Trabajo Práctico Nro 2
Análisis de Distribuciones de Datos y Estimación de Selectividad

Base de Datos - DC - FCEN - UBA

www.dc.uba.ar/materias/bd/pagina/

1 Introducción

La habilidad para elegir un mejor método para resolver una consulta es una característica fundamental en los motores de bases de datos. La diferencia en tiempo puede convertir una operación prohibitiva en una realizable. Es por eso que los motores de bases de datos estudian desde distintos ángulos cómo optimizar cualquier consulta posible.

Uno de los muchos enfoques consiste en entender *cómo* son los datos almacenados. Esta información resulta valiosa pues permite tomar decisiones sobre qué índices usar, teniendo un impacto contundente en la performance. El siguiente caso ejemplifica el valor de lo antes dicho.

Una base de datos almacena, entre otras cosas, un padrón electoral. En esta, cada ciudadano tiene diversos campos que lo caracterizan (nombre, dni, lugar de residencia, etc). A su vez, se cuenta con 2 índices distintos: uno sobre el año del nacimiento y otro sobre el distrito donde vive el ciudadano. Se quiere resolver la siguiente query:

Listar todos los ciudadanos que hayan nacido en 1985 y vivan en el distrito nacional 19.

Para resolverla, podrían armarse los dos siguientes métodos de evaluación (entre otros):

- Usar el índice sobre año de nacimiento y por cada uno de los que coincidan con 1985, filtrar los que pertenezcan también al distrito nacional 19.
- Usar el índice sobre el distrito nacional 19 y para cada uno, filtrar los que hayan nacido en 1985.

Para decidir cuál de los dos índices usar, necesitaríamos estimar cuántos registros cumplen cada condición por separado: año de nacimiento igual a 1985 y cuántos pertenecen al distrito nacional 19. Teniendo esta información podríamos tomar la mejor decisión. Una opción sería, recorrer toda la tabla contando cuántos cumplen la primer condición y, luego, la segunda. Esto sería muy lento, yendo explícitamente en contra de la motivación de estudio. No conociendo los datos, solo podríamos hacer la siguiente estimación:

- Un ciudadano puede tener entre 16 y 110 años (a lo sumo). Por lo tanto, la probabilidad de que alguien nazca en un año particular es $\frac{1}{100-16} \approx 0.0119$
- Distritos nacionales hay 40, tomando el mismo razonamiento, la probabilidad de que alguien pertenezca a uno en particular es $\frac{1}{40} \approx 0.0250$

En consecuencia a esta estimación deberíamos elegir el uso del índice sobre el año de nacimiento, que nos dejaría solo el 1.1% de los registros e iterar sobre estos (en el otro caso, nos quedaríamos con el 2.5% y deberíamos recorrer más). El razonamiento anterior funcionaría si se cumpliese la (implícita) hipótesis que usamos: los datos están **distribuidos uniformemente** (hecho trivialmente falso tratando de distribuciones sobre edades y poblaciones).

El ejemplo anterior ilustra la necesidad de conocer los datos que manejamos y cómo esto permite tomar buenas (o malas) decisiones que impactan en la performance. Es por esto que este trabajo práctico tiene como objetivo estudiar métodos para entender las **distribuciones** subyacentes a los datos para la correcta toma de decisiones.

2 Trabajos relacionados

En función del objetivo propuesto, nos centraremos en algunas técnicas de la literatura presentados en los siguientes trabajos:

Lectura Obligatoria

- G. Piatetsky-Shapiro and C. Connell. Accurate estimation of the number of tuples satisfying a condition. *Proc. of ACM SIGMOD Conf.* pages 256-276, 1984.

Lectura Sugerida

- Poosala, Viswanath, et al. Improved histograms for selectivity estimation of range predicates. *ACM SIGMOD Record* 25.2 (1996): 294-305.
- Halim, Felix, Panagiotis Karras, and Roland HC Yap. Fast and effective histogram construction. *Proceedings of the 18th ACM conference on Information and knowledge management.* ACM, 2009.

