UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

DIPARTIMENTO DI INFORMATICA

CORSO DI LAUREA TRIENNALE IN INFORMATICA

TESI DI LAUREA

IN

RETI DI CALCOLATORI

Progettazione e sviluppo di un classificatore per l'individuazione di commenti non informativi

RELATORI:

Prof.ssa Nicole Novielli

Prof. Pierpaolo Basile

LAUREANDO:

Giuseppe Colavito

ANNO ACCADEMICO 2019-2020

Indice

1. Introduzione

Durante scrittura e lettura del codice, i programmatori utilizzano i commenti al codice sorgente per documentarne il comportamento e per tenere traccia del razionale che guida le scelte di implementazione. I commenti sono una parte del codice che viene ignorata dal compilatore. Spiegano alcuni dettagli implementativi, aiutano gli sviluppatori a comprendere il codice e sono fondamentali per effettuarne la manutenzione.

Molti ricercatori hanno mostrato come un codice con un buon numero di commenti risulti più leggibile e più facile da manutenere [1][2]. Infatti, i commenti sono lo strumento più utilizzato dagli sviluppatori per la comprensione del codice dopo il codice stesso [3].

I commenti sono dunque una parte fondamentale del codice. Spesso la percentuale di commenti viene utilizzata come indice per l'analisi della qualità del codice [4][5]. Ma una semplice analisi quantitativa non è ragionevole, in quanto un commento, in base al suo contenuto e al contesto in cui si trova, potrebbe risultare inutile per la comprensione del codice.

Per esempio, un commento del tipo:

//loop from 1 to N ,

relativo a codice che implementa esattamente un ciclo che va da 1 a N, è un commento superfluo poiché non aggiunge nessuna informazione a quanto non si possa già dedurre dal codice sorgente (in questo caso, il ciclo *for*). Lo stesso vale per delle righe di codice commentate durante la fase di sviluppo o per note di copyright, che non hanno alcuna utilità al fine di comprendere e manutenere il codice.

Sarebbe più semplice ed efficiente la comprensione del codice se non vi fossero questi commenti di dubbia utilità.

Questo lavoro di tesi si svolge nell'ambito di una challenge, "*Second Software Documentation Generation Challenge (DocGen2)*", organizzata per la conferenza "*IEEE International Conference on Software Maintenance and Evolution (ICSME 2020)*"

L'obiettivo è costruire uno strumento automatico per classificare i commenti non informativi (*declutter*).

1.1. Struttura della tesi

Nel capitolo 2 viene presentato lo stato dell'arte sull'analisi dei commenti e sulla classificazione di testi in linguaggio naturale. Nel capitolo 3 vengono descritti il task di cui ci si è occupati e il dataset utilizzato. Nel capitolo 4 viene descritta la metodologia utilizzata per la risoluzione del task proposto, le metriche con cui si sono calcolate le performance e i classificatori utilizzati. Nel capitolo 5 sono elencate e approfondite le features individuate per la classificazione dei commenti. Nel capitolo 6 sono elencati gli strumenti utilizzati per la realizzazione del classificatore. Nel capitolo 7 vengono mostrati i risultati dello studio. Nel capitolo 8 sono riportati le conclusioni e gli sviluppi futuri dello studio effettuato.

2. Background e stato dell’arte

Le informazioni superflue all’interno della documentazione vengono definite ***Non-information.*** La prima definizione di *Non-information* è stata data con la tassonomia dei tipi di conoscenza all’interno della documentazione software a cura di *Walid Maalej* e *Martin P. Robillard* [6]:

“Una sezione della documentazione contenente qualsiasi frase completa o frammento autonomo di testo che contiene solo testo non informativo, inutile.”

I commenti al codice, se inutili, sono classificati come non informativi (***"Non-information = Yes"***) Un commento informativo invece viene etichettato come ***"Non-information = No"***.

2.1. Commenti

Molti ricercatori hanno studiato l'utilità dei commenti, mostrando che un codice commentato è più facile da leggere e da manutenere. [7]

Per esempio, Woodfield [8] ha condotto uno dei primi esperimenti dimostrando che i commenti migliorano la leggibilità del codice.

Tenny [9][10] ha confermato questo aspetto con altri esperimenti. Hartzman [11] ha mostrato che i commenti sono cruciali nel processo di manutenzione del codice.

Dati questi risultati, avere un buon numero di commenti nel codice sorgente è riconosciuto come una buona pratica (de Souza [12])

Schreck [13] ha definito diverse metriche per valutare la qualità dei commenti, ma queste metriche non tengono in considerazione la consistenza tra codice e commenti.

Alcuni ricercatori hanno proposto un metodo di valutazione del codice attraverso una metrica basata sul rapporto tra codice e commenti. (Oman and Hagemeister [14]; Garcia and Granja-Alvarez [15]).

Ogni commento è differente e la sua interpretazione dipende anche dalla posizione in cui si trova rispetto al codice sorgente. Inoltre, ogni commento è utilizzato per uno scopo diverso, come dare informazioni sull'implementazione, separare blocchi logici o inserire promemoria. Nella maggior parte dei casi il commenti servono a spiegare il codice che li

segue. Un altro uso molto diffuso è quello di usare i commenti per comunicare con altri programmatori o per aggiungere note personali (commenti di lavoro) [16]

La definizione di una tassonomia dei commenti al codice è un problema ancora aperto.

Haouari [17] e Steidl [18] hanno mostrato i primi risultati significativi nella classificazione dei commenti.

Haouari ha studiato le abitudini degli sviluppatori nella scrittura dei commenti, concentrandosi sulla posizione dei commenti rispetto al codice sorgente e proponendo una prima tassonomia che include quattro categorie di commenti

Steidl ha proposto un approccio semi automatico per la valutazione quantitativa e qualitativa della qualità dei commenti, basata sulla classificazione dei commenti in sette categorie

Maalej e Robillard [6], analizzando la documentazione in Java SDK 6 e .NET 4.0, hanno elaborato una tassonomia dei tipi di conoscenza all'interno della documentazione, tra cui "Non-information".

Tuttavia, ancora nessuno studio si è occupato di classificare i commenti come informativi o non informativi.

2.2. Text-Classification

Il task della classificazione di commenti al codice come informativi o non informativi può essere ricondotto all'ambito della *Text-categorization*.

La *Text-categorization* (o *Text-classification*) è un task di assegnazione di categorie predefinite a documenti testuali.

Questo tipo di task ha importanti applicazioni nel mondo reale.

Ad esempio, le notizie sono in genere organizzate per categorie di argomenti. I documenti accademici sono spesso classificati per domini tecnici e sottodomini.

Un'altra importante applicazione della Text-categorization è il filtraggio dello spam, in cui le mail sono classificate in due categorie: Spam o Non-Spam.

È utile distinguere i problemi di Text-classification in base al numero di classi a cui un documento può appartenere. Se ci sono esattamente due classi (es.: spam / non-spam), si

tratta di un problema di classificazione binaria. Se ci sono più di due classi (es.: positivo / neutrale / negativo) e ogni documento può appartenere solamente ad una classe, si tratta di un problema multi-classe. Tuttavia, in alcuni casi, un documento potrebbe essere associato a più categorie. Questo tipo di task è chiamato "multi-label". I problemi multi-classe e multi-label sono spesso trattati riducendoli a k problemi di classificazione binaria, uno per categoria. Per ogni problema di classificazione binaria, i membri della rispettiva categoria sono individuati come esempi positivi, mentre tutti gli altri sono esempi negativi. [27]

Un documento di testo è tipicamente rappresentato come un vettore di features x:

$$d = (t^{(1)}, \dots, t^{(p)})$$

Dove $t^{(i)}$ solitamente codifica la presenza di una parola, di un n-gramma, di frasi, etc. all'interno del documento.

Un metodo standard per rappresentare i documenti è l'approccio *bag-of-words.* Un documento d è rappresentato dal vettore delle parole che contiene. Una versione specifica di questo approccio è lo schema di pesatura dei termini *TF-IDF*:

$$t^{(i)} = TF(i,d) \cdot IDF(i)$$

Dove $TF(i,d)$ è il numero di occorrenze del termine i nel documento d.

$IDF(i) = log\frac{N}{n_i}$ è la *Inverse Document Frequency*, dove N è il numero totale di documenti in una collezione e n_i è il numero di documenti in cui compare il termine i [27].

Un articolo importante nell'ambito della Text-classification, a cui si fa riferimento in questa sezione, è quello di Fabrizio Sebastiani [22].

Uno dei primi approcci utilizzati nell'ambito della Text-classification consiste nella costruzione manuale di sistemi esperti in grado di prendere decisioni. Questi sistemi esperti consistono in un insieme di regole logiche del tipo:

$$\boldsymbol{if}\ (DNF\ formula)\ \boldsymbol{then}\ (category)$$

Una *DNF* ("disjunctive normal form") è una disgiunzione di clausole in *and*. Il documento è classificato di una certa categoria se soddisfa una certa formula [28].

Queste regole devono essere scritte manualmente, perciò all'aggiunta di nuove categorie sarà necessario l'esperto del dominio che dovrà scrivere nuove regole.

Più avanti sono diventati più popolari approcci di machine-learning [29] al problema della Text-classification. In questi approcci, un processo induttivo produce automaticamente un classificatore per una categoria c osservando le caratteristiche di un insieme di documenti classificati da un esperto del dominio come appartenenti alla stessa categoria c. Questo genere di approccio al problema è chiamato *apprendimento supervisionato*.

Un insieme di esempi di documenti già classificati per un task di classificazione binaria è chiamato training set ed è del seguente tipo:

$$S = \{(d_1, c_1), \dots, (d_n, c_n)\}$$

dove n è il numero di esempi di training e $c_i \in \{-1, +1\}$ indica la categoria del rispettivo documento. Usando questo insieme di esempi di training, un algoritmo di apprendimento supervisionato ha lo scopo di trovare una regola di classificazione ottimale, cioè una funzione che avendo in input le features restituisca una categoria. Ottimale significa che la regola di classificazione deve essere in grado di classificare accuratamente i documenti di training e di generalizzare per classificare bene anche i documenti mai visti prima [27].

