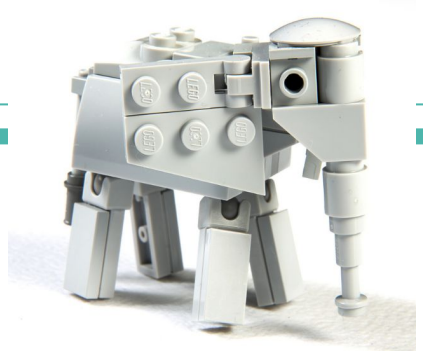
Machine Learning with PostGreSQL

Going MAD with MADLib



At the end of the talk, our  will...

- Join two tables...
  - nothing strange here :-)
- Without foreign keys
  - ... don't use REFERENCES, what's the problem!
- On datasets without natural keys!

```
work=# select * from visitors_a;
       name        |        town
-------------------+--------------------
 Orso Palerma      | Algua
 Egidio Padovano   | Sospirolo
 Mauro Pisani      | Loreto Stazione AN
 Sandra Trevisano  | Aiello Del Friuli
 Ambrosino Conti   | Monte Romano
 Iolanda Calabresi | Ridnaun
 Davide Toscani    | Villa Lagarina
 Tranquillo Loggia | Lunata
(8 rows)
```

```
work=# select * from visitors_b
        name         |         town
---------------------+----------------------
 Fabio Colombo       | Vaglio
 Padovano Egido      | Sorpirolo
 Rina Palerma        | San Benedetto Ullano
 M. Pisani           | Loreto
 Maurilio Siciliano  | Romagnano
 Ambrosino Conti     | Romano Monte
 Trevisano Sandra    | Aiello d. F.
(7 rows)
```

Who am I?

Denis Gasparin
Senior DBA and Web Developer

- Sviluppo di soluzioni software basate su PostgreSQL
  - PHP, NodeJS, Ruby, Python
- Cloud Architect
- Analista e Database Administrator
- Contributor del driver PDO PostgreSQL per PHP
- Pgrepup: tool opensource per l'aggiornamento a caldo di PostgreSQL
- rtshome.pgsql: ruolo ansible per interagire con PostGreSQL

What strategy should we adopt?

- Definiamo due insiemi:
  - i record di **`visitor_a`** sono l'insieme "A"
  - i record di **`visitor_b`** sono l'insieme "B"
- Dato l'insieme di possibili match A x B dove
  - un elemento è dato dalla coppia ($a \in A, b \in B$)
  - ogni elemento ($a \in A, b \in B$) può essere un match oppure no
- Come risultato avremo due insiemi:
  - T: insieme delle coppie che sono un match
  - F: insieme delle coppie che NON sono un match
- I due insiemi T ed F saranno ciascuno una partizione di A x B
- È un problema di classificazione binaria

`P(match) = f(a,b)`

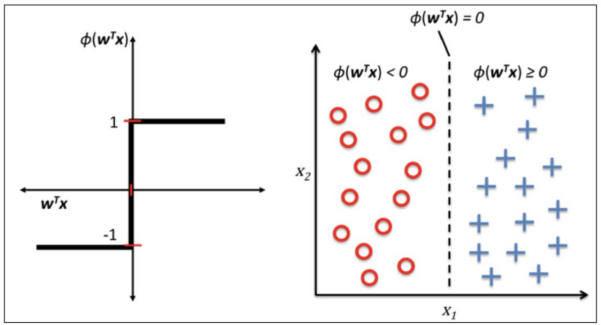
Una coppia appartiene all'insieme T se e solo se la funzione `f(a,b)` ritorna (ad esempio) un valore `> 0.7`

