

Empowering Applications with Spatial Analysis and Mining

An Oracle White Paper
May 2005

Contents

1	INTRODUCTION	3
2	SPATIAL ANALYSIS IN ORACLE SPATIAL 10G: OVERVIEW	4
2.1	Tiled_Bins.....	4
2.2	Aggregates_For_Geometry.....	5
2.2.1	Nearest-neighbor aggregate.....	6
2.2.2	Within-distance aggregate.....	6
2.3	Aggregates_For_Layer.....	6
2.4	Spatial_Clusters.....	7
3	DATASETS.....	8
4	SPATIAL ANALYSIS: EXAMPLE APPLICATIONS	8
4.1	Location Prospecting.....	8
4.2	Clustering Analysis	13
4.3	Neighborhood-based Estimation.....	14
4.3.1	Within-distance aggregates	15
4.3.2	Nearest-neighbor aggregates	17
4.3.3	Weighted Proximity Aggregates.....	18
4.3.4	Real-estate scenario	18
5	SPATIAL DATA MINING: FRAMEWORK AND CASE STUDIES.....	19
5.1	Classification: Case Study using US Blockgroups Data ...	20
5.2	Classification with Spatial Aggregates: Case study with US Blockgroups Data.....	22
5.3	Regression Analysis with and without Spatial Estimates .	25
6	CONCLUSION.....	26

Empowering Applications with Spatial Analysis and Mining

1 INTRODUCTION

Oracle has several products that support efficient and effective analysis of warehouse data. For exploratory analysis, using OLAP/warehousing techniques users could analyze data by rolling-up/drilling-down on specific dimension hierarchies. Unlike such OLAP processing, data mining allows automatic discovery of knowledge (or information content) from a database.

In this context, Oracle Data Mining (ODM) offers several techniques to mine large datasets inside an Oracle database. The techniques include (1) classification of data using some training samples, (2) discovering hidden associations between different attributes of data, and (3) clustering techniques to identify intrinsic cluster behavior within the data. The discovered knowledge/behavior could be used effectively to serve a variety of purposes. Both the exploratory analysis tools and the data mining tools do not include geographic locations of data objects in their analyses.

Why is location important? In some applications, location may be the key to unraveling hidden patterns in the data. A classic example of this is the case of the cholera epidemic in 1855. When the cholera incidence locations were mapped, it was discovered that a water pump was near the centroid of the locations; and after the pump was closed, the epidemic subsided. Tobler's first law of geography aptly conveys the importance of location and the influence of the neighborhood:

Everything is related to everything else, but nearby things are more related than distant things.

Oracle Spatial provides new functionality in Oracle Database 10g to analyze, estimate, and predict the influence of the neighborhood based on locations of objects. Such neighborhood information can be visualized using Oracle Application Server MapViewer, or fed into other analysis tools. Alternately, the neighborhood information can augment existing data used in data mining. In this paper, we describe analysis and mining tasks that can take advantage of the analysis functionality of Oracle Spatial. These tasks include:

Location Prospecting Analysis: Using Location Prospecting Analysis, applications can prospect for geographic regions that conform to a specific, predefined attribute. Location Prospecting groups data into "tiles" and computes aggregates for specified attributes (e.g.: income, age, spending patterns) inside the tiles. You can use this functionality to identify regions or tiles that satisfy

appropriate business criteria. For instance, applications can identify geographic regions that have a high median income and use them to drill down the search for new business site selection.

Clustering Analysis: This analysis function derives common attributes from the data and groups the data into clusters based on their distance to one another. For example, an application can identify regions/clusters where crime rate is high. You can deploy additional police at the center of each cluster.

Neighborhood-based Estimation: This statistical function estimates unknown values based on known values in neighborhood. The estimation functions implement the spatial correlation referred to in Tobler's first law of geography, that is that nearby locations will have similar values to one another. Applications can utilize these functions to estimate a variety of variables such as real-estate values or crime-rate values based on values in the neighborhood.

Spatial Mining: Applications can use the neighborhood estimates to drive data mining algorithms in Oracle such as classification and association rules. Some examples include classification of new regions into high-crime or low-crime based on prior 'trained' values in the application.

In this white paper, we describe some example applications for performing each of the above-mentioned tasks. These tasks will use the newly introduced spatial analysis functionality. We give a brief overview of these functions in the next section before proceeding to the application studies.

2 SPATIAL ANALYSIS IN ORACLE SPATIAL 10G: OVERVIEW

Oracle Spatial provides new functionality for introducing the influence of neighborhood in exploratory or mining-based analysis. Below we describe some of these functions used in the case studies in this white paper. A complete list of all the functions is in the Oracle Spatial documentation. All these functions are part of the SDO_SAM package included in Oracle Spatial. This section is intended as a technical reference. If you are not interested in the technical details of the SAM functions, you may skip to the next section.

2.1 Tiled_Bins

This function has the following signature:

```
SDO_SAM.TILED_BINS(Lower_bound_in_dimension_1 NUMBER,  
                   Upper_bound_in_dimension_1 NUMBER,  
                   Lower_bound_in_dimension_2 NUMBER,  
                   Upper_bound_in_dimension_2 NUMBER,  
                   Tiling_level NUMBER,
```

SRID NUMBER DEFAULT NULL)

RETURNS Table of MDSYS.SDO_REGION

Where the SDO_REGION type has the following structure:

ID NUMBER
GEOMETRY SDO_GEOMETRY

Using the tiling level and the bounds, the function divides the region covered by the lower and upper bounds in both dimensions into as many tiles as “4 to the power of tiling_level”. These tiles (or regions) are returned using the SDO_REGION type. The SDO_REGION data type includes a numerical ID attribute for the tile id, and a GEOMETRY attribute of type SDO_GEOMETRY (see Oracle Spatial documentation for details on this type). The GEOMETRY attribute specifies the region covered by the corresponding tile. The SRID argument, if specified, indicates the spatial reference (coordinate) system for the returned geometries.