I classificatori probabilistici [30] decidono la categoria del documento calcolando $P(c_i|\vec{d}_j)$, la probabilità che un documento rappresentato dal vettore delle features $\vec{d}_j$ appartenga alla categoria c_i. $P(c_i|\vec{d}_j)$ viene calcolata attraverso il teorema di Bayes:

$$P(c_i|\vec{d}_j) = \frac{P(c_i)P(\vec{d}_j|c_i)}{P(\vec{d}_j)}$$

$P(\vec{d}_j)$ è la probabilità che un documento scelto casualmente sia rappresentato del vettore $\vec{d}_j$, $P(c_i)$ è la probabilità che un documento scelto casualmente appartenga alla categoria c_i. La stima di $P(\vec{d}_j|c_i)$ risulta problematica in quando il numero di possibili vettori $\vec{d}_j$ è molto grande. Una scelta comune per aggirare questo problema è l'assunzione di indipendenza di ogni coppia di features nel vettore $\vec{d}_j$. I classificatori che utilizzano l'assunzione di indipendenza sono chiamati *Naïve Bayes.*

Un altro approccio possibile è l'utilizzo di *alberi di decisione* [29]. Un albero di decisione per la classificazione del testo è un albero in cui i nodi interni sono etichettati con delle

features del testo, gli archi sono delle condizioni sul peso della feature presente nel documento. Le foglie dell'albero contengono le categorie. Questo tipo di classificatori testano i pesi dei termini del documento fino a raggiungere una foglia. L'etichetta della foglia sarà l'output del classificatore (una categoria).

Nonostante alberi di decisioni e regole logiche siano stati esplorati per la Text-classification, i classificatori lineari sono tra i più utilizzati. Un classificatore lineare per la categoria c_i è un vettore $\vec{c}_i = (w_{1i}, \dots, w_{|T|i})$ appartenente allo stesso spazio $|T|$ dimensionale in cui i documenti sono rappresentati, tale che la categoria per il documento d_j è data dal prodotto scalare $\sum_{k=1}^{|T|} w_{ki} \cdot w_{kj}$ di $\vec{d}_j$ e $\vec{c}_i$. I metodi per costruire classificatori lineari sono solitamente divisi in due classi:

- Metodi *batch*
- Metodi *on-line*

I metodi *batch* costruiscono un classificatore analizzando tutto il training set in una volta. Un esempio di metodo *batch* è il metodo di *Rocchio* [33]. Il metodo di *Rocchio* consiste in un profilo esplicito di una categoria. Un profilo c_i è una lista pesata dei termini la cui presenza o assenza è più utile per riconoscere c_i. Il metodo di Rocchio costruisce un classificatore $\vec{c}_i = (w_{1i}, \dots, w_{|T|i})$ per la categoria c_i attraverso la formula:

$$w_{ki} = \beta \cdot \sum_{\{d_j \in POS_i\}} \frac{w_{kj}}{|POS_i|} - \gamma \cdot \sum_{\{d_j \in NEG_i\}} \frac{w_{kj}}{|NEG_i|}$$

Dove w_{kj} è il peso del termine k all'interno del documento d_j, POS_i è l'insieme degli esempi positivi per la categoria c_i, NEG_i è l'insieme degli esempi negativi per la categoria c_i. β e γ sono parametri di controllo che permettono di impostare la importanza relativa degli esempi positivi e negativi. Per esempio, se β è impostato a 1 e γ a 0 il profilo della categoria c_i è esattamente il centroide degli esempi positivi. I classificatori che utilizzano il metodo di Rocchio premiano la vicinanza di un documento al centroide degli esempi positivi e la distanza dal centroide degli esempi negativi.

I metodi *on-line* (o incrementali) costruiscono un classificatore subito dopo aver esaminato il primo documento, e lo perfezionano gradualmente mentre ne esaminano altri. Un esempio di metodo *on-line* è il *percettrone* [31][32]. In questo algoritmo il classificatore c_i è inizializzato impostando tutti i pesi w_{ki} allo stesso valore positivo. Quando un documento d_j viene esaminato, il classificatore restituisce una categoria. Se

il risultato del classificatore è corretto, il classificatore non viene modificato. Se, invece, il risultato del classificatore è errato, allora i pesi del classificatore vengono modificati: se d_j era un esempio positivo di categoria c_i, i pesi w_{ki} dei termini attivi, cioè i termini presenti all'interno del documento ($w_{kj} = 1$) vengono "promossi", aumentandoli di una quantità prefissata α chiamata learning rate. In caso contrario i pesi vengono "degradati" diminuendoli della stessa quantità α.

Un classificatore di testo a *rete neurale* è una rete di unità, in cui le unità di input rappresentano le features del testo, le unità di output rappresentano le categorie e i pesi degli archi che connettono le unità rappresentano relazioni di dipendenza. Per classificare un documento d_j, i pesi w_{kj} dei suoi termini vengono passati alle unità di input. L'attivazione di queste unità è propagata i avanti nella rete e il valore delle unità di output determina la categoria. Un modo tipico di allenare le *reti neurali* è la *backpropagation,* i cui i pesi dei termini sono passati alle unità di input e se la classificazione fornita dalla rete è errata, l'errore viene propagato all'indietro per cambiare i parametri della rete e minimizzare l'errore [34][35]. Un semplice caso di rete neurale è il *percettrone*. Una *rete neurale non lineare* [36] è una rete con uno o più strati aggiuntivi di unità.

I classificatori basati su esempi non costruiscono una rappresentazione esplicita di una categoria c_i, ma si affidano alle etichette dei documenti di training simili ai documenti in in input. Sono anche soprannominati *lazy learners* perché "rimandano la decisione su come generalizzare le informazioni acquisite di dati di training fino a quando non arriva un documento da classificare in input [29]. Un esempio di classificatore basato su esempi è il *k-NN* (*k nearest neighbors*) [37]. Per decidere se un documento d_j appartiene alla categoria c_i, *k-NN* trova i k documenti di training più simili a d_j. Se un numero abbastanza alto di questi appartiene alla categoria c_i, allora d_j appartiene a c_i, altrimenti no. È necessario avere quindi una misura di similarità (es.: coseno) tra documenti e un numero k di documenti più simili da consultare per prendere decisioni.

La *support vector machine (SVM)* [38][39] è un classificatore che cerca tra tutte le superfici $\sigma_1, \sigma_2, \ldots$ nello spazio $|T|$ dimensionale, una superficie che separi gli esempi positivi da quelli negativi con il più largo margine possibile. Nel caso bidimensionale, varie linee possono essere scelte come superfici di decisione. La *SVM* sceglie l'elemento centrale dell'insieme più ampio di linee parallele, cioè dall'insieme in cui la distanza massima tra due elementi dell'insieme è più alta. La migliore superficie di decisione è

determinata solo da un piccolo sottoinsieme del training set. Questi esempi si chiamano *support vectors*.

I *committees classifiers* (o *ensambles)* sono basati sull'idea che, dato un task che richiede la conoscenza di un esperto per funzionare, k esperti potrebbero essere meglio di uno solo nello svolgere il task. Si applicano k classificatori differenti $\phi_1, \dots, \phi_k$ allo stesso task, cioè stabilire se d_j appartiene alla categoria c_i, e successivamente si combinano appropriatamente i risultati. Sono necessari quindi k classificatori e una funzione di combinazione dei risultati. I classificatori dovrebbero essere più indipendenti possibile [40]. La funzione di combinazione più semplice è il *majority voting*, in cui gli output binari dei k classificatori vengono aggregati e la decisione che raggiunge la maggioranza $\left(\frac{k+1}{2}\right)$ viene presa [41].

3. Task e Dataset

Il task di cui ci si è occupati è stato definito nel contesto della "DeClutter Challenge"**.** L'obiettivo della "DeClutter Challenge" è costruire uno strumento automatico in grado di identificare documentazione software superflua al livello di classe o di file. (*decluttering* della documentazione[1]). A partire da un dataset di commenti già etichettati, l'obiettivo è predire le etichette per commenti mai visti in precedenza.

Attualmente non esiste alcuno strumento in grado di svolgere questo task automaticamente.

3.1. Repository

Ai partecipanti alla *DeClutter Challenge* è stato fornito un set di commenti già etichettati. Il repository da cui sono estratti i commenti è *Jabref* [19], un software di gestione di riferimenti bibliografici. Il codice a cui si riferiscono è scritto in linguaggio *Java*.

3.2. Commenti

Java offre tre modi alternativi per commentare il codice sorgente:

- *Line* comment, viene utilizzato per commentare una singola linea di codice.

  Il *line* comment inizia con "//" e termina alla fine della riga.

  Esempio:

  ```
  boolean maskText = (textInputControl instanceof PasswordField)
  ; // (maskText("A") != "A");
  ```

- *Block* comment, viene utilizzato per commentare in più linee di codice.

[1] https://dysdoc.github.io/docgen2/index.html

Il *block* comment inizia con " /* " e termina con " */ ".

Esempio:

```
MOVE_TAB_ARROW(MaterialDesignIcon.ARROW_UP_BOLD),
/*css:
arrow-up-bold */
```

- *Javadoc* comment, è un caso particolare di *block* comment che viene utilizzato per la generazione automatica di documentazione grazie al tool *javadoc*.

  Il *javadoc* comment inizia con " /** " e termina con " */ ".

- Esempio:

```
/**
 * This class contains some code taken from {@link com.sun.javaf
x.scene.control.behavior.TextInputControlBehavior},
 * witch is not accessible and thus we have no other choice.
 * TODO: remove this ugly workaround as soon as control behavior
is made public
 * reported at https://github.com/javafxports/openjdk-jfx/issues
/583
 */
```

3.3. Training Set

Il "*Declutter Development Set*"[2] è composto di 1047 commenti.

Per ogni commento è disponibile:

- il link al codice sorgente che lo contiene
- il tipo di commento (Line, Block, Javadoc)

[2] https://github.com/dysdoc/declutter/blob/master/declutter-gold_DevelopmentSet.csv

- la linea in cui inizia il commento
- il testo del commento
- l'etichetta "**Non-information**" (Yes/No)

Gli esempi sono sbilanciati: sono presenti $743(71\%)$ esempi di commenti non informativi e $304(29\%)$ esempi di commenti informativi.