3 Estimadores

Como se ejemplificó en la introducción, conocer los datos almacenados repercute fuertemente en la toma de decisiones y, por ende, en la performance. Es por eso que tendrán que implementar estimadores. Los estimadores son clases que permiten, a partir de la creación de una estructura, resolver diversas consultas que estimen cuántas tuplas (o valores de la columna en este caso) satisfacen una condición. Usando el ejemplo de la introducción desearíamos realizar dos consultas a dos estimadores distintos, pues predican sobre los años (que es una columna) y sobre el distrito (otra columna). La primera, sobre el año de nacimiento por igualdad con 1985 y la otra, por igualdad (para el segundo estimador) con distrito 19.

Es importante entender que la construcción del estimador puede ser costosa pero una vez que esté creado las consultas (estimaciones) serán rápidas y muchas. Es por esto que vale la pena contar con una estructura costosa con consultas veloces.

La figura 1 resume el ciclo de vida de los estimadores.

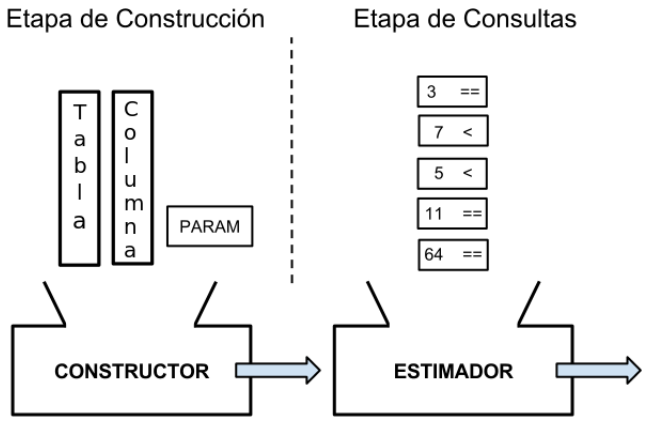

Fig. 1. Ciclo de vida de los estimadores.

1. **Etapa de Construcción**: Para poder realizar las consultas es necesario primero armar una estructura que de soporte a éstas. Dicha estructura debe almacenar información que permita tomar decisiones. La función encargada de crearla tendrá como input una tabla, una columna y un parámetro de estimación. Por ejemplo, para el caso del histograma clásico descrito en el trabajo de Piatetsky-Shapiro los inputs serán: la tabla, una columna y un parámetro llamado `param` que representa la cantidad de *bins* que tendrá el histograma resultante.
2. **Etapa de consulta**: Una vez creada la instancia del estimador, éste deberá resolver los dos tipos posibles de consultas: comparación por igual y comparación por mayor. Para resolverlas deberán basarse solo en la estructura creada anteriormente.

   El output será la *selectividad* estimada. Donde *selectividad* dada una condición representa la fracción de tuplas que satisfacen la condición (ver Piatetsky-Shapiro)

4 Ejercicios

4.1 Interacción con SQLite

El motor de base de datos que vamos a usar es SQLite[1]. Usaremos este *motor* por dos características: es accesible desde muchos lenguajes de programación, en particular Python[2], y a su vez simplifica la conexión al motor pues guarda la BD en un archivo.

A modo de práctica, deberán crear el modulo `interfazbd.py` y resolver las consultas que se detallan:

1. Crear una BD.
2. Crear una tabla en la BD anterior e insertar registros.
3. Realizar distintas consultas sobre la tabla creada y mostrar los resultados.
4. Realizar actualizaciones sobre los datos cargados.
5. Borrar algunos de los registros cargados.
6. Mostrar todas las tablas definidas en la BD.
7. Agregar índices sobre algunos de los campos de la tabla.
8. Listar todas las tablas de la BD.[3]
9. Listar todos los índices de una tabla.
10. Listar todos las columnas de una tabla.