2.2 Aggregates_For_Geometry

This function computes a specified aggregate for a specified region: “ref_geometry”. It uses the information in the theme_table to compute this aggregate. This function has the following signature.

AGGREGATES_FOR_GEOMETRY (Theme_table VARCHAR2,
 Theme_geom_column VARCHAR2,
 Aggregate_type VARCHAR2,
 Aggregate_column VARCHAR2,
 Ref_Geometry SDO_GEOMETRY,
 Dist_spec VARCHAR2 default NULL)

RETURNS NUMBER

The first two arguments specify the theme_table and the theme_geom_column. The theme table has the required (demographic) information at a finer or coarser level. This information is utilized in computing the aggregate for the region corresponding to the ref_geometry. The third argument specifies the aggregate-type. This can be any numeric aggregate, such as ‘SUM’, ‘MIN’, ‘MAX’, or ‘AVG’. The fourth argument specifies the column of the theme_table on which the aggregate is to be computed. For instance, this can be the population column of the theme table. The fifth argument specifies the ref_geometry for which the aggregate is to be computed. The last argument, dist_spec, specifies additional parameters for the ref_geometry. The dist_spec function can be either ‘sdo_num_res=N’ or ‘distance= <val> unit=mile’. If it is the former, the aggregate is called nearest-neighbor aggregate. If it is the latter, the aggregate is termed as “within-distance” aggregate.

2.2.1 Nearest-neighbor aggregate

In this case, the `aggregate_for_geometry` function identifies the N rows in `theme_table` that are (i.e., where the `theme_geom_column` is) closest to “`ref_geometry`”. Let this set be $R = \{R1, \dots Rn\}$ where $R1, \dots Rn$ indicate the rowids of `theme_table`. Then the function returns the following value.

$$\text{Aggr_type } R1 \leq r \leq Rn ((\text{aggr_col}(r))$$

For instance, if the `aggregate_type` is set to ‘SUM’ and the `aggregate_column` is set to ‘CRIMEINDEX’, then the function returns the sum of the `crimeindex` values for the specified N neighbors.

2.2.2 Within-distance aggregate

Here the `dist_spec` is of the form ‘`distance = <d> unit=<units>`’ or null in which case it is treated as distance of 0. The aggregate is computed by taking the neighbors in the specified distance ‘d’. Specifically, it identifies all rows of `theme_table` where “`theme_geom_column`” is within specified distance d of “`ref_geometry`”. Let these rows be the set $R = \{R1, R2, \dots Rn\}$, where $R1, \dots Rn$ indicate rowids of the `theme_table`. Then, this function returns the following value:

$$\text{Aggregate_type } R1 \leq r \leq Rn ((\text{aggregate_column}(r) * \text{weight}(r))$$

Note that `weight(r)` indicates weighed contribution from row r based on how much geometry `ref_geometry` (expanded by distance d) and the contributing geometry (`theme_geom_column`) in `theme_table` intersect. Figure 1 shows an example. The `ref_geometry` in Figure 1 only intersects with 20% of the contributing geometry (`theme_geom_col`). So, we should proportionally weight the aggregate column for this row by 20%.

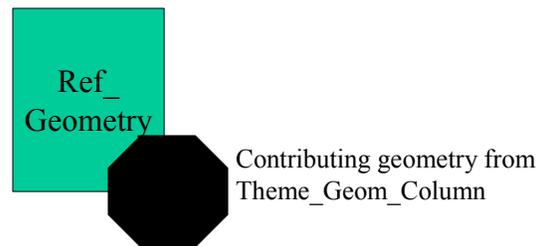


Figure 1: “`Ref_Geometry`” intersects with only 20% of contributing geometry. The associated ‘`aggregate-column`’ attribute of the contributing geometry is multiplied by 0.2 (20%).

2.3 Aggregates_For_Layer

The function `aggregates_for_geometry` computes aggregate for one “`ref_geometry`”.

This function computes the aggregates for a set of geometries in a specified “ref_table” (instead of one specific geometry). This function has the following signature:

```
AGGREGATES_FOR_LAYER (Theme_table VARCHAR2,
                      Theme_geom._column VARCHAR2,
                      Aggregate_type VARCHAR2,
                      Aggregate_column VARCHAR2,
                      Ref_Table VARCHAR2,
                      Ref_Geometry SDO_GEOMETRY,
                      Dist_spec VARCHAR2)
```

RETURNS Table of MDSYS.SDO_REGAGGR

Where SDO_REGAGGR type has the following structure:

Name	Type
-----	-----
REGION_ID	VARCHAR2(24)
GEOMETRY	MDSYS.SDO_GEOMETRY
AGGREGATE_VALUE	NUMBER

This function returns a table of SDO_REGAGGR objects where each object contains the aggregate computed using the ref_geometry in a row of the ref_table. The SDO_REGAGGR object stores the rowid in the “id” attribute, the ref_geometry in the “geometry” attribute and the computed aggregate in the “aggregate_value” attribute.

2.4 Spatial_Clusters

This function computes the clusters for a set a geometries in a specified table. Additional analysis can be performed to identify the cluster center or for visualization using Oracle Mapviewer. This function has the following signature.

```
SPATIAL_CLUSTERS (geometry_table VARCHAR2,
                  geometry_column VARCHAR2,
                  max_clusters NUMBER)
```

RETURNS Table of MDSYS.SDO_REGION

Where the SDO_REGION type has the following structure:

Name	Type
------	------

```

-----
ID          NUMBER
GEOMETRY    MDSYS.SDO_GEOMETRY

```

This function computes the clusters based on the geometry_columns of the specified geometry table. It returns each cluster as a geometry in the SDO_REGION type. The ID value is set to a number from 1 to max-clusters. The function returns a table of such SDO_REGIONs.