	Non-information = Yes	**Non-information = No**	**Totale**
Javadoc	66	305	371
Line	219	424	643
Block	19	14	33
	304 (29%)	743 (71%)	**1047**

Non è attualmente disponibile il test set etichettato, essendo la competizione ancora in corso. È stato quindi utilizzato esclusivamente il development set per la valutazione delle performance, attraverso la tecnica *K-Fold Cross Validation*, di cui si parlerà nel capitolo 4.

È utile riportare la definizione di alcune metriche utilizzate nella discussione, che verranno poi spiegate dettagliatamente nel capitolo 4.

In particolare, parliamo di un task di classificazione binaria.

L'*accuracy* misura quanti esempi sono stati classificati correttamente rispetto al numero totale di esempi.

Le metriche *precision*, *recall* e *F1*, derivano dal campo dell'Information Retrieval [20], e possono essere adottate a un task di classificazione. Esse forniscono dei giudizi sul risultato di una classificazione su un insieme di test con etichette conosciute a priori. Questi giudizi sono espressi come valori numerici compresi tra 0 e 1, dove 1 è il miglior risultato ottenibile mentre 0 il peggiore. La precision riguarda il numero di istanze classificate correttamente rispetto quelle sbagliate, il recall il numero di istanze classificate correttamente rispetto al massimo possibile, mentre la F1 è la media armonica di precision e recall, il che vuol dire che ha un risultato alto solo se entrambi i valori sono alti.

Un’altra metrica considerata in questo studio è il *Coefficiente di correlazione di Matthews*[21]. Questa metrica è molto valida nello scenario di classificazione binaria con i numeri di esempi per classe sbilanciati. Essendo un coefficiente di correlazione ha valori tra -1 e 1.

4. Metodologia

Per il task di "decluttering" della documentazione, si è scelto di non utilizzare un semplice approccio di Text-classification. Questo perché l'utilità di un commento al codice non dipende solamente dal testo che contiene. Analizziamo l'esempio di commento già citato nelle sezioni precedenti:

//loop from 1 to N

Questo commento risulta inutile perché spiega una riga di codice auto-esplicativa, che non ha bisogno di ulteriori spiegazioni:

for (int i=1; i<N; i++)

Si è cercato quindi di utilizzare una serie di informazioni diverse dalle semplici features testuali. L'obiettivo è quello di trovare delle caratteristiche dei commenti e del codice a cui si riferiscono che possano essere utili nella classificazione dei commenti come informativi o non informativi.

Fissato l'obiettivo, si è scelto di utilizzare un classificatore con le sole features testuali come baseline da superare.

La prima caratteristica analizzata è stata la lunghezza del commento. Si è ipotizzata una correlazione tra l'utilità del commento e il numero di caratteri/parole utilizzati. Un commento più lungo potrebbe essere più utile e contenere maggiore informazione rispetto ad uno più breve. Oppure, al contrario, un commento più breve e conciso potrebbe essere più utile e più efficiente in termini di tempo speso per la lettura e la conseguente comprensione del codice a cui è riferito.

Un altro aspetto analizzato è stata la ripetitività del commento rispetto al codice. Questo aspetto è suggerito direttamente nella descrizione del task della "Declutter Challenge" (capitolo 2). Se un commento è ripetitivo, utilizza le stesse parole del codice o comunque non aggiunge nulla a quello che è facilmente comprensibile dal codice, non è informativo e potrebbe essere causa di perdita di tempo per chi legge il codice cercando di comprenderlo. Si è cercato quindi di misurare la sovrapposizione tra commenti e codice a cui si riferisce. Tuttavia, non sempre è banale capire a quale riga di codice si riferisce il commento senza leggere prima il codice e il contenuto. Uno degli aspetti trattati da questo studio è l'individuazione della riga a cui si riferisce il commento.

Un'ipotesi che è sorta è quella secondo cui l'utilità del commento possa dipendere dal costrutto del codice a cui si riferisce: un costrutto semplice (es.: espressione) potrebbe essere più facile da comprendere autonomamente (e quindi inutile da commentare) rispetto ad un costrutto più ampio e complesso (es.: dichiarazione di una classe o di un metodo).

Successivamente, essendo la repository da cui sono stati estratti i commenti scritta in linguaggio *Java*, è stata effettuata un'analisi dei *tags* presenti nei commenti. I *tags* sono costrutti utili nella creazione automatica di documentazione a partire dai commenti. Si è cercato di capire se vi fosse qualche tag discriminante, che rendesse il commento utile o inutile.

L'ultimo aspetto che è stato tenuto in considerazione per la classificazione dei commenti come informativi o non informativi, è il tipo di commento utilizzato. (sezione 3.2).

4.1. Metriche utilizzate

Le metriche di valutazione *Precision, Recall, f-measure* e *Accuracy*, utilizzate in questo studio, sono riprese da "Machine Learning in Automated Text Categorization" di Fabrizio Sebastiani [22]. Brevemente ora si fornisce una spiegazione su cosa queste misure rappresentano, mostrandone anche le formule per calcolarle.

L'efficacia di una classificazione è misurata solitamente in termini delle classiche notazioni, utilizzate in information retrieval di:

- *precision* (π) (precisione)

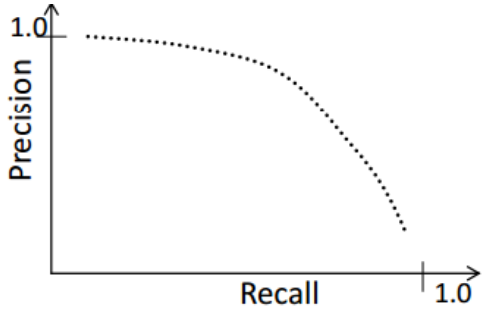
$$\pi = \frac{TP}{TP + FP}$$

- *recall*(ρ) (richiamo)

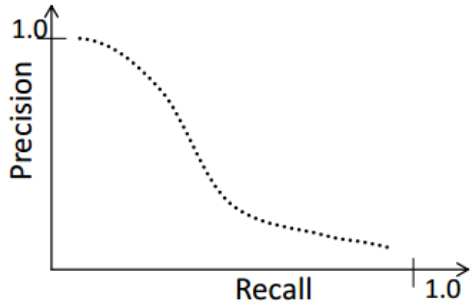
$$\rho = \frac{TP}{TP + FN}$$

Dove con *TP* ci si riferisce ai veri positivi (true positive), documenti classificati correttamente come appartenenti alla classe, con *FP* ai falsi positivi (false positive), documenti classificati incorrettamente come appartenenti alla classe, e con *FN* ai falsi negativi (false negative), oggetti appartenenti alla classe che non sono stati classificati.

Nell'*information retrieval*, la *precision* è la percentuale di documenti ritrovati che sono rilevanti, mentre il *recall* è la percentuale di documenti rilevanti che sono ritrovati. Quindi, la *precision* può essere vista come una misura di correttezza e il richiamo come una misura di completezza. Le due misure sono grandezze inversamente proporzionali, nel senso che all'aumentare di una, diminuirà l'altra. Ne consegue che classificatori "perfetti", che riescano cioè a categorizzare tutti i documenti nella classe corretta, non siano realizzabili.



Curva ideale della precision e del recall



Curva tipica della precision e del recall

Spesso può essere utile una misura che comprenda sia *precision* che *recall*: la *f-measure* (*F1*), media armonica delle due:

$$F1 = 2 \times \frac{TP}{(2TP + FP + FN)} = 2 \times \frac{\pi \times \rho}{\pi + \rho}$$

Inoltre, una misura importante è l'*accuracy* del classificatore, cioè la quantità di classificazioni corrette sul totale dei casi.

$$A = \frac{TP + TN}{TP + TN + FP + FN}$$

Tuttavia, questa metrica può rivelarsi inadeguata in un caso di dataset sbilanciato come quello della "Declutter Challenge". Viene comunque riportata essendo una metrica molto intuitiva, ma non verrà considerata per la valutazione dei classificatori.

Per ottenere stime di precision, recall, f-measure e accuracy si adottano 2 metodi differenti:

- Micro media (*microaverage*)
    - *precision*

$$\hat{\pi}^{\mu} = \frac{TP}{TP + FP} = \frac{\sum_{i=1}^{|\mathcal{C}|} TP_i}{\sum_{i=1}^{|\mathcal{C}|}(TP_i + FP_i)}$$

    - *recall*

$$\hat{\rho}^{\mu} = \frac{TP}{TP + FN} = \frac{\sum_{i=1}^{|\mathcal{C}|} TP_i}{\sum_{i=1}^{|\mathcal{C}|}(TP_i + FN_i)}$$

    - *f-measure*

$$\hat{F}^{\mu} = \frac{2 \times (\hat{\pi}^{\mu} \times \hat{\rho}^{\mu})}{(\hat{\pi}^{\mu} + \hat{\rho}^{\mu})}$$

    - *accuracy*

$$\hat{A}^{\mu} = \frac{(\mathrm{TP} + \mathrm{TN})}{(\mathrm{TP} + \mathrm{FP} + \mathrm{TN} + \mathrm{FN})}$$

Dove π e ρ sono ottenute sommando il numero di *TP*, *FP* e *FN* per ogni classe. La micro f-measure è ottenuta calcolando la media armonica tra micro precision e micro recall. La micro accuracy è la quantità di classificazioni corrette sul totale dei casi.

- Macro media (*macroaverage*)

- *precision*

$$\hat{\pi}^M = \frac{\sum_{i=1}^{|\mathcal{C}|} \hat{\pi}_i}{|\mathcal{C}|}$$

- *recall*

$$\hat{\rho}^M = \frac{\sum_{i=1}^{|\mathcal{C}|} \hat{\rho}_i}{|\mathcal{C}|}$$

- *f-measure*

$$\hat{F}^M = \frac{2 \times (\hat{\pi}^M \times \hat{\rho}^M)}{(\hat{\pi}^M + \hat{\rho}^M)}$$

- *accuracy*

$$\hat{A}^M = \frac{\sum_{i=1}^{|\mathcal{C}|} \hat{A}_i}{|\mathcal{C}|}$$

Dove *precision, recall, f-measure* e *accuracy* sono prima calcolati "localmente" per ogni categoria, e poi "globalmente" facendo la media dei risultati delle diverse categorie.

