encontrar los grupos de hoteles mas comunes a través de los datos, y luego usarlos como la predicción. Para esto usamos **.value_counts()** que ordena de mayor a menor frecuencia, **.head()** devuelve los primeros 5 valores, e **index** devuelve el indice.

```
#Devuelve de forma descendente una lista con los indices de los primeros grupos
de hoteles mas frecuentes
most_common_clusters = list(train.hotel_cluster.value_counts().head().index)
```

8.1 Generando la prediccion

Convertimos **most_common_clusters** en una lista de predicción, al crear una lista con tantos elementos como filas de t2, donde cada elemento es igual a **most_common_clusters**.

```
#Shape[0] dimensión de la columna 0 de t2, lo que hago es imprimir en predition
s i veces most_common_clusters.
predictions=[most_common_clusters for i in range(t2.shape[0])]
```

8.2 Evaluando el error

Para esto usamos el método *Mean Average Precision* (explicar en la pizarra), computamos nuestra medida de error con el metodo **.mapk** de la librería **ml_metrics**, la cual cuenta con un conjunto de medidas de evaluación (Cap 10) de Machine Learning. Donde cercano a 1 es muy bueno, y cercano a 0 es muy malo.

```
import ml_metrics as metrics

#Asignamos a target cada valor de hotel_cluster de t2
target=[[i] for i in t2["hotel_cluster"]]

#Medida, target representa el valor esperado, y predictions el "modelo" y k el
numero de predicciones.
metrics.mapk(target, predictions, k=5)
```

Nuestro resultado no es bueno, pero nos da idea de como seguir.

8.3 Encontrando correlaciones

Antes de pasar a la creación de un mejor algoritmo, vamos a ver si algo se correlaciona bien con `hotel_cluster`. Esto nos dirá si debemos buscar mas profundo en alguna columna o columnas en particular.

pandas.DataFrame.corr

`DataFrame.corr(method='pearson', min_periods=1)`

Compute pairwise correlation of columns, excluding NA/null values

Parameters:

method : {'pearson', 'kendall', 'spearman'}

- pearson : standard correlation coefficient
- kendall : Kendall Tau correlation coefficient
- spearman : Spearman rank correlation

min_periods : int, optional

Minimum number of observations required per pair of columns to have a valid result. Currently only available for pearson and spearman correlation

Returns: y : DataFrame

```
train.corr()["hotel_cluster"]
```

Esto nos dice que no hay columnas que se correlacionen linealmente con **hotel_cluster**, desafortunadamente, esto significa que las técnicas como la regresión lineal y regresión logística no funcionarán bien en nuestros datos, ya que se basan en correlaciones lineales entre la predicción y el valor esperado.

9. Generando características de *destinations.csv*


Hasta ahora no hemos tocado el archivo *destinations.csv*. por lo que podemos aprovechar ahora para indagar en el. Este tiene un identificador **srch_destination_id** junto con 149 columnas de información importante sobre ese destino, veamos una muestra:

```
destinations.shape
destinations.head(5)
```

La información sumistrada no nos dice exactamente lo que cada columna es, podemos especular que son características del destino, las cuales fueron convertidas en números para conservar el anonimato.

Usaremos la información de este conjunto de datos pero vamos a necesitar comprimir el número de columnas para reducir el tiempo de ejecución. Lo haremos con PCA (Análisis de componentes principales), este métodos reducirá el número de columnas tratando de preservar la varianza por fila, busca la proyección según la cual los datos queden mejor representados en términos de mínimos cuadrados. En el codigo siguiente, nosotros:

- Inicializamos un modelo de PCA usando **scikit-learn**, <http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html> (<http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>)
- Dejaremos solo 3 columnas en nuestros datos. transformaremos las columnas desde la d1 hasta la d149 en 3 columnas.



The banner shows the scikit-learn logo, navigation links (Home, Installation, Documentation, Examples), a search bar, and a 'Fork me on GitHub' button. Below the banner, there are six categories of machine learning tasks, each with a brief description, applications, and algorithms.