3 DATASETS

We use the following datasets (which you can import into Oracle Database 10g) in describing the usage of spatial analysis functionality.

Crimerate dataset: This dataset contains demographic and crime-rate information for the BlockGroups regions of the United States, referred to as USBG data. Specifically, it pertains to the 1500 blockgroups in 10-mile area around San Francisco county. You can obtain this data from the US Census or other commercial vendors.

Property Values dataset: This dataset contains fictitious data about properties and their values.

4 SPATIAL ANALYSIS: EXAMPLE APPLICATIONS

In this section, we describe example applications using spatial analysis functions. These include location prospecting, clustering and neighborhood estimation.

4.1 Location Prospecting

In this application, we will identify regions that have a high median income. We can use such regions to identify new business locations. For this purpose, we will use the tiled_bins and aggregates_for_layer functions in the sdo_sam package. Note that the USBG_DATA table has a column AVGGHICY (AVerage High InCome) that specifies the income for each blockgroup. But these values correspond to blockgroups, which are very small regions (less than 0.2 miles in diameter in most cases). The SAM functions allow you to generate regions of appropriate sizes using the tiled_bins function and estimate the incomes in these regions using the aggregates_for_layer function. Below is a sequence of operations for identifying regions/tiles of high income.

1. Generate regions of appropriate size using the tiled_bins function: We specify the lower and upper bounds to cover the extent of the data set. We specify the tiling level to 2 to generate 4*4=16 tiles for the specified extent. These tiles (with the ids and the geometry attributes) are stored in the table USBG_BINS. The first four arguments to the Tiled_bins function specify the lower and upper bounds in two dimensions. The fifth argument specifies the

tiling level as 2 and the last argument specifies the Spatial Reference ID (SRID) as 8307 (see the Oracle Spatial documentation for more details).

```
SQL> CREATE TABLE USBG_bins AS
SELECT * FROM TABLE (
    sdo_sam.tiled_bins(
        -122.58,
        -122.15,
        37.54,
        37.98,
        2,
        8307));
```

2. Compute the total income in each of these 16 tiles by using the `aggregates_for_layer` function. This function has the signature mentioned in a previous section.

The income is computed using the fine-grained blockgroups data in `USBG_DATA` table. The first two arguments specify the `USBG_DATA` table and the geometry column as the theme table and the column. This means that the information in this `USBG_DATA` table is to be used to compute aggregates. The next two arguments specify the aggregate-type, which is `SUM`, and the aggregate column, which is `AVGHHICY`. The last two arguments specify the table, and geometry for which the aggregates need to be computed. In this case, we compute the aggregates for the tiles created in the `USBG_BINS` table. The following SQL shows how to obtain the `tile_id` and the `income_per_tile` values for each tile in the `USBG_BINS` table.

```
SQL> SELECT b.id tile_id, a.aggregate_value income_per_tile
FROM TABLE(sdo_sam.aggregates_for_layer(
    'USBG_DATA', 'GEOM', 'SUM', 'AVGHHICY',
    'USBG_BINS', 'GEOMETRY')) a,
USBG_BINS b
WHERE b.rowid=a.region_id;
```

```
SQL> SELECT b.id tile_id, a.aggregate_value income_per_tile
FROM TABLE(sdo_sam.aggregates_for_layer(
    'USBG_DATA', 'GEOM', 'SUM', 'AVGHHICY',
    'USBG_BINS', 'GEOMETRY')) a,
```

```

USBG_BINS b
WHERE b.rowid=a.region_id;
TILE_ID TILED_INCOME

```

TILE_ID	TILED_INCOME
0	807658.441
1	6054774.45
2	4363591.11
3	4568570.62
4	4153349.47
5	18242767.5
6	15879459.7
7	658047.239
8	1898098.06
9	16337.1206
10	5581901.86
11	8068864.04
13	2484907.79
14	12586631.7
15	386153.109

Note that if a tile partially intersects a geometry from the theme table, then the aggregate is proportionally weighted based on the ratio of the area of the intersection to the area of the contributing geometry from the USBG_DATA table. Figure 2 shows an example. Since tile A only intersects with 20% of the area of the contributing geometry, the associated aggregate of the contributing geometry is multiplied by 0.2.

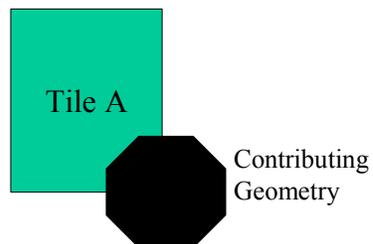


Figure 2: Tile A intersects with only 20% of contributing geometry. The associated 'aggregate-column' attribute of the contributing geometry is multiplied by 0.2 (20%).