Questi due metodi (micro e macro) possono dare risultati molto diversi, specialmente se le categorie sono molto sbilanciate. L'uso di uno o dell'altro dipende, comunque, dai bisogni applicativi. Nel caso del dataset della "Declutter Challenge" il dataset è molto sbilanciato, perciò sono state utilizzate le metriche con micro media.

Un'altra metrica molto utile in caso di classificazione binaria e dataset sbilanciato è il *Coefficiente di correlazione di Matthews* [21]. *Accuracy* and *F1* possono mostrare risultati troppo ottimisti specialmente in dataset sbilanciati. Il *Coefficiente di correlazione di Matthews* (MCC) è una statistica più affidabile che produce un risultato elevato solo se le predizioni ottenute segnano un buon risultato in tutte e quattro le categorie della matrice di confusione (*true positive*, *false negative*, *true negative*, e *false positive*), proporzionalmente al numero di esempi sia positivi, sia negativi del dataset.

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}}$$

Tuttavia, essendo la metrica *F1* utilizzata per valutare i risultati della competizione *Kaggle* relativa alla "Declutter Challenge**"**, anche in questo studio le performance vengono valutate principalmente attraverso l'*F1*.

Le formule sono state calcolate a partire dalla matrice di confusione generata dal classificatore. Le interpretazioni delle due matrici, rispettivamente per la classe "NO" e "YES" sono:

Predetti / Annotati	Non-information = Yes	Non-information = No
Non-information = Yes	TP	FN
Non-information = No	FP	TN

4.2. Validazione

Per la valutazione delle performance dei modelli proposti non è disponibile un test set etichettato sul quale calcolare le metriche. Per questo motivo il training set deve essere diviso in un training set ed un test set.

Una procedura molto utilizzata per stimare o confrontare le performance di algoritmi di classificazione su un dataset è la *K-Fold cross validation* [23].

Questa procedura divide in modo casuale il dataset in K sottoinsiemi disgiunti (folds), e a turno ogni sottoinsieme viene utilizzato per testare il modello e i restanti $K-1$ sottoinsiemi per effettuare training del modello.

Le performance del classificatore vengono valutate facendo la media delle metriche calcolate sulle varie *fold.* K è stato impostato al valore di 10.

4.3. Apprendimento supervisionato

L'approccio utilizzato per il task del decluttering dei commenti è di tipo supervisionato. Dato un insieme di esempi di training, sotto forma di coppie input-output, un task di *apprendimento supervisionato* [24] consiste nel predire l'output di nuovi esempi di cui è fornito solo l'input. Questo genere di task è chiamato *classificazione* se l'output da predire

è di tipo discreto, *regressione* se l'output da predire è di tipo continuo. Il decluttering dei commenti in particolare è un task di classificazione in quanto l'output da predire è binario, quindi discreto.

In un task di apprendimento supervisionato, al classificatore vengono forniti:

- Un insieme di features di input $X_1, \dots, X_n$
- Un insieme di features obiettivo $Y_1, \dots, Y_n$
- Un insieme di esempi di training, in cui sono presenti i valori per le features di input e le features obiettivo (training set).
- Un insieme di esempi, in cui sono presenti solo i valori per le features di input (test set).

L'obiettivo è predire i valori delle features obiettivo per esempi mai visti prima.

4.4. Classificatori

I classificatori utilizzati rispecchiano quelli descritti nello stato dell'arte dei classificatori per la classificazione del testo (sezione 2.2).

In particolare, si è scelto di utilizzare i seguenti classificatori in diverse varianti e realizzazioni:

- Naive Bayes
- SVM
- Albero decisionale
- Percettrone
- Ensemble

Un approccio in particolare tra quelli Ensamble, che è stato utilizzato per effettuare la classificazione è quello del voting. In particolare si parla di:

- Hard Voting (o majority voting), in cui gli output binari dei k classificatori vengono aggregati e la decisione che raggiunge la maggioranza $\left(\frac{k+1}{2}\right)$ viene presa.
- Soft Voting, in cui gli output dei classificatori non sono binari ma livelli di confidenza (in percentuale). Anche in questo caso gli output vengono aggregati e la decisione che raggiunge la maggioranza $\left(\frac{k+1}{2}\right)$ viene presa.

5. Implementazione

I metodi proposti sono tre: il primo utilizza features extratestuali per la classificazione dei commenti, il secondo e il terzo utilizzano sia le features testuali, sia quelle extratestuali.

In particolare, si è scelto di considerare come features extratestuali:

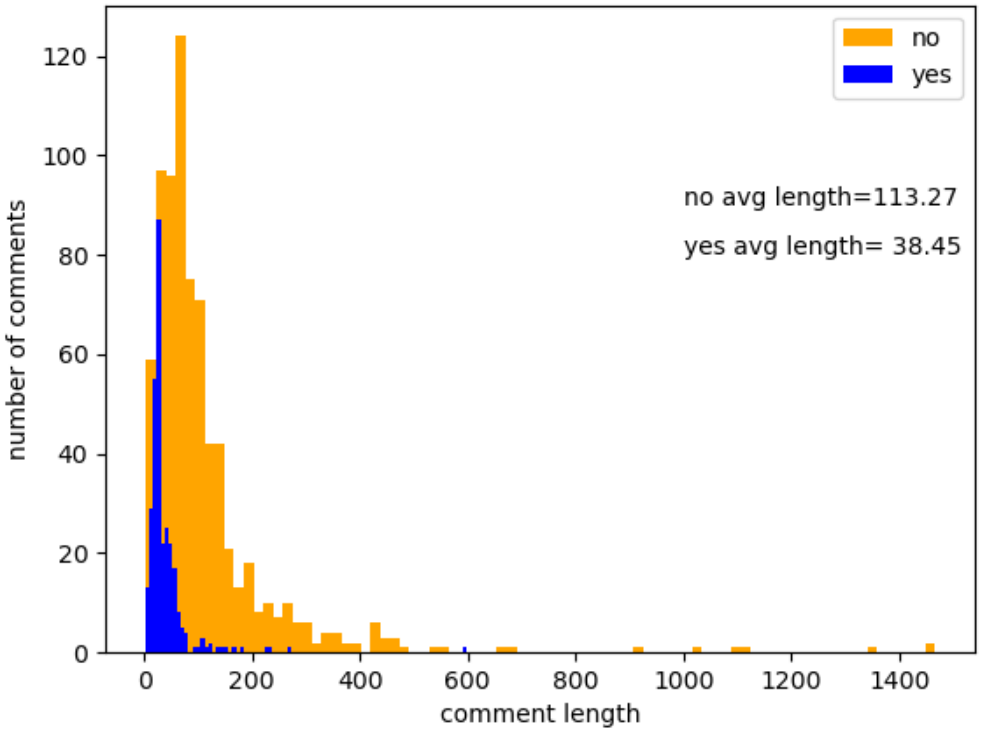
- Lunghezza
- Indice di Jaccard
- Posizione
- Tags
- Tipo

Le motivazioni della scelta di queste features sono approfondite nel capitolo 4.

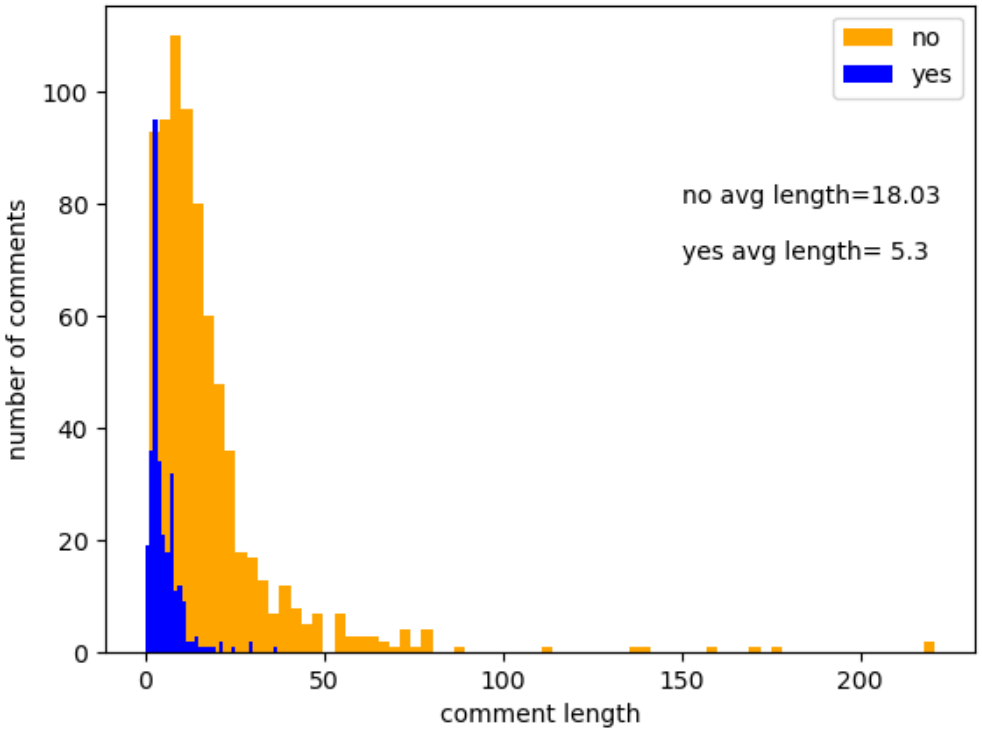
5.1. Lunghezza

Da una analisi della lunghezza dei commenti emerge che, mediamente, i commenti informativi sono più lunghi di quelli non informativi. Tra le feature sono state inserite due tipi di lunghezza:

- Lunghezza come numero di lettere
- Lunghezza come numero di parole (separazione sul carattere spazio/blank)



Analisi lunghezza (numero lettere) commenti



Analisi lunghezza (numero di parole) commenti

5.2. Indice di Jaccard

L'*indice di Jaccard* [25] viene utilizzato per misurare quanto sono simili due insiemi. Dati due insiemi A e B, l'*indice di Jaccard* è:

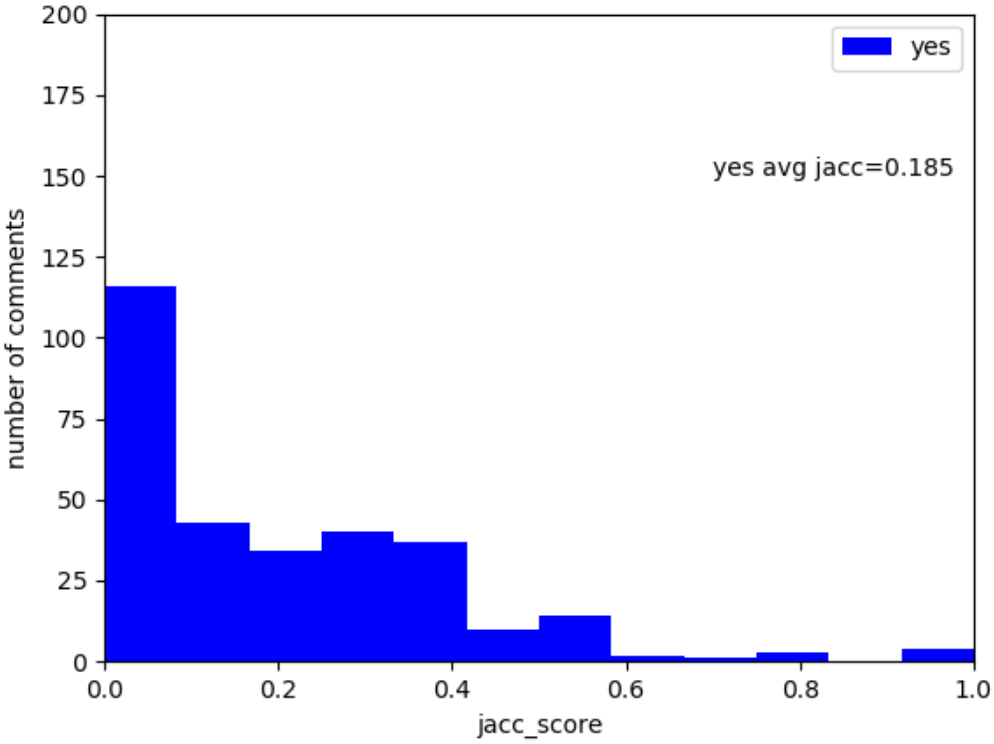
$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

L'idea è quella di misurare la "sovrapposizione" tra le parole dei commenti e token del codice. Se i due insiemi sono molto sovrapposti (utilizzano le stesse parole), il commento non aggiunge nulla al codice, quindi non è informativo. I due insiemi corrispondono al commento e alla riga di codice a cui si riferisce. Se l'*indice di Jaccard* è alto, probabilmente i nomi di variabili, metodi, funzioni, ecc. sono auto-esplicativi e il commento è ridondante. Mediamente l'*indice di Jaccard* è più alto nei commenti non informativi.

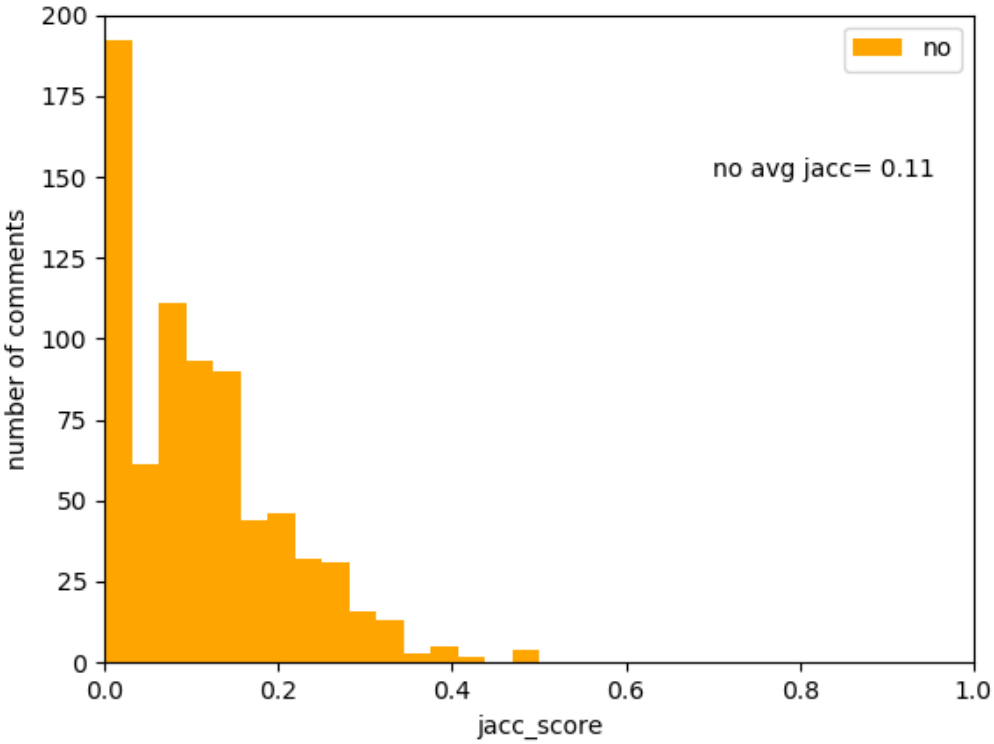
Nella tokenizzazione del codice vengono effettuati stemming e rimozione delle parole chiave del linguaggio.

Le parole chiave vengono rimosse perché raramente si trovano nei commenti e contribuiscono a far aumentare il denominatore per i commenti utili.

Un approccio simile è descritto da D. J. Lawrie, H. Feild [26].



Analisi indice di Jaccard su commenti "Non-information=Yes"



Analisi indice di Jaccard su commenti "Non-information=No"

5.3. Linea a cui si riferisce il commento

Per calcolare l'*indice di Jaccard* è necessario sapere a quale porzione di codice si riferisce il commento. Per convenzione, i commenti dovrebbero riferirsi alla linea successiva in quanto il loro ruolo è quello di aiutare nella lettura del codice o di evitarla. Tuttavia, nel caso di *line* e *block* comment, il commento potrebbe riferirsi alla stessa linea in cui si trova. L'approccio utilizzato è il seguente:

1. Viene selezionata la linea in cui si trova il commento se non è vuota (se c'è solo il commento è considerata vuota).
2. Se la linea in cui si trova è l'inizio di un blocco *else*, *try* o *finally* o se è vuota, si va alla successiva finché sono vuote o contengono altri commenti.

   Una riga con solo un *else*, *try* o *finally* non fornisce alcuna informazione. Quindi conviene cercare nei dintorni.
3. Se il commento si trova in un blocco vuoto ({\\commento}), si prende la prima linea non vuota che precede il commento.

   Se il blocco è vuoto e c'è un commento all'interno, è probabile che il commento di riferisca ad una eventuale condizione che precede il blocco.
4. Se nelle linee successive al commento si arriva a fine blocco ("}"), si ricerca una linea rilevante nelle righe che precedono il commento, seguendo gli stessi criteri di cui sopra.
5. Se si arriva a fine o inizio file si seleziona la stringa vuota.

Alcuni esempi:

```
return Optional.of(suffix); // return the first one we found, anyway.

}
```

Il commento si trova in una linea non vuota, viene quindi selezionata la stessa riga in cui si trova

```
@Override
public void abort(Executor executor) throws SQLException {

// Auto-generated method stub

}
```

Il commento (in azzurro) si trova all'interno di un blocco vuoto, viene quindi selezionata la prima riga non vuota che si trova prima del commento (in giallo)

```
public String getSchema() throws SQLException {

// Auto-generated method stub

return null;

}
```

Viene selezionata la prima riga non vuota sotto il commento

```
} else { //Dir must be a folder, not a file

if (!Files.isDirectory(directory)) {

directory = directory.getParent();

}
```

Viene analizzata la linea in cui si trova il commento. La linea contiene solo un else che non da alcuna informazione. Si cerca quindi la prima riga non vuota al di sotto del commento (in giallo)

5.4. Posizione commenti

Conoscendo la linea di codice a cui si riferisce il commento, possiamo capire in che posizione si trova. Le posizioni che vengono rilevate sono:

- Dichiarazione di un metodo
- Dichiarazione di una classe
- Dichiarazione di una variabile
- Dichiarazione di un enum
- Assegnazione
- Chiamata a metodo
- Return
- Ciclo
- If
- Catch
- Else
- Continue e Break
- Requires
- Dichiarazione di import e package

Queste posizioni rispecchiano i costrutti del linguaggio *Java,* in particolare ci si è concentrati su quelli effettivamente presenti nel dataset.

Le posizioni vengono individuate attraverso espressioni regolari.

5.5. Tags

Da una analisi dei tags presenti nei commenti del training set, è emerso in particolare che, in presenza del tag "link", i commenti sono quasi sempre informativi. La presenza o meno del tag "link " è stata inserita come feature.

label	param	return	see	link	Override	implNote	code	inheritDoc
yes	5	4	0	3	0	1	0	1
no	6	7	2	74	1	1	7	0

5.6. Tipo

Il tipo di commento viene fornito direttamente nel dataset. I tipi possibili sono:

- Line
- Block
- Javadoc

Le differenze tra i vari tipi di commenti sono approfondite nella sezione 3.2

5.7. Modelli proposti

Il primo modello proposto utilizza esclusivamente le features extratestuali di cui si è parlato nelle sezioni precedenti.

Il secondo approccio prevede l'utilizzo della matrice document-term pesata con *tf-idf* a cui vengono aggiunte le features extratestuali.

Il terzo approccio utilizza i conteggi (*word-count*) delle parole presenti all'interno dei commenti insieme alle features extratestuali.

Nel secondo e terzo modello, le matrici delle features testuali vengono normalizzate, mentre la matrice delle features viene standardizzata. Successivamente le matrici vengono unite.

