



UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

INFERENZA STATISTICA NELL'ANALISI SULLA CORRELAZIONE TRA FEATURE E COVERAGE DI UN TEST CASE

RELATORE

Prof. Fabio Palomba

Dott.ssa **Valeria Pontillo**

Università degli studi di Salerno

CANDIDATO

Olexiy Lysytsya

Matricola: 0512108083

Sommario

Il test del software è il processo di valutazione e verifica del corretto funzionamento di un'applicazione o di un prodotto software rispetto alle aspettative. I vantaggi del test includono la prevenzione dei bug, la riduzione dei costi di sviluppo e il miglioramento delle prestazioni, oltre che ad offrire al business una prospettiva oggettiva del prodotto considerato. Si stima che almeno la metà dell'intero costo di sviluppo di un progetto sia investita nel processo di testing del prodotto che si sta sviluppando; ragion per cui l'automatizzazione del processo di testing è di forte interesse dalla comunità scientifica da almeno due decenni a questa parte. Come già riscontrato in molteplici altri ambiti, uno degli strumenti dal potenziale più significativo per l'automatizzazione del testing è costituito dall'utilizzo di algoritmi d'intelligenza artificiale adottando tecniche di machine learning.

Un test case è l'elemento atomico protagonista durante la fase di testing, ogni test case ha diverse caratteristiche quali la **line coverage**, una metrica descrittiva del numero di linee di codice coperte del singolo test case rispetto al source project che si sta analizzando.

Nelle seguenti pagine è descritto lo sviluppo di un modello di regressione tale che, date in input diverse caratteristiche di un test case, restituisca la predizione del numero di righe di codice ricoperte dal suddetto caso di test. **Un modello di questo tipo si basa sull'ipotesi che vi sia una correlazione tra le diverse caratteristiche di un test case ed la line coverage.**

Le analisi statistiche descritte nel seguente articolo così come il training del modello sono state eseguite su un dataset consistente in una raccolta di diecimila casi di test inerenti a diciotto diversi progetti software. Il frutto di questa tesi consiste quindi in un modello di regressione tale che, prese in input diverse feature statiche descrittive di un generico caso di test, restituisce in output una variabile dipendente indice della coverage associata al caso di test stesso.

Indice	2
1 Introduzione	1
1.1 Contesto Applicativo	1
1.1.1 Processo di testing	1
1.1.2 Metriche del caso di test (test case)	2
1.2 Motivazioni e Obiettivi	4
1.3 Sviluppo e Risultati ottenuti	4
1.4 Struttura della tesi	5
2 Stato dell'arte	6
2.1 Modello di regressione	6
2.1.1 Regressione lineare semplice	7
2.1.2 Funzione di costo - Cost function	8
2.1.3 Ordinary Least Squares	9
2.1.4 Gradient Descent	10
2.1.5 regolarizzazione	10
2.2 Compromesso tra Bias e Varianza	10
3 Metodologia di Analisi	12
3.1 Descrizione delle features	14
3.2 Analisi delle features	16
3.3 Variabile dipendente	16

3.4	Violin plot	18
3.5	Indice di correlazione di Pearson	20
3.5.1	Risultati	23
4	Il modello	25
4.1	Data Engineering sulla variabile dipendente	26
4.2	Strategia nella selezione del modello	27
4.3	Ridge model	27
4.4	Lasso model	28
4.5	Extreme Gradient Boosting (XGBoost)	29
4.6	Random Forest Regression	31
4.7	Analisi delle feature	35
4.7.1	Random Forest Built-in Feature Importance	35
4.7.2	Permutation Based Feature Importance	36
4.7.3	SHAP Values	37
4.8	Ottimizzazione degli iperparametri	38
5	Conclusioni	40
	Bibliografia	42

1.1 Contesto Applicativo

Il collaudo del software (software testing) è una delle fasi costituenti il processo di sviluppo di un prodotto software[4; 12]. Secondo uno studio del National Institute of Standards and Technology del 2002, si stima che nel complesso i bug nei software abbiano avuto un costo complessivo pari a 59.5 miliardi di dollari l'anno per l'economia statunitense. Più di un terzo di questa spesa potrebbe essere potenzialmente evitata con l'adozione di un testing del software adeguato [22].

Lo sviluppo del software è un processo dispendioso e ad alta intensità di lavoro umano, l'interesse comune è quello di andare a semplificare tale processo in tutte le sue fasi.