Let us create a view, USBG_TILES, to return the above information (tile_ids, incomes) along with the tile geometries (i.e., the regions).

```
SQL> CREATE VIEW USBG_TILES AS
      SELECT b.id tile_id, a.geometry, a.aggregate_value
      income_per_tile
      FROM TABLE(sdo_sam.aggregates_for_layer(
                'USBG_DATA', 'GEOM', 'SUM', 'AVGHHICY',
                'USBG_BINS', 'GEOMETRY')) a,
      USBG_BINS b
      WHERE b.rowid=a.region_id;
```

3. Visualize the tiles and the three highest income tiles: Using Oracle Application Server MapViewer, we can color code the tiles based on their aggregate income. Figure 3 shows these tiles. The underlying blockgroups data are shown in black. All the tiles in the USBG_BINS are shown in grey (green in a colored image). The three tiles, A, B, and C, that have the highest income are shown with a black boundary. These can be obtained using the following SQL:

```
SQL> SELECT geometry
      from
      (select * from USBG_TILES order by tiled_income desc)
      where rownum<=3;
```

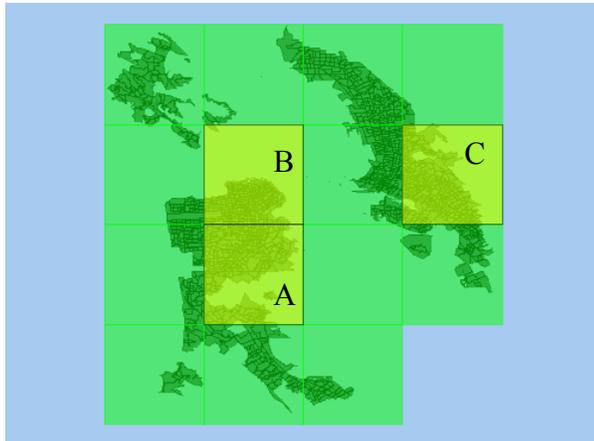


Figure 3: Identify 3 Regions/Tiles A, B, and C of high income for US blockgroups data.

4. Drill-down by sub-tile analysis: We can identify sub-regions that have high incomes in any of the tiles by repeating steps 1 to 3. For instance, we may identify sub-regions of tile A that have high income. We specify the tile boundary of A to create tiles in step 1 and repeat steps 2 and 3 with these tiles. Figure 4 shows the sub-tiles, 1, 2, and 3, that have high income within tile A.

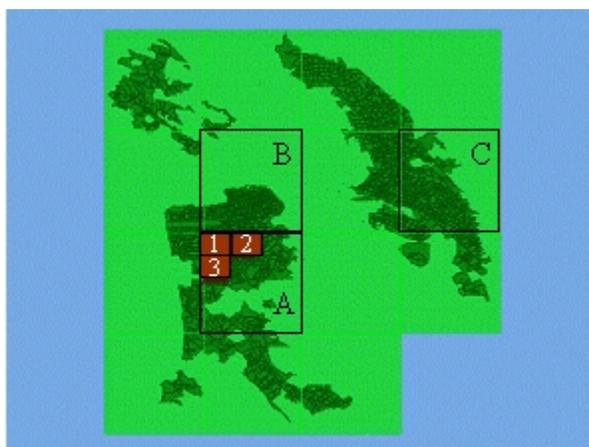


Figure 4: Identifying sub-regions/sub-tiles 1, 2, and 3 of tile A that have highest income.

The application can repeat this process any number of times to identify tiles of appropriate size.

5. Overlay with roads and other data: The application can overlay these tiles with other interesting information such as road networks to determine best sites within the identified subtiles.

Next, we will describe how to perform clustering analysis.

4.2 Clustering Analysis

An application can cluster the geometry data in a specified table using the `spatial_clusters` function in the `sdo_sam` package. This function takes the table name and the geometry column name as the first two arguments and the maximum number of clusters as the third argument. It then correlates the resulting data to create clusters. The following sequence of operations identifies clusters of high crime regions.

1. Separate out the high-crime data into a separate table, `USBG_HIGH_CRIMES`. Create a spatial index on this table.

```
SQL> CREATE TABLE USBG_HIGH_CRIMES AS
      SELECT * FROM USBG_DATA WHERE crimeindex > 150;
```

```
SQL> INSERT INTO user_sdo_geom._metadata
      SELECT 'USBG_HIGH_CRIMES', 'GEOM', diminfo, srid
      FROM user_sdo_geom._metadata
      WHERE table_name='USBG_DATA';
```

```
SQL> CREATE INDEX usbg_high_crimes_sidx ON
      USBG_HIGH_CRIMES(geom)
      indextype is mdsys.spatial_index;
```

2. Obtain four clusters from this `USBG_HIGH_CRIMES` table. The following SQL shows how to obtain geometry objects representing the clusters. Note that the geometry objects are regular Oracle objects containing a couple of numeric array (Varrays) attributes.

```
SQL > SELECT geometry
      FROM table(sdo_sam.spatial_clusters(
      'USBG_HIGH_CRIMES', 'GEOM', 4))
```

3. Visualize the cluster geometries using Oracle MapViewer. Oracle Mapviewer is a servlet to render geometry objects stored in Oracle Spatial. Figure 5 shows the clusters in transparent black. The blockgroups are shown in white (yellow in a colored image) and the high-crime regions are shown in black (dark-blue in colored image) color.

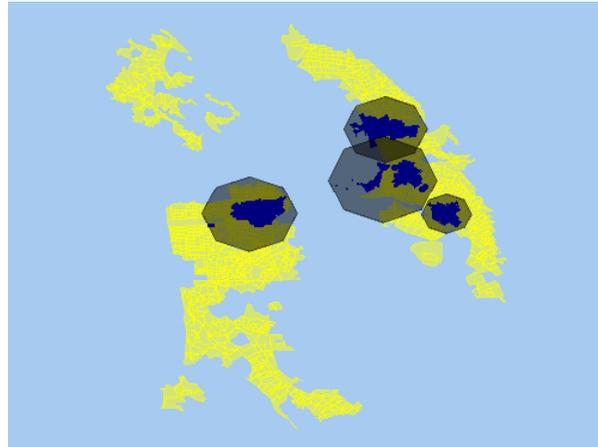


Figure 5: Clustering of high-crime blockgroups: Blockgroups are in white (or yellow), high crime blockgroups are in blue-black color and the clusters are in transparent black.