Classification	Regression	Clustering
Identifying to which category an object belongs to.	Predicting a continuous-valued attribute associated with an object.	Automatic grouping of similar objects into sets.
Applications: Spam detection, Image recognition.	Applications: Drug response, Stock prices.	Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: SVM, nearest neighbors, random forest, ...	Algorithms: SVR, ridge regression, Lasso, ...	Algorithms: k-Means, spectral clustering, mean-shift, ...

Dimensionality reduction	Model selection	Preprocessing
Reducing the number of random variables to consider.	Comparing, validating and choosing parameters and models.	Feature extraction and normalization.
Applications: Visualization, Increased efficiency	Goal: Improved accuracy via parameter tuning	Application: Transforming input data such as text for use with machine learning algorithms.
Algorithms: PCA, feature selection, non-negative matrix factorization. ...	Modules: grid search, cross validation, metrics. ...	Modules: preprocessing, feature extraction. ...

```
from sklearn.decomposition import PCA

#Aplicando PCA
pca = PCA(n_components=3)
dest_small = pca.fit_transform(destinations[["d{0}".format(i + 1) for i in range(149)]])

#Convertimos en una DataFrame
dest_small = pd.DataFrame(dest_small)

#agregamos a dest_small la columna de srch_destination_id
dest_small["srch_destination_id"] = destinations["srch_destination_id"]
```

10. Generando características

Haremos lo siguiente:

- Nuevas características basadas en **date_time**, **srch_ci** y **srch_co**.
- Removeremos las columnas no numéricas como **date_time** (recordamos que ya hemos tomado la información de esta columna).
- Añadiremos **dest_small**
- Reemplazaremos valores que falten con -1.

#Definimos una función la cual llamaremos `calc_fast_features()` cuya entrada llamaremos hipotéticamente `df`

```
def calc_fast_features(df):
    #Pasamos a tipo datetime las tres columnas que tienen que ver con fecha.

    df["date_time"] = pd.to_datetime(df["date_time"])
    df["srch_ci"] = pd.to_datetime(df["srch_ci"], format='%Y-%m-%d', errors="coerce")
    df["srch_co"] = pd.to_datetime(df["srch_co"], format='%Y-%m-%d', errors="coerce")

    #Separamos en mes, día del mes, hora, minuto, día de la semana, y trimestre

    props = {}
    for prop in ["month", "day", "hour", "minute", "dayofweek", "quarter"]:
        #Devuelve el valor del atributo con nombre de objeto. Nombre debe ser una cadena. Si la cadena es el nombre de uno de los atributos del objeto, el resultado es el valor de ese atributo.
        props[prop] = getattr(df["date_time"].dt, prop)

    #carryover se le asigna p, siendo p que se pasea por las columnas de df siempre que p no sea date_time, srch_ci o srch_co. Para prop que se pasea por carryover, se le asigna a props el valor de prop

    carryover = [p for p in df.columns if p not in ["date_time", "srch_ci", "srch_co"]]
    for prop in carryover:
        props[prop] = df[prop]

    #Para date_props asignamos las etiquetas month, day... Para prop que se pasea por date_props,

    date_props = ["month", "day", "dayofweek", "quarter"]
    for prop in date_props:
        props["ci_{0}".format(prop)] = getattr(df["srch_ci"].dt, prop)
        props["co_{0}".format(prop)] = getattr(df["srch_co"].dt, prop)

    props["stay_span"] = (df["srch_co"] - df["srch_ci"]).astype('timedelta64[h]')

    #Creamos una DataFrame de props
    ret = pd.DataFrame(props)
    #Añadimos destinations
    ret = ret.join(dest_small, on="srch_destination_id", how='left', rsuffix="dest")
    ret = ret.drop("srch_destination_iddest", axis=1)
```

```

    return ret

df = calc_fast_features(t1)
df.fillna(-1, inplace=True)

```

Lo calculado anteriormente son las características tales como la duración de la estancia, detalles del registro de entrada y salida. Reemplazar los valores omitidos por -1 no es la mejor opción, pero funcionará bien por ahora, y siempre se puede optimizar el comportamiento más adelante.