L'automazione del testing contribuirebbe quindi sia alla riduzione dei tempi richiesti per lo sviluppo di un prodotto software, che anche ad aumentare l'affidabilità del suddetto prodotto. Questo è possibile agevolando lo sviluppatore tramite strumenti di generazione automatica di casi di test applicabili al codice appena prodotto [3].

1.1.1 Processo di testing

Essere in grado di poter dimostrare l'affidabilità di un prodotto software è una parte fondamentale del processo di sviluppo. Nel tempo sono state sperimentate diverse e nuove strategie nel processo del collaudo, pertanto è ostico darne una descrizione dettagliata.

In un contesto applicativo si potrebbe fare riferimento ad un procedimento del testing costituito dalle seguenti fasi:

- **Testing d'unità (Unit testing)**

Il collaudo di una singola unità del software, della minima componente di un programma dotato di funzionamento autonomo; a seconda del paradigma di programmazione o linguaggio di programmazione considerato, questa componente può corrispondere per esempio ad una singola funzione nella programmazione procedurale, o una singola classe o metodo nella programmazione a oggetti.

- **Testing d'integrazione (Integration testing)**

Nella pratica presa singolarmente un singolo caso di test è fine a se stesso poiché un prodotto software è dato dall'integrazione di molteplici unità ad alta coesione tra loro, le quali combinando le proprie funzioni portano a termine un obiettivo comune. Una volta assicurato il funzionamento delle singole unità di test, esse vengono combinate e testate in un contesto di gruppo, al fine di intercettare eventuali problemi di coupling, oltre che alla verifica di conformità rispetto ai requisiti funzionali.

- **Testing di sistema (System Testing)**

Collaudo dell'intero prodotto software integrato, enfasi sulla verifica di conformità rispetto ai requisiti raccolti e descritti durante il processo di specifica. Il testing di sistema prende in input tutte le componenti che hanno superato il test d'integrazione, col fine di individuare inconsistenze tra l'integrazione stessa delle unità ma anche nel sistema nel complesso.

- **Acceptance testing**

Questa è l'ultima fase prima della consegna al committente del prodotto finale. Viene eseguito un confronto tra il prodotto software ed i requisiti esplicitati dal committente del progetto.

1.1.2 Metriche del caso di test (test case)

Il protagonista di questa tesi è l'elemento atomico caratterizzante l'intero processo di collaudo di un prodotto software: la singola unità di test. Quest'ultima consiste in una serie di istruzioni di computazione caratterizzate dalla presenza di **asserzioni**. In questo ambito per asserzione s'intende una proposizione logica che restituisce risultato vero o falso, a seguito di un controllo su una porzione del codice, ad esempio per verificarne il corretto funzionamento.

Non è possibile migliorare qualcosa che non si può misurare. Non tutti i test case sono uguali, esistono diverse possibili soluzioni ad uno stesso problema e per determinare la migliore tra queste è necessario fare riferimento a delle metriche per un confronto, come ad esempio:

- **Code coverage**

Data dal rapporto (in percentuale) tra il numero di righe di codice testate ed il numero di codice costituenti l'intero progetto [17], in altre parole maggiore è la code coverage della singola unità e maggiore è il numero di righe di codice effettivamente collaudate dal caso di test.

- **Complessità ciclomatica**

La complessità ciclomatica [10] è una metrica che descrive la complessità di un software, misurando direttamente il numero di cammini linearmente indipendenti attraverso il grafo di controllo di flusso. La validità di tale metrica è stata messa in dubbio dalla comunità scientifica [24].

- **Test pass**

Il rapporto (in percentuale) del numero di test passati, e quindi di asserzioni ad esito positivo all'interno dell'unità di test.

- **Durata del test**

All'aumentare della complessità degli obiettivi raggiunti da un prodotto software, aumenta anche il volume e la complessità del prodotto stesso. In alcune situazioni il costo computazionale del singolo caso di test può risultare un parametro significativo da considerare nel confronto delle unità.

Uno degli aspetti più importanti nel processo di collaudo del codice è la determinazione delle capacità del test suite, della sua abilità nell'individuare difetti ed incongruenze nel progetto software. L'analisi del code coverage risulta essere uno degli approcci più validi oltre che più comunemente adottato nel settore per raggiungere questi obiettivi [11]. Nel corso di questa tesi, si è deciso quindi di analizzare (e di predire) la **code coverage** di uno specifico caso di test input.

1.2 Motivazioni e Obiettivi

Avere un riferimento del code coverage di un test case rappresenta un parametro che lo sviluppatore può considerare per valutare l'efficacia e l'adeguatezza del test appena generato; oltre che ad essere un'ipotetica informazione d'interesse ad un'eventuale software di generazione automatica di casi di test, contribuendo dunque all'automazione del processo di testing.