Note that clustering of geometry objects in Spatial assumes the existence of an R-tree spatial index on the corresponding geometry table. This limitation is expected to be removed and the clustering usage simplified in subsequent releases. Besides, Oracle expects to support clustering of geometries grouped by a non-spatial attribute in future releases.

As we observed in the previous examples, Spatial only supports clustering on geometry (SDO_GEOMETRY) objects. Spatial does not cluster non-spatial dimensions. For this functionality, the user is advised to use the Oracle Data Mining product.

4.3 Neighborhood-based Estimation

In this section, we will describe the functions to estimate an attribute value based on the values in the neighborhood. Specifically, we will work with the US blockgroups data. We separate the table into two tables: by random sampling 30% of the USBG_DATA, we create the USBG_SP_TEST table; the rest (i.e.,

USBG_DATA minus USBG_SP_TEST) are stored in the USBG_SP_TRAIN table.

We will estimate the crimeindex for each row of the USBG_SP_TEST table using the neighborhood crimeindex values in the USBG_SP_TRAIN table. We can perform this estimation using either within-distance aggregates or nearest-neighbor aggregates. Each of these express the neighborhood in terms of distance, e.g., 2-mile radius, or in terms of # of neighbors, for example 5 neighbors.

Both these alternatives are possible using the aggregates_for_geometry function in the sdo_sam package. This function takes the theme table name, column name, aggregate-type (sum), aggregate-column (or column-string), and the query geometry, i.e., geometry from the USBG_SP_TEST table as the first five arguments. The theme table is the table that has the information from which neighbors are to be located and their associated aggregates processed. In this case, the thematic table will be the USBG_SP_TRAIN table. The query geometries come from the USBG_SP_TEST table.

4.3.1 Within-distance aggregates

First, let us consider the estimation using neighbors within 0.25-mile distance. We specify a distance parameter as the 6th parameter to the aggregate_for_geometry functions. This computes the aggregates using the geometries of the thematic table that are within the specified distance. We refer to this as a within-distance aggregate.

Below are the within-distance aggregates for 10 rows of USBG_SP_TEST. These are computed using the crimeindex values in the USBG_SP_TRAIN table. Note that the crimeindex is the rate of crimes for a population of a thousand. So, in order to estimate the crimeindex for different neighbors, we can either take their average, or we can weight the crime-rates based on the population. Below we describe how to obtain the crimeindex with the latter scheme where the crime-index of n-neighbors within the specified distance is calculated as: $\text{sum}(\text{population} * \text{crime}) / \text{sum}(\text{population})$. The SQL below shows an example.

```
SQL> SELECT a.case_id, a.crimeindex,
           (sdo_sam.aggregates_for_geometry('USBG_SP_TRAIN',
           'GEOM', 'SUM', '(CRIMEINDEX*POPCY)',
           a.geom, 'distance=0.25 unit=mile')/
           sdo_sam.aggregates_for_geometry('USBG_SP_TRAIN',
           'GEOM', 'SUM', 'POPCY',
           a.geom, 'distance=0.25 unit=mile')) est_crimeindex
FROM   USBG_SP_TEST a
WHERE  rownum<=10;
```

CASE_ID	CRIMEINDEX	EST_CRIMEINDEX
60014008002	135.91297639609	136.87794211886
60014025002	127.90569881499	128.7467151247
60750179999	87.146558483565	(null)
60750229004	104.14295487802	107.39809849166
60750203001	138.80094285833	142.61684136892
60750110002	104.62948870238	109.94998505543
60750479007	128.3307133733	128.01680754087
60014030005	95.831228330397	95.529505779919
60750215003	108.98418468505	114.92647798412
60750227005	123.75467482927	116.15322511001

Note that the estimated crimeindex (est_crimeindex) is very close to the actual crimeindex in most cases. This is because the crimeindex is spatially correlated (i.e., values in close neighborhood tend to be close/influence one another). However, this approach might result in null values as there may not be any geometries in the USBG_SP_TRAIN in 0.25-mile radius (e.g., case_id=60750179999) of the query windows in USBG_SP_TEST.

Note also that if the quarter-mile buffer around the query only partially intersects a geometry from the thematic (USBG_SP_TRAIN) table, then the contribution of this row to the aggregate is proportionally weighted based on ratio of the area of the intersection to the area of the contributing geometry. This is illustrated in Figure 1.

We materialize the resulting estimate as the CRIME_QM attribute in the USBG_SP_TEST table. This estimate is computed using the crimeindex for the neighbors in a quarter mile. We compute the Root-Mean-Square (RMS) error for this estimation as follows. From the following SQL, we see that just performing spatial estimation may result in an error of up to 10.7% in the crimeindex. This means the accuracy of prediction on the average using just nearest-neighbor aggregates is 100-rms_error, i.e., 89.3%. Note that only 463 out of 467 rows have non-NULL values, i.e., have neighbors within a quarter mile.

```
SQL> SELECT count(*) cnt, count(crime_qm) cnt_qm,
          sqrt(avg(((crime_qm-crimeindex)*(crime_qm-crimeindex)))) rms_error
        FROM usbg_sp_test;
```

```

CNT    CNT_QM  RMS_ERROR
-----
467    463      10.6701902

```

4.3.2 Nearest-neighbor aggregates

Next, let us describe the nearest-neighbor based estimation. Specifying the string 'sdo_num_res=N' as the 6th parameter to aggregates_for_geometry functions, computes the aggregates for the nearest N neighbors for each query window. We refer to such aggregates as nearest-neighbor aggregates. Below, we show the nearest-neighbor aggregates for 10 rows of USBG_SP_TEST. These are computed using the crimeindex values in the USBG_SP_TRAIN table.

```

SQL> SELECT a.case_id, a.crimeindex,
           (sdo_sam.aggregates_for_geometry('USBG_SP_TRAIN',
           'GEOM', 'SUM', '(CRIMEINDEX*POPCY)', a.geom,
           'sdo_num_res=5')/
           sdo_sam.aggregates_for_geometry('USBG_SP_TRAIN',
           'GEOM', 'SUM', 'POPCY', a.geom, 'sdo_num_res=5'))
           est_crimeindex
FROM USBG_SP_TEST a
WHERE rownum<=10 ;
CASE_ID    CRIMEINDEX  EST_CRIMEINDEX
-----