6. Strumenti utilizzati

Per implementare il declutter dei commenti è stato utilizzato il linguaggio Python 3.8

Le librerie utilizzate:

Libreria	**Scopo**
matplotlib==3.2.1	Per disegnare grafici
mlxtend==0.17.2	Per alcuni metodi di feature selection wrapper
nltk==3.5	Per effettuare lo stemming e per una lista di stopwords inglesi
numpy==1.18.4	Per alcune funzioni matematiche utili
pandas==1.0.3	Per la gestione dei file .csv
regex==2020.5.14	Per individuare la posizione dei commenti e per il tokenizzatore
requests==2.23.0	Per scaricare il codice da Github
scikit-learn==0.23.1	Per la classificazione
scipy==1.4.1	Per l'utilizzo di matrici sparse
seaborn==0.10.1	Per disegnare matrici di confusione

6.1. Baseline

Per il classificatore "banale" è stato utilizzato DummyClassifier dalla libreria Scikit-Learn di Python, con parametro strategy="most_frequent".

Per il classificatore con il modello Bag-of-words è stato utilizzato Tfidf-Vectorizer della libreria Scikit-Learn di Python. Il tokenizzatore è stato creato appositamente per il task ed è disponibile su Github, nella repository peppocola/DeClutter-Challenge-2020 3

6.2. Tokenizzazione

Nel processo di tokenizzazione del testo vengono rimossi caratteri separatori, operatori matematici e vengono separate le parole in Camel-case-notation.

Questo perchè la Camel-case-notation è una convenzione utilizzata per il linguaggio Java e nel repository in esame viene rispettata. Nei commenti spesso si trovano parti di codice quindi ha senso effettuare questa separazione.

Non vengono rimosse le stop-words e non vengono effettuate operazioni di lemmatizzazione e stemming.

[3] https://github.com/peppocola/DeClutter-Challenge-2020

6.3. Classificazione

I classificatori utilizzati sono delle implementazioni o delle varianti di quelli presentati all'interno stato dell'arte della Text-classification:

- BernoulliNB
- ComplementNB
- MultinomialNB
- LinearSVC
- SVC
- MLPClassifier
- RandomForestClassifier
- AdaBoostClassifier
- BaggingClassifier
- ExtraTreesClassifier
- GradientBoostingClassifier
- LogisticRegression
- DecisionTreeClassifier
- SGDClassifier
- SoftVoting (votazione pesata sul livello di confidenza delle predizioni dei 5 classificatori con miglior f1-score sulla classe di minoranza)
- HardVoting (votazione maggioritaria a partire dalle predizioni dai 5 classificatori con miglior f1-score sulla classe di minoranza)

In particolare, nella strategia di voting si è scelto di "puntare" sui classificatori che si comportassero meglio nel predire esempi della classe di minoranza, essendo quelli più difficili da predire proprio per la scarsità di esempi all'interno del dataset.

Avendo standardizzato i dati nei modelli proposti, è possibile che ci siano dei numeri negativi nella matrice delle features. Perciò i classificatori MultinomialNB e ComplementNB sono stati utilizzati solo per il modello Bag-of-words.

7. Sperimentazione

Non essendoci strumenti con cui confrontare le prestazioni, sono state costruite delle baseline che i classificatori proposti devono superare per essere considerati validi.

Le baseline sono:

- Classificatore che predice sempre la classe di maggioranza
- Classificatore che sfrutta il modello Bag-of-words

Il primo scelto perché un qualsiasi classificatore, per essere valido, deve superare le performance di uno "banale" che predice sempre la classe più numerosa.

Il secondo scelto perchè è un semplice approccio utilizzato nel campo della Text-Classification. Il modello Bag-of-words ci serve per confrontare un approccio che si basa solamente sulle parole del testo con un modello che invece tiene conto di altre caratteristiche.

La domanda di ricerca alla quale si vuole rispondere con questo studio è: è possibile ottenere prestazioni migliori di queste semplici baseline utilizzando features extratestuali? Si possono ottenere predizioni più efficienti osservando il codice a cui si riferiscono i commenti?

7.1. Risultati

MODELLO BAG-OF-WORDS (TF-IDF):

classifier	accuracy	precision	recall	f1-score	matthews corrcoef
DummyClassifier	0.71	0.35	0.5	0.41	0.0
BernoulliNB	0.74	0.72	0.76	0.72	0.48
ComplementNB	0.76	0.71	0.67	0.68	0.37
MultinomialNB	0.77	0.83	0.61	0.61	0.38
LinearSVC	0.79	0.77	0.68	0.7	0.44
SVC (rbf)	0.79	0.84	0.65	0.66	0.44
MLPClassifier	0.78	0.74	0.72	0.73	0.46
RandomForestClassifier	0.8	0.77	0.73	0.74	0.49
AdaBoostClassifier	0.78	0.74	0.7	0.71	0.44
BaggingClassifier	0.79	0.75	0.72	0.73	0.46
ExtraTreesClassifier	**0.82**	**0.79**	**0.75**	**0.76**	**0.53**
GradientBoostingClassifier	0.8	0.8	0.69	0.71	0.48
LogisticRegression	0.78	0.82	0.63	0.63	0.4
DecisionTreeClassifier	0.77	0.72	0.73	0.72	0.45
SGDClassifier	0.79	0.76	0.7	0.72	0.46
SoftVoting	**0.81**	**0.77**	**0.76**	**0.76**	**0.52**
HardVoting	**0.81**	**0.78**	**0.75**	**0.76**	**0.52**

Prestazioni dei classificatori col modello Bag-of-words

classifier	precision no	recall no	f1 no	precision yes	recall yes	f1 yes
DummyClassifier	0.71	1.0	0.83	0.0	0.0	0.0
BernoulliNB	0.9	0.72	0.8	0.54	0.8	0.65
ComplementNB	0.8	0.89	0.84	0.62	0.44	0.51
MultinomialNB	0.76	0.99	0.86	0.89	0.24	0.37
LinearSVC	0.8	0.94	0.86	0.74	0.43	0.54
SVC (rbf)	0.78	0.98	0.87	0.89	0.31	0.45
MLPClassifier	0.84	0.86	0.85	0.63	0.58	0.6
RandomForestClassifier	0.83	0.91	0.87	0.71	0.55	0.62
AdaBoostClassifier	0.82	0.88	0.85	0.65	0.53	0.57
BaggingClassifier	0.83	0.89	0.86	0.67	0.55	0.6
ExtraTreesClassifier	**0.84**	**0.92**	**0.88**	**0.74**	**0.58**	**0.65**
GradientBoostingClassifier	0.8	0.96	0.87	0.79	0.43	0.55
LogisticRegression	0.77	0.98	0.86	0.86	0.28	0.41
DecisionTreeClassifier	0.85	0.83	0.84	0.6	0.63	0.61
SGDClassifier	0.82	0.92	0.86	0.71	0.49	0.58
SoftVoting	**0.86**	**0.88**	**0.87**	**0.68**	**0.64**	**0.66**
HardVoting	**0.84**	**0.9**	**0.87**	**0.71**	**0.59**	**0.65**

Prestazioni dei classificatori per classe col modello Bag-of-words

Con questo tipo di approccio i classificatori Voting e ExtraTrees si comportano meglio, raggiungendo $f1 = 0.76$. Tutti i classificatori hanno performance migliori nel riconoscere i commenti informativi rispetto ai commenti non informativi. Una causa è il dataset sbilanciato che presenta più esempi di commenti informativi.

In seguito all'utilizzo del modello Bag-of-word, è stata effettuata una analisi delle features del classificatore. Dopo una valutazione basata su Information Gain, è emerso che le stopwords sono tra le parole più utili nella classificazione dei commenti.

feature	IG
the	0.2593048812314631
to	0.1437949681334318
a	0.10753760807434255
is	0.08667756619302583
method	0.08423403623162272
for	0.07971860214701591
of	0.07720682418524089
if	0.0760782901808569
auto	0.07507110882498183
and	0.07391573266404008
in	0.07251097926505225
this	0.07107578421584607
generated	0.06890218954876381
stub	0.06758436656908315
entry	0.04762104309261849
file	0.047555547946109875
it	0.046807599622539005
be	0.044616531849853126
set	0.04263989040489038

Top 20 features per information gain

Questa evidenza ci suggerisce che probabilmente i commenti meno schematici (che quindi contengono stopwords) sono più ricchi di informazione e ci forniscono informazioni aggiuntive rispetto a commenti più schematici.

Alcuni esempi estratti dal dataset:

- icon.setToolTipText(printedViewModel.getLocalization());

Questo commento, come quasi la totalità dei commenti nei quali c'è solo codice, è etichettato come Non-information=Yes. Il commento non è affatto discorsivo e non contiene stopwords.

- Synchronize changes of the underlying date value with the temporalAccessorValue

  Questo commento, invece, è etichettato come Non-information=No. Il commento è discorsivo e contiene le stopwords of, the, with.

MODELLO CON FEATURES EXTRATESTUALI:

classifier	accuracy	precision	recall	f1-score	matthews corrcoef
DummyClassifier	0.71	0.35	0.5	0.41	0.0
BernoulliNB	0.7	0.69	0.73	0.68	0.41
LinearSVC	0.81	0.78	0.74	0.76	0.52
SVC (poly degree=2)	0.74	0.75	0.58	0.57	0.27
MLPClassifier	0.83	0.79	0.77	0.78	0.57
RandomForestClassifier	0.82	0.78	0.76	0.77	0.54
AdaBoostClassifier	**0.84**	**0.82**	**0.78**	**0.8**	**0.6**
BaggingClassifier	0.82	0.78	0.76	0.77	0.54
ExtraTreesClassifier	0.81	0.77	0.75	0.76	0.52
GradientBoostingClassifier	0.83	0.8	0.77	0.78	0.57
LogisticRegression	0.81	0.78	0.75	0.76	0.53
DecisionTreeClassifier	0.78	0.73	0.73	0.73	0.45
SGDClassifier	0.8	0.77	0.74	0.74	0.5
SoftVoting	0.83	0.8	0.77	0.78	0.57
HardVoting	0.84	0.81	0.78	0.79	0.59

Prestazioni dei classificatori con le features extratestuali

classifier	precision no	recall no	f1 no	precision yes	recall yes	f1 yes
DummyClassifier	0.71	1.0	0.83	0.0	0.0	0.0
BernoulliNB	0.88	0.67	0.76	0.49	0.78	0.6
LinearSVC	0.84	0.9	0.87	0.71	0.59	0.64
SVC (poly degree=2)	0.74	0.97	0.84	0.75	0.18	0.29
MLPClassifier	0.86	0.9	0.88	0.73	0.65	0.68
RandomForestClassifier	0.85	0.9	0.87	0.71	0.61	0.66
AdaBoostClassifier	**0.87**	**0.91**	**0.89**	**0.77**	**0.65**	**0.7**
BaggingClassifier	0.85	0.9	0.88	0.72	0.62	0.66
ExtraTreesClassifier	0.85	0.89	0.87	0.69	0.62	0.65
GradientBoostingClassifier	0.86	0.9	0.88	0.73	0.64	0.68
LogisticRegression	0.84	0.9	0.87	0.72	0.59	0.65
DecisionTreeClassifier	0.84	0.85	0.84	0.61	0.61	0.61
SGDClassifier	0.84	0.89	0.86	0.7	0.58	0.62
SoftVoting	0.86	0.91	0.88	0.74	0.64	0.68
HardVoting	0.86	0.91	0.89	0.75	0.65	0.7

Prestazioni dei classificatori con le features per classe

Il classificatore AdaBoost raggiunge $f1 = 0.80$, un incremento di 0.03 rispetto al Voting con il modello Bag-of-words.

