Relazione esercizio Plagiarism

L'esercitazione 7 ha visto la definizione di un algoritmo di individuazione di plagio. Il programma controlla la presenza di plagio su 2 file di scripting python avvalendosi di diverse euristiche. Nella cartella dell'esercizio sono presenti i seguenti file:

- **Input**: Contiene i programmi python da confrontare
- **dataPreprocessing.py:** contiene le definizioni di alcune funzioni di preprocessing dei dati
- **plagiarism.py:** contiene la definizione delle euristiche di individuazione di plagio
- **main.py:** il programma principale.

Il file main.py rappresenta l'entry point del programma. Seguendo il flusso di esecuzione del main si andrà a spiegare nello specifico quali sono state le scelte implementative che hanno portato allo sviluppo dell'algoritmo.

All'atto dell'esecuzione, il programma richiede 2 parametri da riga di comando:

- **ORIGIN**: path al file base, il programma da confrontare con un eventuale plagio
- **SOUSPICIOUS**: path al file sospettato di avere plagiato ORIGIN.

I due programmi vengono analizzati e il loro codice viene recuperato sotto forma di stringhe di caratteri. Tale codice viene passato alla funzione "**select_all_relevants_words**" che si occuperà di individuare tutte le keyword presenti nel testo del codice python. In un programma i termini rilevanti da selezionare risultano molto diversi rispetto a quelli di un qualsiasi documento di testo, infatti il codice è formato da variabili, metodi, classi, funzioni, e la presenza di plagio è direttamente collegata a quanto di questi elementi è stato in parte o in tutto copiato in un altro programma. Per tali ragioni, sono stati considerati 2 criteri di individuazione di plagio:

- rappresentazione vettoriale TF-IDF dei termini rilevanti dei 2 file python
- misurazione della longest common subsequence tra i due codici

I criteri di plagio considerati per individuare le parole rilevanti in un programma python sono i seguenti:

- nomi di variabili
- nomi di metodi
- nomi di classi
- eccezioni
- invocazione di metodi di librerie
- importazione di librerie
- commenti

Tra i vari elementi abbiamo considerato anche i commenti. Una serie di evidenze ci hanno portato a fare questa scelta. Ad esempio, assumendo una brutale operazione di copia incolla, i commenti possono contribuire alla presenza di plagio in quanto occupano ed esplicano significative porzioni di un programma, specialmente quelli su singola riga, che sono spesso usati per spiegare in breve un certa istruzione. I commenti, inoltre, rappresentano gli unici elementi di un programma che possono essere manipolati come informazioni testuali. La funzione "**select_all_comment**", invocata in seguito, esegue una scansione del testo del programma, riga per riga per individuare tutti gli eventuali commenti presenti. Di questi si fa un bag of words usando le funzioni di preprocessing definite nel file **dataPreprocessing.py** e si restituisce una lista di parole.

Tornando indietro, la funzione "select_all_relevants_words**"**, analizza il testo del codice e lo parsifica con un **Abstract Syntax Tree** (libreria ast). Una volta ottenuto l'albero della sintassi del programma, lo si interroga per recuperare tutti i nodi (node.id) che rappresentano gli oggetti keyword descritti in precedenza, quindi si restituiscono tali informazioni come liste di stringhe.

I commenti i le keywords di ogni programma vengono uniti in un array. Successivamente si costruisce una rappresentazione vettoriale TF-IDF dei 2 vettori (TF_IDF_trasmorm) e viene restituito un punteggio di similarità ottenuto applicando la formula della similarità del coseno.

Una seconda evidenza di plagio è stata catturata calcolando la longest common subsequence tra i due codici. In questa fase abbiamo pensato di dare rilevanza non sono alle keywords, nella loro interezza, ma anche alla loro disposizione. La longest common subsequenze misura la sottosequenza di caratteri lunghezza massima non contigua che 2 stringhe hanno in comune. Dalla letteratura la definizione di un tale algoritmo è da preferirsi in forma iterativa con uso della programmazione dinamica, per tale ragione ogni confronto tra un carattere di una stringa e tutti i possibili caratteri dell'altra viene salvato in una matrice. Non è stato necessario ricordare l'informazione testuale, bensì solo la lunghezza della sotto sequenza più lunga. I due testi sono stati tokenizzati e i confronti avvengono tra coppie di token.

Il risultato della funzione "longest_common_subsequence" è un numero intero positivo. Per calcolare uno score abbiamo considerato la proporzione della lunghezza della sottosequenza di token in comune tra i due codici sul numero di token del codice plagiato ottenendo un valore compreso tra 0 e 1 che può essere usato come ulteriore evidenza di plagio.

I due valori di score, **similarity** per TDF-IDF e **rate_lcs** sono stati combinati. Si è deciso di restituire plagio quando entrambi assumono un valore maggiore di 0.5. Nel caso in cui uno solo fra i due è maggiore di 0.5 si riscontra un tentativo di plagio da verificare, se invece, i valori sono compresi tra 0.2 e 0.5 la presenza di plagio è debole. Per tutti altri valori è da escludersi il tentativo di plagio.

Risultati:

Per valutare il sistema abbiamo usato 4 file, presenti nella cartella input:

- codice_base.py
- codice_simile.py
- codice_non_molto_simile.py
- codice_diverso.py

I nomi dei file suggeriscono le loro affinità. Il file base è quello da cui è stato copiato qualcosa: variabili, funzioni, commenti ed altro. Codice_simile è quasi la copia esatta del file di origine, l'unica differenza sostanziale è la definizione di una funzione che in codice_base non è presente. Codice_non_molto_simile presenta affinità limitate. I nomi dei metodi sono leggermente diversi, in più vengono aggiunte altre funzioni. Infine codice diverso è completamente un altro programma. L'algoritmo è stato eseguito considerando i possibili accoppiamenti. In seguito i risultati:

codice_base.py - codice_simile.py

```
Plagiarism TDF-IDF:  72.3 %
Plagiarism Longest common subsequence:  68.10000000000001 %

We have a plagiarism attempt!
```

codice_base.py - codice_non_molto_simile.py

```
Plagiarism TDF-IDF:  46.400000000000006 %
Plagiarism Longest common subsequence:  48.6 %

The two programs seems to be not much similar
```

codice_base.py - codice_diverso.py

```
Plagiarism TDF-IDF:  0.7000000000000001 %
Plagiarism Longest common subsequence:  10.6 %

We have no plagiarism attempt!
```