11. Machine Learning

Ahora que tenemos las características de nuestros datos de entrenamiento, podemos intentar métodos de aprendizaje automático.

- Vamos a utilizar la validación cruzada de 3 folds a través del conjunto de entrenamiento. La validación cruzada divide el entrenamiento en 3 partes, y se predice el **hotel_cluster** para cada parte. El resultado final se corresponde a la media aritmética de los valores obtenidos para las diferentes divisiones.
- Vamos a generar predicciones usando el algoritmo Random Forest. Esto nos permitirá hacer predicciones, dado que ninguna de nuestras columnas están relacionados linealmente.

Primero vamos a inicializar el modelo y calcular las puntuaciones de validación cruzada:

```

predictors = [c for c in df.columns if c not in ["hotel_cluster"]]

from sklearn import cross_validation
from sklearn.ensemble import RandomForestClassifier

#n_estimators=numero de arboles
clf = RandomForestClassifier(n_estimators=10, min_weight_fraction_leaf=0.1)

#modelo
scores = cross_validation.cross_val_score(clf, df[predictors], df['hotel_cluster'], cv=3)
scores

```

Obtenemos una muy mala precision, una vez más.

12. Clasificación Binaria

La clasificación binaria se usa cuando lo que queremos clasificar puede expresarse en función de dos opciones, los métodos empleados para estos tipos problemas son: Arboles de decisión, redes neuronales, clasificación bayesiana, y support vector machines. Nosotros emplearemos, siguiendo el modelo anterior Random Forests.

Lo que haremos es lo siguiente, la información que tenemos son todas las columnas que conforman las características (aquí excluimos **hotel_cluster**), tenemos lo que queremos predecir, que es el mismo **hotel_cluster** y conocemos que **hotel_cluster** tiene valores únicos que van del 0 al 100, con las características calcularemos mediante Random Forest y cross validation la probabilidad que cada valor único de **hotel_cluster** sea el elegido por cada fila. Finalmente por fila, tomaremos las 5 probabilidades más grandes correspondientes a los grupos de hoteles del 0 al 99 y esta será nuestra predicción, luego calcularemos el error usando **mapk**.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.cross_validation import KFold
from itertools import chain

all_probs = []
unique_clusters = df["hotel_cluster"].unique() #Valores unicos de hotel_cluster
for cluster in unique_clusters:
    df["target"] = 1 #Reutilizamos la columna target la cual asignamos a todos
    los espacios 1
    df["target"][df["hotel_cluster"] != cluster] = 0 # y 0 a los espacios que
    coinciden en que hotel_cluster es diferente a cluster
    predictors = [col for col in df if col not in ['hotel_cluster', "target"]]
    probs = []
    cv = KFold(len(df["target"]), n_folds=2)
    clf = RandomForestClassifier(n_estimators=10, min_weight_fraction_leaf=0.1)
    for i, (tr, te) in enumerate(cv):
        clf.fit(df[predictors].iloc[tr], df["target"].iloc[tr])
        preds = clf.predict_proba(df[predictors].iloc[te])
        probs.append([p[1] for p in preds])
    full_probs = chain.from_iterable(probs)
    all_probs.append(list(full_probs))

prediction_frame = pd.DataFrame(all_probs).T
prediction_frame.columns = unique_clusters
def find_top_5(row):
    return list(row.nlargest(5).index)

preds = []
for index, row in prediction_frame.iterrows():
    preds.append(find_top_5(row))

metrics.mapk([[l] for l in t2.iloc["hotel_cluster"]], preds, k=5)
```

El resultado final de **mapk()** es peor que antes 0.041083333333333326.

13. Top clusters basados en hotel_cluster

13.1 Fabricacion de predicciones basadas en destinos

13.2 calculando error (0.2238813628)

14. Generando mejores predicciones para la competición

14.1 Encontrando pares de usuarios

El primer paso es encontrar los usuarios en el conjunto de entrenamiento que responden a los usuarios en el conjunto de pruebas.

14.2 Combinacion de las predicciones
(0.28400041050903119)

15. Fabricación de archivo para la competencia

Resumen