È interessante come il classificatore basato sulle features extratestuali performi meglio rispetto a quello basato sul modello Bag-of-words.

Il classificatore che sfrutta il modello Bag-of-words, su questo dataset, utilizza 2078 features (che corrispondono ai diversi token presenti nei commenti)

Quello basato sulle sole features extratestuali, invece, ne utilizza solamente 6.

MODELLO CON FEATURES BAG-OF-WORDS (TF-IDF) E FEATURES EXTRATESTUALI:

classifier	accuracy	precision	recall	f1-score	matthews corrcoef
DummyClassifier	0.71	0.35	0.5	0.41	0.0
BernoulliNB	0.73	0.72	0.77	0.72	0.49
LinearSVC	0.83	0.8	0.77	0.79	0.58
SVC (poly degree=2)	0.74	0.71	0.58	0.57	0.25
MLPClassifier	0.8	0.76	0.75	0.75	0.51
RandomForestClassifier	0.84	0.83	0.77	0.79	0.6
AdaBoostClassifier	**0.84**	**0.81**	**0.8**	**0.81**	**0.62**
BaggingClassifier	0.84	0.82	0.78	0.79	0.6
ExtraTreesClassifier	0.83	0.81	0.74	0.76	0.55
GradientBoostingClassifier	0.85	0.83	0.79	0.8	0.61
LogisticRegression	0.82	0.79	0.76	0.77	0.54
DecisionTreeClassifier	0.81	0.77	0.78	0.77	0.55
SGDClassifier	0.83	0.8	0.79	0.79	0.59
SoftVoting	**0.85**	**0.83**	**0.8**	**0.81**	**0.62**
HardVoting	**0.85**	**0.83**	**0.79**	**0.81**	**0.62**

Prestazioni dei classificatori con tfidf e features extratestuali

classifier	precision no	recall no	f1 no	precision yes	recall yes	f1 yes
DummyClassifier	0.71	1.0	0.83	0.0	0.0	0.0
BernoulliNB	0.92	0.69	0.78	0.53	0.85	0.65
LinearSVC	0.86	0.91	0.88	0.75	0.64	0.69
SVC (poly degree=2)	0.74	0.96	0.84	0.68	0.2	0.3
MLPClassifier	0.85	0.87	0.86	0.67	0.63	0.65
RandomForestClassifier	0.85	0.94	0.89	0.82	0.6	0.69
AdaBoostClassifier	**0.88**	**0.9**	**0.89**	**0.75**	**0.7**	**0.72**
BaggingClassifier	0.86	0.92	0.89	0.77	0.64	0.7
ExtraTreesClassifier	0.84	0.94	0.88	0.79	0.54	0.64
GradientBoostingClassifier	0.87	0.93	0.9	0.79	0.65	0.71
LogisticRegression	0.85	0.9	0.87	0.72	0.62	0.66
DecisionTreeClassifier	0.87	0.86	0.87	0.67	0.69	0.68
SGDClassifier	0.88	0.88	0.88	0.71	0.7	0.7
SoftVoting	**0.87**	**0.92**	**0.9**	**0.78**	**0.67**	**0.72**
HardVoting	**0.87**	**0.93**	**0.9**	**0.79**	**0.66**	**0.72**

Prestazioni per classe dei classificatori con il tf-idf e le features extratestuali

I classificatori AdaBoost e Voting raggiungono $f1 = 0.81$. Un leggero incremento rispetto ai classificatori con le sole features extratestuali.

MODELLO CON FEATURES BAG-OF-WORDS (WORD-COUNT) E FEATURES EXTRATESTUALI:

classifier	accuracy	precision	recall	f1-score	matthews corrcoef
DummyClassifier	0.71	0.36	0.5	0.42	0.0
BernoulliNB	0.78	0.73	0.74	0.73	0.47
LinearSVC	0.83	0.8	0.77	0.78	0.57
SVC (poly degree=2)	0.75	0.76	0.59	0.58	0.3
MLPClassifier	0.8	0.75	0.74	0.75	0.5
RandomForestClassifier	0.84	0.82	0.77	0.79	0.59
AdaBoostClassifier	0.81	0.78	0.76	0.77	0.54
BaggingClassifier	0.83	0.8	0.77	0.78	0.57
ExtraTreesClassifier	0.83	0.81	0.75	0.77	0.55
GradientBoostingClassifier	0.83	0.81	0.76	0.77	0.57
LogisticRegression	0.81	0.78	0.75	0.76	0.53
DecisionTreeClassifier	0.79	0.75	0.74	0.75	0.49
SGDClassifier	0.81	0.77	0.75	0.76	0.52
SoftVoting	**0.84**	**0.82**	**0.78**	**0.8**	**0.6**
HardVoting	0.83	0.81	0.77	0.78	0.57

Prestazioni dei classificatori con word-count e features extratestuali

classifier	precision no	recall no	f1 no	precision yes	recall yes	f1 yes
DummyClassifier	0.71	1.0	0.83	0.0	0.0	0.0
BernoulliNB	0.86	0.83	0.84	0.61	0.65	0.63
LinearSVC	0.86	0.91	0.88	0.74	0.64	0.68
SVC (poly degree=2)	0.75	0.97	0.85	0.77	0.2	0.32
MLPClassifier	0.85	0.87	0.86	0.66	0.62	0.63
RandomForestClassifier	0.86	0.94	0.89	0.79	0.61	0.68
AdaBoostClassifier	0.86	0.89	0.87	0.7	0.63	0.66
BaggingClassifier	0.86	0.91	0.88	0.74	0.62	0.68
ExtraTreesClassifier	0.84	0.94	0.89	0.77	0.56	0.65
GradientBoostingClassifier	0.85	0.93	0.89	0.78	0.58	0.66
LogisticRegression	0.85	0.91	0.87	0.72	0.59	0.65
DecisionTreeClassifier	0.85	0.86	0.86	0.65	0.62	0.63
SGDClassifier	0.85	0.89	0.87	0.7	0.61	0.64
SoftVoting	**0.86**	**0.93**	**0.89**	**0.78**	**0.63**	**0.7**
HardVoting	0.85	0.93	0.89	0.77	0.6	0.68

Prestazioni per classe dei classificatori con word-count e le features extratestuali

7.2. Risultati competizione Kaggle (in corso)

Il *declutter* costruito è stato utilizzato per partecipare alla "Declutter Challenge" svoltasi su Kaggle.

Per la competizione è stato fornito un training set più ampio (1311 commenti) ed un test set (260 commenti).

Per partecipare alla challenge si è deciso di selezionare i 5 migliori classificatori per f1-score sulla classe di minoranza per ogni modello:

Modello	**Classificatori**
Features testuali ed extratestuali	SDGClassifier LinearSVC GradientBoostingClassifier AdaBoostClassifier BaggingClassifier
Features extratestuali	LogisticRegression MLPClassifier RandomForestClassifier GradientBoostingClassifier AdaBoostClassifier
Bag-of-words	MLPClassifier BaggingClassifier BernoulliNB ExtraTreesClassifier RandomForestClassifier

Le predizioni di questi classificatori vengono combinate attraverso le tecniche di Hard e Soft Voting.

In particolare, ci si è concentrati sul modello risultato più promettente, cioè quello con features testuali ed extratestuali, facendo tuning sui 5 classificatori migliori per quel modello.

I risultati riportati sono valutati esclusivamente sul 50% del test set, essendo la competizione non ancora conclusa.

La metrica utilizzata da Kaggle è la *F1*.

Modello	**Voting**	**F1-Score**
Bag-of-Words (word-count) e features extratestuali	hard	0.81
Bag-of-Words (tf-idf) e features extratestuali	hard	0.80
Features extratestuali	hard	0.79
Bag-of-Words (word-count) e features extratestuali	soft	0.77
Bag-of-Words (tf-idf) e features extratestuali	soft	0.77
Features extratestuali	soft	0.76
Bag-of-Words (tf-idf)	hard	0.74
Bag-of-Words (tf-idf)	soft	0.73

Ad oggi 06/07/2020 risultano iscritti alla “Declutter Challenge” 6 teams.

La classifica pubblica è la seguente e vede il risultato di questo studio al primo posto.

#	Team Name	Notebook	Team Members	Score	Entries	Last
1	just another trickster			0.81538	25	22m
	Your Best Entry ↑ Your submission scored 0.81538, which is not an improvement of your best score. Keep trying!					
2	codeup-yang			0.80769	6	2h
3	BTW			0.76153	9	21d
4	NPW			0.75384	4	21d
5	[Deleted]			0.73846	5	1mo
6	Boda0124			0.73846	4	21d

8. Conclusioni e sviluppi futuri

I classificatori che utilizzano le features si sono rivelati più performanti di quelli basati sul modello Bag-of-words. Nonostante le dimensioni ridotte del dataset e la bassa presenza di esempi di commenti non informativi all'interno del training set, i risultati sono incoraggianti.

Le features individuate si sono rivelate utili nella classificazione dei commenti come informativi o non informativi.

Un altro aspetto importante riguarda l'elevata importanza delle stopwords nella classificazione basata su testo. Questo ci indica che i commenti più discorsivi sono generalmente più utili di quelli più schematici nella comprensione del codice.