```

```

60014008002 135.91297639609 136.8652607734
60014025002 127.90569881499 129.96087246971
60750179999 87.146558483565 133.37113237008
60750229004 104.14295487802 105.8676456603
60750203001 138.80094285833 143.74645352145
60750110002 104.62948870238 107.35064837647
60750479007 128.3307133733 128.04846613008
60014030005 95.831228330397 89.435635831754
60750215003 108.98418468505 112.02556745779
60750227005 123.75467482927 117.97754465441

```

Observe that the estimated est_crimeindex values are close to the actual crimeindex values. This is similar to the case of within-distance aggregates. However, there are

no null values in this approach using nearest-neighbor aggregates (as compared to the approach using within-distance aggregates). The case_id=60750179999 has a value of 133. Note that there is a large variation here as the nearest neighbor is more than 0.25 miles away.

We materialized the resulting estimate as the CRIME_5NBR attribute in the USBG_SP_TEST table. This estimate is computed using the crimeindex for the 5-nearest neighbors. We compute the Root-Mean-Square (RMS) error for this estimation as follows. From the following SQL, we see that just performing spatial estimation may result in an error of up to 10% in the crimeindex. This means the accuracy of prediction on the average using nearest-neighbor aggregates is 100-rms_error, i.e., 90.5%.

```
SQL>SELECT count(*) cnt, count(crime_5nbr) cnt_5nbr,
       sqrt(avg((crime_5nbr-crimeindex)*(crime_5nbr-crimeindex))) rms_error
FROM USBG_SP_TEST;

CNT  CNT_5NBR  RMS_ERROR
-----
467    467  9.49264835
```

4.3.3 Weighted Proximity Aggregates

The drawback of the nearest-neighbor aggregate is that it gives equal weight to all the neighbors irrespective of how far they are from the query. In this function, we weight the contribution of a neighbor based on its distance from the query. Note that if the distance is 0, then it will be treated as a user-specified “min_distance” value.

For instance, if there are 5 neighbors one at distances d1,d2,...d5 with aggregates a1,a2,..., a5, then the estimated aggregate would be:

$$\text{sum}(a1/d1 + a2/d2 + \dots + a5/d5) / \text{sum}(1/d1 + 1/d2 + \dots + 1/d5)$$

This function is only available with Oracle Spatial 10g Release 2.

4.3.4 Real-estate scenario

Likewise, the above aggregates can be used to estimate the values for new properties using existing data for property-values. For instance, if the existing data is in EXISTING_PROPERTIES table and the new properties are in NEW_PROPERTIES table, then we can estimate the values of the properties in NEW_PROPERTIES using the following two methods:

Within-distance Aggregate:

```
SQL> select a.property_id,
       (sdo_sam.aggregates_for_geometry(
```

```

'EXISTING_PROPERTIES',
'GEOM', 'AVG', 'VALUE',
a.geom, 'distance=2 unit=mile')) est_value
from NEW_PROPERTIES a
where rownum<=10 ;

```

Nearest-neighbor Aggregate:

```

SQL> SELECT a.property_id,
(sdo_sam.aggregates_for_geometry(
'EXISTING_PROPERTIES',
'GEOM', 'AVG', 'VALUE',
a.geom, 'sdo_num_res=5')) est_value
FROM NEW_PROPERTIES a
WHERE rownum<=10 ;

```

5 SPATIAL DATA MINING: FRAMEWORK AND CASE STUDIES

The Spatial analysis functionality described above can be used to materialize the influence of the neighborhood as additional attributes in application data. We can use this additional information to improve the effectiveness of data mining. After augmenting application data with spatial influence attributes, i.e., neighborhood estimates, the application can mine the data for interesting patterns using traditional data mining tools. Figure 6 shows this process of combining spatial analysis functions with data mining. Data mining can be performed on an augmented table data instead of the original table data to produce better results.

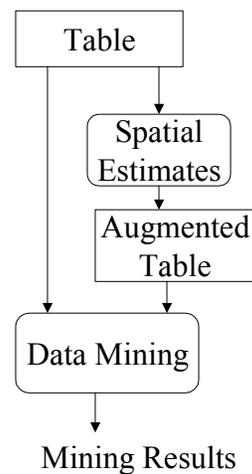


Figure 6: Data Mining on augmented data derived using Spatial Neighborhood Estimation Functions.

Oracle Data Mining (ODM) provides a variety of mining functions to automatically detect patterns in application data. These include classification, association rule mining and clustering. In this section, we will describe a case study combining ODM's classification algorithms and spatial analysis functions. First, we will describe how to perform classification using the data in the USBG_DATA table. Then, we will describe how augmenting this table with additional spatial neighborhood estimates (as described in previous sections) helps in improving the data mining results. In this comparison, we use the default Naïve Bayes algorithm for classification. The comparison results also hold for other classification algorithms such as Adaptive Bayes Network and regression analysis using Support Vector Machines (SVM).

5.1 Classification: Case Study using US Blockgroups Data

To perform classification on US Blockgroups data, we divide the USBG_DATA into two subsets: USBG_TRAIN and USBG_TEST. The USBG_DM_TEST table is formed by random sampling 30% of USBG_DATA. The USBG_TRAIN table is formed using the rest 70% of the USBG_DATA. Using the USBG_TRAIN data, we create a classification model to predict the crimeindex. Using the USBG_TEST data, we test the accuracy of the prediction. Below we list the steps to prepare the data before we construct a classification model, test it and measure its accuracy.