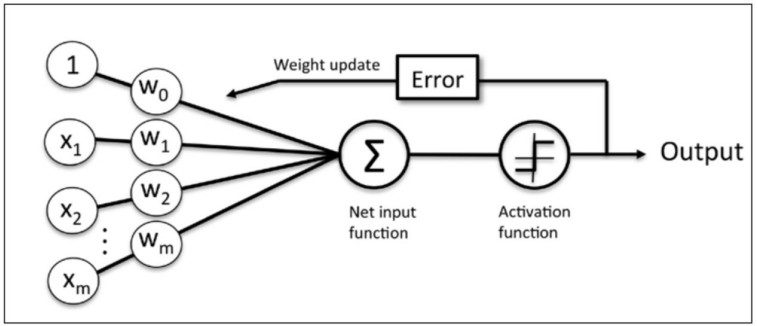
Binary Classification ML Algorithms - Perceptron

- Perceptron:
  - Due classi:
    - Positiva: +1
    - Negativa: -1
  - φ(*z*): combinazione lineare di
    - caratteristiche (x)
    - peso (w)

$$z = w_0 x_0 + w_1 x_1 + \ldots + w_m x_m = \boldsymbol{w}^T \boldsymbol{x} \text{ and } \phi(z) = \begin{cases} 1 & \text{if } z \geq \theta \\ -1 & \text{otherwise} \end{cases}$$

  - Si applica bene in casi in cui la classificazione è perfettamente lineare





Binary Classification - Logistic Regression

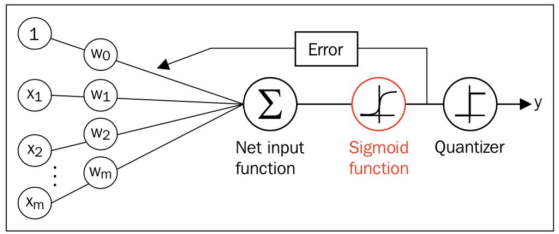
- Logistic Regression:
  - Si applica sempre in casi in casi di classificazione lineare
  - Non necessariamente le classi devono essere perfettamente separate
- Deriva dalla “Odds Ratio”:
  - probabilità che un evento avverso si verifichi rispetto all’evento positivo

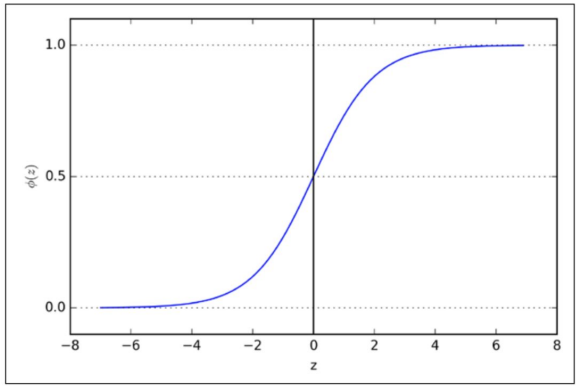
$$odds\ ratio = \frac{1 - P}{P}$$

- Sigmoid function:
  - probabilità che un campione appartenga ad una classe

$$\varphi(z) = \frac{1}{1+e^{-z}}$$

$$z = \boldsymbol{w}^T \boldsymbol{x} = w_0 + w_1 x_1 + \cdots + w_m x_m$$





How to choose our features (x_n) ?

- Devono essere variabili indipendenti
- Alcuni algoritmi di comparazione stringhe:
  - Jaro-winkler Distance: 0-1
  - Levenshtein Distance: Z
  - Longest Common Substring: Z
  - Shared n-gram features

- Dati derivanti da altre sorgenti dati:
  - Distanza tra località: M
  - Data di nascita: G

String Comparison Algorithms

- Possiamo usare pg_similarity
  - https://github.com/eulerto/pg_similarity

- Installazione molto semplice:

```
> git clone https://github.com/eulerto/pg_similarity.git
> cd pg_similarity
> USE_PGXS=1 make
> USE_PGXS=1 make install
> psql my_database -c “CREATE EXTENSION pg_similarity;”
```