4.2 Implementación

Deberán implementar **tres** tipos distintos de estimadores. Los estimadores son clases que deben cumplir con tres métodos: un constructor de la estructura que se usará con cada consulta que se le haga, un estimador por igualdad y un estimador por mayor. Más detalle en el apéndice (7.1).

1. Complete la clase ClassicHistogram del archivo `estimators.py`. Dicha clase implementa el método de estimación basado en histogramas clásicos (en Piatetsky et. al. 4.1).
2. Complete la clase DistributionSteps del archivo `estimators.py`. Dicha clase implementa el método de estimación basado en *Distribution Steps* (en Piatetsky et. al. 4.2).
3. Complete la clase EstimadorGrupo del archivo `estimators.py`. Dicha clase implementa el método de estimación basado en: a) *su creatividad* o b) *otro estimador* expuesto en algún *paper* o similar. Para ello puede inspirarse en los trabajos presentado en la sección 2.

[1] `http://sqlite.org/`
[2] `http://docs.python.org/2/library/sqlite3.html`
[3] Para este punto (y los siguientes), investigar el uso de la tabla `sqlite_master` y de la sentencia `PRAGMA table_info()`

Importante :

1. No pueden suponer que la columna sobre la que se construye el estimador entre en memoria. Es decir, para la construcción de la estructura deben basarse fuertemente en los servicios del motor y usar todas las herramientas que éste provee (contar tuplas que satisfagan condiciones, encontrar mínimos, máximos, etc.) y no resolver estos problemas usando Python.
2. En la inicialización de cada estimador se debe validar que la tabla y la columna sobre la que se está realizando la estimación de selectividad existan. Además, se debe controlar que el tipo de dato de la columna sea de la familia `INTEGER`.

4.3 Análisis Teórico

Podemos definir la *performance* de un estimador para una consulta dada (por igual o por mayor) sobre un dataset particular como la diferencia absoluta entre la selectividad estimada y la selectividad real. Una manera de resumir la performance de un conjunto de consultas sobre un dataset es tomando la media de las diferencias absolutas y el error relativo de la performance de cada una de las consultas.

1. Dado un dataset con *distribución uniforme*, ¿Qué estimador debería exhibir mejor performance sobre un mismo conjunto de consultas? Justificar.
2. Ídem anterior pero con un dataset con *distribución normal*.
3. Mostrar, para cada estimador, una distribución de datos que haga que el método exhiba una baja performance. Justificar.

4.4 Análisis Empírico

1. Analizar empíricamente cómo impacta la variación de los parámetros de los estimadores para datasets con distribución uniforme. Para ello:
   (a) Generar un dataset con *distribución uniforme*.
   (b) Generar N consultas por *igualdad*.
   (c) Para cada estimador y para un valor P *fijo* de parámetro (del estimador), realizar las N consultas por igualdad y calcular la performance del estimador sobre el conjunto de consultas. Repetir el procedimiento variando el parámetro P. Graficar los valores obtenidos usando un *error bar*[4] (el eje X representa el valor del parámetro mientras que el eje Y representa media de la performance para cada conjunto de consultas).
   (d) Realice lo mismos pasos del punto anterior pero tomando un conjunto de N consultas por *mayor*.
   (e) Analizar y comentar cada uno de los gráficos.
2. Repetir el punto anterior para un dataset con *distribución normal*.
3. Para el siguiente punto, deben utilizar la base de datos provista por la cátedra (`db.sqlite3`). La misma contiene una tabla con varias columnas, cada una de las cuales tiene una distribución distinta.
   (a) Graficar cada dataset (i.e., cada columna).
   (b) Mediante el *Student's T-Test Apareado* determine para qué columnas de la BD la performance entre los distintos estimadores es significativa.

[4] `http://matplotlib.org/examples/statistics/errorbar_demo.html`

4.5 Discusión

En base al análisis previo y la experimentación realizada sobre los estimadores discutir qué método es mejor para cada tipo de distribución de datos. Comentar si existe un método que se comporte mejor para el caso general (un dataset con distribución desconocida). Discutir qué otros factores hay que tener en cuenta para emplear los estimadores, por ejemplo, tiempo de construcción, costo de mantenimiento, etc.