1. Binning (see ODM documentation for details): Bin all columns except the “case_id” and “geom” columns of the USBG_TRAIN and the USBG_TEST tables. We use 4 bins for all the attributes including the target CRIMEINDEX attribute.
2. Drop all non-binned columns except the case_id column from these tables. Now these tables have the case_id column and binned columns for other attributes such as Asian, Hispanic, White, Black populations, and Incomes.
3. Create Classification Model:

Using the USBG_TRAIN table, we create a classification model. We use the function `dbms_data_mining.create_model` for this purpose. The first argument specifies the model name. The second specifies the mining task -- whether it is classification or association rules etc. The third argument specifies the table to be used as the training dataset for the mining task. The fourth argument specifies the key for this table and the fifth specifies the mining target attribute. Here is an example using the USBG_TRAIN table.

```
SQL>  
DECLARE  
BEGIN
```

```

dbms_data_mining.create_model('USBG_CLF',
                             'CLASSIFICATION',
                             'USBG_TRAIN',
                             'CASE_ID',
                             'CRIMEINDEX_BIN');

END;

/

```

4. Apply Classification Model to Test table:

Next, we use the model created to predict values for the blockgroups in USBG_TEST table. We use the apply function in the dbms_data_mining package. This function takes the model name as the first argument, the test table, USBG_TEST, as the second argument,

the primary key, CASE_ID, attribute of this table as the third argument, and a result table, USBG_TEST_RES, as the fifth argument.

```

SQL>
DECLARE
BEGIN
  dbms_data_mining.apply('USBG_CLF',
                        'USBG_TEST',
                        'CASE_ID',
                        'USBG_TEST_RES');

END;

/

```

5. Compute Accuracy of the Model:

The accuracy of the results in USBG_TEST_RES that are predicted for the test table USBG_TEST can be measured using the function compute_confusion_matrix. This function takes the results table as the first argument, the test table as the second argument, the primary key for the test table as the third argument, the target attribute in the test table as the fourth argument. The results of the confusion matrix are stored in a table passed in as the fifth argument. This function returns the accuracy of prediction. The following SQL shows an example.

```

SQL>
DECLARE
  acc number;

```

```

BEGIN
dbms_data_mining.compute_confusion_matrix(acc,
                                         'USBG_TEST_RES',
                                         'USBG_TEST',
                                         'CASE_ID',
                                         'CRIMEINDEX_BIN',
                                         'USBG_CM_TBL');
dbms_output.put_line('Accuracy = ' || TO_CHAR(acc));
END;
/
Accuracy = 0.62 (62%)

```

This shows that the accuracy of predicting crime-rate for new blockgroups in USBG_TEST table using the model created from the USBG_TRAIN data is 62%. This means that nearly 38% of the test blockgroups were incorrectly classified by the model.

We can greatly improve the accuracy by augmenting the data with additional information on the influence of the neighborhood. The next subsection examines such a scenario.

5.2 Classification with Spatial Aggregates: Case study with US Blockgroups Data

In this subsection, we examine the effect of materializing the “neighborhood influence” as additional attributes in the application data and using this data for data mining.

For this purpose, we first divide the USBG_DATA into two subsets USBG_SP_TRAIN and USBG_SP_TEST. The USBG_SP_TEST table is formed by randomly sampling 30% of USBG_DATA. The USBG_SP_TRAIN table is formed using the rest 70% of the USBG_DATA. We augment these two tables with two attributes CRIME_5NBR and CRIME_QM. The CRIME_5NBR columns in these tables shows the estimated crime using 5 nearest-neighbors, and the CRIME_QM column shows the estimated crime using quarter-mile aggregates. The estimates are derived using spatial neighborhood aggregates with the USBG_SP_TRAIN table as the theme table.

With these two additional attributes, we perform classification: bin the columns, create a model, apply the model to predict crimeindex for the USBG_SP_TEST table and compute the accuracy.

1. Binning (see ODM documentation for details): Bin all columns except the “case_id” and “geom” columns of the USBG_SP_TRAIN and

USBG_SP_TEST tables. We use 4 bins for all the attributes including the target CRIMEINDEX attribute. This binning also bins the CRIME_5NBR and CRIME_QM attributes.

2. Drop all non-binned columns except the case_id column from these tables. Now these tables have the case_id column and binned columns for other attributes such as Asian, Hispanic, White, Black populations, and Incomes.

3. Create Classification Model:

Using the USBG_SP_TRAIN table, we create a classification model. We use the function dbms_data_mining.create_model for this purpose. The first argument specifies the model name. The second specifies the mining task -- whether it is classification or association rules etc. The third argument specifies the table to be used as the training dataset for the mining task. The fourth argument specifies the key for this table and the fifth specifies the mining target attribute. Here is an example using the USBG_SP_TRAIN table.

```
SQL> DECLARE
BEGIN
    dbms_data_mining.create_model('USBG_CLF',
                                'CLASSIFICATION',
                                'USBG_SP_TRAIN',
                                'CASE_ID',
                                'CRIMEINDEX_BIN');
END;
/
```

4. Apply Classification Model to Test table:

Next, we use the model created to predict values for the blockgroups in USBG_SP_TEST table. We use the apply function in the dbms_data_mining package. This function takes the model name as the first argument, the test table, USBG_TEST, as the second argument, the primary key, CASE_ID, attribute of this table as the third argument, and a result table, USBG_TEST_RES, as the fifth argument.

```
SQL>
DECLARE
BEGIN
    dbms_data_mining.apply('USBG_CLF',
                           'USBG_SP_TEST',
```

```
'CASE_ID',  
'USBG_TEST_RES');
```

```
END;
```

```
/
```