Jaro-winkler: an example

- La distanza tra due parole è il numero minimo di trasposizioni di caratteri singoli richiesti per cambiare una parola in un'altra[1]
- Rapporto "favorevole" per stringhe che iniziano per la stessa sequenza

```
SELECT a.name, b.name, jarowinkler(a.name, b.name)
FROM visitors_a a, visitors_b b
ORDER BY jarowinkler(a.name, b.name) DESC;
```

➔ `JW('Ambrosino Conti','Ambrosino Conti') = 1`
➔ `JW('Sandra Trevisano', 'Trevisano Sandra') = 0.80`
➔ `JW('Mauro Pisani', 'M. Pisani') = 0.79`
➔ `JW('Egidio Padovano', 'Padovano Egido') = 0.56`
➔ `JW('Mauro Pisani', 'Maurilio Siciliano') = 0.85`

1) https://en.wikipedia.org/wiki/Jaro%E2%80%93Winkler_distance

Trigram distance: an example

- n-gram: sequenza contigua di caratteri di una frase [1]
  - 1-gram: unigram, 2-gram: bigram, 3-gram: trigram
- Un modello “n-gram” definisce la probabilità dell’n-gram successivo all’interno di una sequenza
- PostgreSQL: estensione pg_trgm
  - similarity(): 3-gram in comune tra due stringhe
  - distance: `<-> = 1 - similarity()`

➔ `<->('Ambrosino Conti','Ambrosino Conti') = 0 (JW=1)`
➔ `<->('Sandra Trevisano', 'Trevisano Sandra') = 0 (JW=0.80)`
➔ `<->('Egidio Padovano', 'Padovano Egido') = 0.27 (JW=0.56)`
➔ `<->('Mauro Pisani', 'M. Pisani') = 0.42 (JW=0.79)`

1) https://en.wikipedia.org/wiki/N-gram

Levenshtein distance: could it help?

- La distanza di Lev è l'insieme minimo di modifiche ad un singolo carattere (ins, sost, canc) necessari per trasformare una stringa in un'altra [1]
- Utile in situazione dove il numero di modifiche è basso (spell checkers)

```
SELECT a.name, b.name,
jarowinkler(a.name, b.name), a.name <-> b.name, lev(a.name, b.name)
FROM visitors_a a, visitors_b b
ORDER BY lev(a.name, b.name) DESC;
```

➔ `lev('Ambrosino Conti','Ambrosino Conti') = 1 (JW=1) (<->=0)`
➔ `lev('Sandra Trevisano', 'Trevisano Sandra') = 0.125 (JW=0.80) (<->=0)`
➔ `lev('Egidio Padovano', 'Padovano Egido') = 0.2 (JW=0.56) (<->=0.27)`
➔ `lev('Mauro Pisani', 'M. Pisani') = 0.66 (JW=0.79) (<->=0.42)`

1) https://en.wikipedia.org/wiki/Levenshtein_distance

MADLib: Big Data Machine Learning in SQL

- Progetto Apache: http://madlib.apache.org/
- Supporta PostgreSQL e Pivotal HDB
- Pacchetti disponibili per Linux e Mac
- Installazione anche mediante pgxnc
- ...oppure da sorgente:

```
> git clone https://github.com/apache/madlib
> cd madlib
> ./configure
> make && make install

# Ensure you've installed or compiled postres with python (--with-python)
> madpack -p postgres -c user@host/db install
```

MADLib: features overview

- Supervised learning:
  - Multilayer Perceptron
  - Regression models
    - linear
    - logistic
  - Support Vector Machine
  - Tree methods
- Unsupervised learning:
  - k-means clustering
- Graph:
  - All Pairs Shortest Path
  - Pagerank
- Sampling
- Tools for Linear Algebra