Consideraciones

Todos los gráficos que presenten deben ser **reproducibles**. Por lo tanto, deben crear un *script* que los genere de manera automática y que permita cargar otros datos, distintos a los que usaron para generarlos.

Sugerimos usar PyLab[5] para graficar.

5 Informe

Además del código, deberán entregar un informe dónde detallen todo lo realizado. Este debe incluir, como **mínimo**, las siguientes secciones:

- **Introducción**: Introducción al tema.
- **Estimadores**: Descripción y pseudo-código de cada estimador.
- **Análisis Teórico**: Esta sección contiene lo realizado en el punto 4.3 de la sección de ejercicios.
- **Análisis Empírico**: Esta sección contiene lo realizado en el punto 4.4 de la sección de ejercicios.
- **Discusión**: Esta sección contiene lo realizado en el punto 4.5 de la sección de ejercicios.
- **Conclusiones**: Conclusiones generales sobre los estimadores y sus aplicaciones.

6 Normas de Aprobación y Condiciones de Entrega

El trabajo práctico deberá ser resuelto en grupo (el mismo que para el Trabajo Práctico Nro 1). Además, contarán con el mismo tutor asignado que en el trabajo práctico anterior para guiarlos y resolver sus consultas.

La entrega del trabajo consiste de dos partes: el **código** y el **informe**. Cada parte se aprueba por separado. No se aceptará para ninguna entrega una parte sola. Es decir, tienen que entregar el informe para que se les corrija el código y viceversa.

Fechas de Entrega

- Fecha de Entrega: **Viernes 14 de Noviembre**
- Fecha de Recuperatorio: **Viernes 12 de Diciembre**

Nota: Se recomienda fuertemente consultar con su tutor a medida que avancen con el trabajo. No obstante, podrán realizar preguntas generales a cualquier docente.

[5] `http://wiki.scipy.org/PyLab`

7 Apéndice

7.1 Módulos

`estimators.py`**:** Este módulo es el encargado de resolver las consultas de *selectividad*. Se implementan tres clases (dos de la literatura más una propuesta por ustedes). Se deben respetar tres funciones por cada clase (pudiendo agregar las que les hagan falta). Estas son:

def __init__ (self , bd_name, table_name, column_name, parameter=10)

Inicializa una nueva instancia de la clase. Toma información de dónde sacar los datos (base de datos, tabla y columna) y *parameter*, que representa el parámetro que usa cada método de estimación (cantidad de *bins*, para el *Classic Histograms*, cantidad de *step*, para el método de *Distribution Steps*, etc).

Recordar las restricciones/notas mencionadas en la sección 4.2.

def estimate_equal (self , value)

Devuelve la *selectividad estimada* para la condición por **igualdad** al parámetro *value*.

def estimate_greater (self , value)

Devuelve la *selectividad estimada* para la condición por **mayor** al parámetro *value*.

Ejemplo de uso:

```
# Creo un estimador del tipo Classic Histogram conectandome a la base de datos
# 'bd1', a la tabla 'table1', a la columna 'c1' y usando '10' como parametro.
aClassicEstimator = estimators.ClassicHistogram('db1', 'table1', 'c1', 10)

# Obtengo la selectividad para la operacion igualdad con valor a 5.
selectivity = aClassicEstimator.estimate_equal(5)
```

7.2 Librerías Recomendadas

A continuación se listan paquetes que pueden servir para la resolución del trabajo práctico.

- **numpy** (generación de distribuciones, etc.): `http://www.numpy.org/`
- **pylab** (plots, etc.): `http://matplotlib.org/`
- **scipy** (estadísticas, etc.): `http://docs.scipy.org/`
- **sqlite3** (uso de *SQLite* desde *Python*): `http://docs.python.org/2/library/sqlite3.html`