Tra i possibili sviluppi futuri del presente lavoro di tesi, una prima possibilità potrebbe riguardare l'aggiunta del supporto ad altri linguaggi di programmazione (attualmente il software sviluppato funziona solamente con codice Java). Si potrebbe inoltre aggiungere il supporto ad altre lingue (attualmente il tokenizzatore rimuove solo le stopwords inglesi dal testo e non è possibile selezionare una lingua).

Idealmente, lo studio andrebbe replicato su dataset più numerosi e bilanciati, valutando nuovamente la performance dell'approccio implementato. Inoltre, si potrebbe pensare di integrare il tool di "declutter" in IDE di sviluppo, per supportare i programmatori nella documentazione del proprio software. In tal senso, si potrebbe utilizzare il "declutter" anche per misurare la percentuale di commenti non informativi nella valutazione della qualità del software.

9. Ringraziamenti

Ringrazio l'asfalto per ogni salto che mi ha lasciato fare
Ringrazio le gambe, le mie e le altre, per ogni tuffo al mare
E dire grazie ogni tanto, d'altro canto non si può evitare

Grazie ai miei relatori, prof.ssa *Nicole Novielli* e prof. *Pierpaolo Basile*. Siete dei punti di riferimento per me (ma anche dei fantastici colleghi di Desk di registrazione). Vi ringrazio per avermi seguito con cura e per avermi trasmesso la vostra passione.

Grazie alla mia famiglia, per avermi sostenuto in ogni modo, senza inutili pressioni ma con tanto affetto e fiducia.

Grazie a *Valentina*. Grazie per esserci sempre e per credere in me più di chiunque altro.

Grazie ai miei compagni di viaggio:

Antonio F., Daniela, Eleonora, Gianluca.

Grazie ad *Antonio G*. Mi hai insegnato tantissime cose e mi hai sopportato *quasi* sempre! Strano non vederti nella lista dei prof del dipartimento!

Grazie a *Paolo,* un amico leale, sincero, divertente e ospitale. E hai anche meno capelli bianchi di me…

Grazie a *Lorenzo*. Sei fonte di ispirazione, ti stimo molto!

Grazie a *Simona,* la mia prima vera amica.

Grazie a chi c'è *post-pranzo*, *post-cena* e non solo: *Nicola*, *Pasquale*, *Giuseppe*, *Francesco*, *Gianvito, Filippo*.

Grazie a chi ha contribuito, nel bene e nel male, alla mia crescita.

Ti dico grazie e suono più forte
E suono ancora più forte

Bibliografia

[1] T. Tenny, “Program Readability: Procedures Versus Comments,” IEEE Trans. Softw. Eng., vol. 14, no. 9, 1988.

[2] S. N. Woodfield, H. E. Dunsmore, and V. Y. Shen, “The effect of modularization and comments on program comprehension,” ser. ICSE ’81, 1981.

[3] S. C. B. de Souza, N. Anquetil, and K. M. de Oliveira, “A Study of the Documentation Essential to Software Maintenance,” ser. SIGDOC ’05, 2005.

[4] M. J. B. Garc´ıa and J. C. G. Alvarez, “Maintainability as a Key Factor in Maintenance Productivity: A Case Study,” ser. ICSM ’96, 1996.

[5] P. Oman and J. Hagemeister, “Metrics for Assessing a Software System’s Maintainability,” ser. ICSM ’92, 1992.

[6] Maalej, Walid & Robillard, Martin. (2013). Patterns of Knowledge in API Reference Documentation. Software Engineering, IEEE Transactions on. 39. 1264-1282. 10.1109/TSE.2013.12.

[7] Pascarella, L., Bruntink, M. & Bacchelli, A. Classifying code comments in Java software systems. Empir Software Eng 24, 1499–1537 (2019). https://doi.org/10.1007/s10664-019-09694-w

[8] Woodfield SN, Dunsmore HE, Shen VY (1981) The effect of modularization and comments on program comprehension, pp 215–223

[9] Tenny T (1985) Procedures and comments vs. the banker’s algorithm. SIGCSE Bull 17:44–53

[10] Tenny T (1988) Program readability: Procedures versus comments. IEEE Trans Softw Eng 14(9):1271–1279

[11] Hartzman CS, Austin CF (1993) Maintenance productivity: Observations based on an experience in a large system environment, pp 138–170

[12] de Souza SCB, Anquetil N, de Oliveira KM (2005) A study of the documentation essential to software maintenance. In: Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information. ACM, pp 68–75

[13] D. Schreck, V. Dallmeier, and T. Zimmermann. How documentation evolves over time. In Proceedings of the 9th International Workshop on Principles of Software Evolution, September 2007.

[14] Oman P, Hagemeister J (1992) Metrics for assessing a software system's maintainability. In: Conference on Software Maintenance Proceerdings. IEEE, pp 337–344

[15] Garcia MJB, Granja-Alvarez JC (1996) Maintainability as a key factor in maintenance productivity: a case study. In: icsm, pp 87

[16] D. Haouari, H. Sahraoui and P. Langlais, "How Good is Your Comment? A Study of Comments in Java Programs," 2011 International Symposium on Empirical Software Engineering and Measurement, Banff, AB, 2011, pp. 137-146, doi: 10.1109/ESEM.2011.22.

[17] Haouari D, Sahraoui H, Langlais P (2011) How good is your comment? a study of comments in java programs. In: 2011 International Symposium on Empirical Software Engineering and Measurement (ESEM). IEEE, pp 137–146

[18] Steidl D, Hummel B, Juergens E (2013b) Quality analysis of source code comments. In: IEEE 21st International Conference on Program Comprehension (ICPC). IEEE, pp 83–92

[19] https://it.wikipedia.org/wiki/JabRef

[20] C.D. Manning, P. Raghavan, H. Schütze. 2009. An Introduction to Information Retrieval.

[21] Chicco D, Jurman G. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. BMC Genomics. 2020;21(1):6. Published 2020 Jan 2. doi:10.1186/s12864-019-6413-7

[22] Sebastiani, F. (2002). Machine learning in automated text categorization. ACM computing surveys (CSUR), 34(1), 1-47.

[23] Tzu-Tsung Wong. 2015. Performance evaluation of classification algorithms by k-fold and leave-one-out cross validation. Pattern Recogn. 48, 9 (September 2015), 2839–2846. DOI:https://doi.org/10.1016/j.patcog.2015.03.009

[24] David L. Poole and Alan K. Mackworth. 2017. Artificial Intelligence: Foundations of Computational Agents (2nd. ed.). Cambridge University Press, USA.

[25] I. C. Mogotsi. 2010. Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze: Introduction to information retrieval. Inf. Retr. 13, 2 (April 2010), 192–195. DOI:https://doi.org/10.1007/s10791-009-9115-y

[26] D. J. Lawrie, H. Feild, and D. Binkley, "Leveraged Quality Assessment using Information Retrieval Techniques," ser. ICPC '06, 2006.

[27] Yiming Yang and Thorsten Joachims (2008) Text categorization. Scholarpedia, 3(5):4242.

[28] Hayes, P. J., Andersen, P. M., Nirenburg, I. B., And Schmandt, L. M. 1990. Tcs: a shell for content-based text categorization. In Proceedings of CAIA-90, 6th IEEE Conference on Artificial Intelligence Applications (Santa Barbara, CA, 1990), 320–326.

[29] Mitchell, T.M. 1996. Machine Learning. McGraw Hill, New York, NY.

[30] Lewis, D. D. 1998. Naive (Bayes) at forty: The independence assumption in information retrieval. In Proceedings of ECML-98, 10th European Conference on Machine Learning (Chemnitz, Germany, 1998), 4–15.

[31] Schütze, H., Hull, D. A., And Pedersen, J. O. 1995. A comparison of classifiers and document representations for the routing problem. In Proceedings of SIGIR-95, 18th ACM International Conference on Research and Development in Information Retrieval (Seattle, WA, 1995), 229–237.

[32] Wiener, E. D., Pedersen, J. O., And Weigend, A. S. 1995. A neural network approach to topic spotting. In Proceedings of SDAIR-95, 4th Annual Symposium on Document Analysis and Information Retrieval (Las Vegas, NV, 1995), 317–332.

[33] Hull, D. A. 1994. Improving text retrieval for the routing problem using latent semantic indexing. In Proceedings of SIGIR-94, 17th ACM International Conference on Research and Development in Information Retrieval (Dublin, Ireland, 1994), 282–289.

[34] Dagan, I., Karov, Y., And Roth, D. 1997. Mistakedriven learning in text categorization. In Proceedings of EMNLP-97, 2nd Conference on Empirical Methods in Natural Language Processing (Providence, RI, 1997), 55–63.

[35] Ng, H. T., Goh, W. B., And Low, K. L. 1997. Feature selection, perceptron learning, and a usability case study for text categorization. In Proceedings of SIGIR-97, 20th ACM

International Conference on Research and Development in Information Retrieval (Philadelphia, PA, 1997), 67–73.

[36] Lam, S. L. And Lee, D. L. 1999. Feature reduction for neural network based text categorization. In Proceedings of DASFAA-99, 6th IEEE International Conference on Database Advanced Systems for Advanced Application (Hsinchu, Taiwan, 1999), 195–202.

[37] Yang, Y. 1994. Expert network: effective and efficient learning from human decisions in text categorisation and retrieval. In Proceedings of SIGIR-94, 17th ACM International Conference on Research and Development in Information Retrieval (Dublin, Ireland, 1994), 13–22.

[38] Joachims, T. 1998. Text categorization with support vector machines: learning with many relevant features. In Proceedings of ECML-98, 10th European Conference on Machine Learning (Chemnitz, Germany, 1998), 137–142.

[39] Joachims, T. 1999. Transductive inference for text classification using support vector machines. In Proceedings of ICML-99, 16th International Conference on Machine Learning (Bled, Slovenia, 1999), 200–209.

[40] Tumer, K. And Ghosh, J. 1996. Error correlation and error reduction in ensemble classifiers. Connection Sci. 8, 3-4, 385–403.

[41] Li, Y. H. And Jain, A. K. 1998. Classification of text documents. Comput. J. 41, 8, 537–546.