Using MADLib to train Pgsql to find similar names

- Creiamo tabella dove inserire i dati per allenare MADLib

```
CREATE TABLE visitors (
    a_id INTEGER,
    b_id INTEGER,
    jw_distance NUMERIC,
    trg_distance NUMERIC,
    is_true BOOLEAN
);
```

```
➔ INSERT INTO visitors(5, 6, 1, 0, true) -- Ambrosino Conti
➔ INSERT INTO visitors(2, 2, 0.56, 0.27, true) -- Egidio Padovano
➔ INSERT INTO visitors(3, 4, 0.79, 0.42, true) -- Mauro Pisani
➔ INSERT INTO visitors(4, 7, 0.80, 0, true) -- Sandra Trevisano
➔ INSERT INTO visitors(3, 5, 0.85, 0.85, false) -- Mauro Pisani/Maurilio S.
➔ INSERT INTO visitors(1, 3, 0.85, 0.55, false) -- Orso/Rina Palerma
```

Using MADLib to train Pgsql to find similar names

- Alleniamo Postgres!

```
SELECT madlib.logregr_train(
    'visitors', 'visitors_links', 'is_true',
    'ARRAY[jw_distance, trg_distance]'
);
```

- Vediamo il modello generato:

```
\x
SELECT * FROM visitors_links;
SELECT * FROM visitors_links_summary;
```

Can Elephants make predictions?

```
work=# select * from visitors_to_check_a;
 id |        name
----+--------------------
  1 | Denis Gasparin
  2 | Marco Rossi
  3 | M. Verdi
  4 | Michael Stonebraker
  5 | A. Yu
  6 | Jolly Chn
(6 rows)
```

```
work=# select * from visitors_to_check_b;
 id |      name
----+----------------
  1 | Gasparin Denis
  2 | M. Stonebraker
  3 | Jolly Chen
  4 | Yu Andrew
(4 rows)
```

The answer is YES... without mediums!

```
SELECT * FROM (

    SELECT *, jarowinkler(a.name, b.name), a.name <-> b.name AS trg_distance
    FROM visitors_to_check_a a, visitors_to_check_b b

) combinations

JOIN LATERAL (

    SELECT madlib.logregr_predict_prob(
        coef,
        ARRAY[combinations.jarowinkler, combinations.trg_distance]
    ) FROM visitors_links

) checked_visitors
ON TRUE;
```

How to update training data?

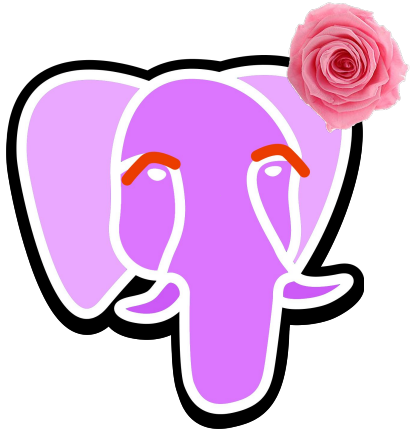
```
DROP TABLE visitors_links;
DROP TABLE visitors_links_summary;
INSERT INTO visitors VALUES (
    23,
    12,
    jarowinkler('A. Yu', 'Yu Andrew'),
    'A. Yu' <-> 'Yu Andrew', true
);
select madlib.logregr_train('visitors',
'.....');
```

Final Thoughts

- Abbiamo bisogno di molti record per fare previsioni affidabili
- In alcuni casi le variabili indipendenti non bastano
  - aggiungere ulteriori variabili
    - di tipo relazionale
    - geografico
    - etc
  - aggiungere a cascata una seconda rete neurale

Demo Time...

- Un piccolo esempio di utilizzo di PostGreSQL e Machine Learning
- Ciao! Sono Slonìka, la tua elefantessa preferita!
- Sai dirmi chi ha tenuto questo talk?
- Ambrosino Conti o Denis Gasparin?



Demo courtesy of [Esosphera Gaia AI Platform]

ESOSPHERA we tech care

LISTEN questions;

Grazie!

denis@gasparin.net

@rtshome

http://www.gasparin.net