5. Compute Accuracy of the Model:

The accuracy of the results in USBG_TEST_RES that are predicted for the test table USBG_SP_TEST can be measured using the function compute_confusion_matrix. This function takes the results table as the first argument, the test table as the second argument, the primary key for the test table as the third argument, the target attribute in the test table as the fourth argument. The results of the confusion matrix are stored in a table passed in as the fifth argument. This function returns the accuracy of prediction. The following SQL shows an example.

```
SQL>
```

```
DECLARE
```

```
acc number;
```

```
BEGIN
```

```
dbms_data_mining.compute_confusion_matrix(acc,
```

```
'USBG_TEST_RES',
```

```
'USBG_SP_TEST',
```

```
'CASE_ID',
```

```
'CRIMEINDEX_BIN',
```

```
'USBG_CM_TBL');
```

```
dbms_output.put_line('Accuracy = ' || TO_CHAR(acc));
```

```
END;
```

```
/
```

```
Accuracy = 0.92 (62%)
```

This shows that the accuracy of predicting crime-rate for new blockgroups in USBG_SP_TEST table using the model created from the USBG_SP_TRAIN data is 92%. This means that only 8% of the test blockgroups were incorrectly classified by the model.

Compare this accuracy with the 62% accuracy obtained in the previous section. This illustrates that adding spatial neighborhood information to the original data improves the predictive power of the data mining tools. Such augmentation with

spatial aggregates is especially effective whenever the target column is spatially correlated.

5.3 Regression Analysis with and without Spatial Estimates

Oracle Data Mining provides the Support-Vector-Machine (SVM) model to support regression analysis. SVMs use machine learning theory to maximize predictive accuracy while automatically avoiding over-fit to the data. Neural networks (NN) and radial basis functions (RBFs), both popular data mining techniques, can be viewed as a special case of SVMs. SVMs perform well with real-world applications such as classifying text, recognizing hand-written characters, classifying images, as well as bioinformatics and biosequence analysis. Their introduction in the early 1990s led to an explosion of applications and deepening theoretical analysis that established SVM along with neural networks as one of the standard tools for machine learning and data mining.

One can use SVM to predict the crimeindex values for new data such as the rows in the USBG_SP_TEST table. For this, you first need to create an SVM model using the existing data in the USBG_SP_TRAIN table. The model can then be applied on the data in the USBG_SP_TEST table to predict crimeindex values.

In this section, we examine the impact of spatial estimate functions on regression using the SVM model. Table 1 shows the results of performing regression with and without spatial estimates. We use the weighted-5neighbor function as the spatial estimate function (as that was the most effective according to previous sections). The following table shows the root-mean-square error and the mean-absolute error for predicting the crimeindex for all the rows in the test table. We note that without any spatial information, the RMSE is around 25.8. By including the weighted-5neighbor estimate function, the RMSE improves to 7.24. The third row shows the effect of including the 5 neighbors and their crimeindex values explicitly with each row in the test (USBG_SP_TEST) table. The RMSE in this case is a little worse at 8.15. This implies that aggregating the crimeindex values based on their distances (using the weighed_5neighbor function) and performing SVM is an effective mechanism for capturing the neighborhood influence. Note that the RMSE using this combination of SVM and spatial estimate function is much better than that with just the spatial estimate function.

Regression Analysis	Root-Mean Square Error (RMSE)	Mean-Absolute Error (MAE)
1. Without spatial aggregates	25.8	18.8
2. Using <i>weighed_5neighbor</i> spatial crimeindex estimates	7.24	4.27
3. Using spatial neighbors, their	8.15	5.25

distances, crimeindex values (use sdo_nn functionality to materialize this)		
4. Combining 2 and 3 above	7.18	4.53

6 CONCLUSION

Spatial analysis functionality provides a convenient mechanism to incorporate neighborhood influence in application analysis. This functionality is particularly useful in identifying prospective sites for new businesses, identifying data clusters, estimating property values or crime-rates using existing data. In a specific case study, the estimates generated using spatial neighborhood estimates are around 89% accurate. This functionality can be easily integrated with Oracle Application Server MapViewer for easy visualization of the analysis results.

The analysis functions can be used as pre-processing tools to augment data for data mining tasks. Such usage of spatial analysis functions in data mining tasks may substantially improve the accuracy of the mining results. In a specific case study using spatial aggregates, the accuracy of classification improves by nearly 50% (increases from 62% to 92%) when spatial estimate data are included in mining data.

We have also performed regression analysis with and without spatial aggregate data. ODM has the Support Vector Machine model to perform regression analysis. When spatial aggregate information is used in regression analysis (in addition to existing table data), the error in regression task decreases from 25% to 7.2%.

From these studies one can conclude that spatial aggregates (computed using the sdo_sam functions) can improve the effectiveness of mining tasks such as classification and regression. In most applications, one can use the spatial aggregate functions to estimate values for the target variable and utilize such estimates in mining tasks. If the data is spatially-correlated, then this approach may improve the predictive power of mining tasks. Together with Oracle Data Mining, this spatial analysis functionality serves as a powerful engine for enabling location-based analysis and mining in business applications.



Empowering Business Applications with Spatial Analysis and Mining

May 2005

Author: Ravi Kothuri

Contributing Authors: ODM Team

Oracle Corporation

World Headquarters

500 Oracle Parkway

Redwood Shores, CA 94065

U.S.A.

Worldwide Inquiries:

Phone: +1.650.506.7000

Fax: +1.650.506.7200

oracle.com

Copyright © 2005, Oracle. All rights reserved.

This document is provided for information purposes only and the contents hereof are subject to change without notice.

This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle, JD Edwards, and PeopleSoft are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.