Essere inoltre in grado di avere un riferimento ad un parametro quale la code coverage in modo **statico**, e quindi senza dover ricorrere ad una computazione del test a runtime, agevola il procedimento dello sviluppo di un prodotto software, in primis portando tutti i benefici di una metrica come il code coverage, senza dover però investire un capitale computazionale potenzialmente oneroso non trascurabile. Per progetti caratterizzati da un grande peso computazionale dovuta ad un'elevata complessità algoritmica, andare a computare la code coverage per ogni singolo case test può voler dire investire un capitale di calcolo costituente ore se non intere giornate.

Infine avere accesso alla code coverage di un test case in modo statico e non a run time, risulta particolarmente utile qualora nel suddetto caso di test risulti il fenomeno del **flakiness** [23] ovvero una situazione ove computando iterativamente uno stesso test, i risultati non siano consistenti ma anzi contrastanti tra loro.

Lo sviluppo di questa tesi è quindi costituito da un'analisi statistica applicata alle diverse metriche o caratteristiche (features) descrittive di un generico caso di test e la variabile dipendente che esprime la code coverage; avendo come obiettivo quello di costruire un modello di regressione in grado di andare a predire nel modo più accurato possibile la code (line) coverage di un generico caso di test dato in input al modello. Essere in grado quindi di poter analizzare un caso di test staticamente senza dover necessariamente compilarlo a runtime.

1.3 Sviluppo e Risultati ottenuti

Partendo da un dataset grezzo contenente diecimila singole unità di casi di test, rappresentati sotto forma di trentasei diverse colonne, una per caratteristica del test unit, sono state apportate procedure di **data engineering** volte a rendere l'informazione contenuta nel dataset, il più appetibile possibile sicché da agevolare l'apprendimento del modello di regressione finale. Una volta applicati metodi statistici di correlazione lineare [5] alle diverse

(feature) dei casi di test presenti nel dataset considerato, è stato possibile determinare quali siano le feature utili alla predizione del line coverage, e quali invece hanno un impatto potenzialmente negativo per il training del modello. Andando a sfoltire quindi le trentasei colonne del dataset di partenza, rimuovendo dall'informazione contenuta nel dataset il rumore che potrebbe incidere negativamente nel training del modello di regressione, tale procedimento prende il nome di **feature analysis**.

A questo punto non resta che individuare la tipologia di modello più attinente allo scenario preso in esame. A tale scopo è stata eseguita una fase di sperimentazione in cui vengono confrontate alcune delle diverse tipologie di modello di regressione. Tale confronto consiste nel costruire i diversi modelli considerati, uno per tipologia. Andare ad effettuare il training dei singoli modelli con il dataset conseguente le due fasi precedenti, e infine confrontare l'accuratezza nella precisione dei modelli addestrati. Si è constatato che la tipologia **Random Tree Forest** [6], per il problema considerato, è la tipologia di modello di regressione più adatta in quanto in grado di predire il line coverage di un generico test case preso in input, con uno scarto assoluto medio di **1,51**. Un valore superiore a quello restituito dai suoi rivali.

1.4 Struttura della tesi

Il precedente capitolo si è impegnato a dare un'infarinatura generale sui concetti di testing e sulla validità di tale disciplina nel mondo dello sviluppo software.

A seguire nel secondo capitolo 'Stato dell'arte' viene invece fornita al lettore un'insieme di nozioni di base appartenenti al mondo del data science e nello specifico del machine learning. Il terzo capitolo 'Metodologia di analisi' descrive il processo di ricerca analitica nel contesto: un'analisi sulle metriche descrittive dei casi di test (ovvero le variabili indipendenti considerate), un approfondimento sulla variabile dipendente da predire, una narrazione sulle metodologie scientifiche statistiche adottate.

In conclusione nel quarto capitolo di questa tesi 'Il modello', viene descritto il procedimento di selezione che ha coinvolto le diverse tipologie di modelli di regressione trattati, quale di questi è stato selezionato come il 'vincitore' e a conclusione un'analisi sull'impatto che le variabili indipendenti hanno nella predizione del modello in questione.

2.1 Modello di regressione

La regressione statistica consiste nello studio della regressione verso la media. La regressione lineare è una tecnica statistica utilizzata per studiare la relazione tra due o più variabili. Da un punto di vista delle formule, la regressione lineare è una funzione matematica basata dall'equazione della retta.

Un modello di regressione lineare è composto da:

- Una sola variabile dipendente (detta anche risposta o Y)
- Una o più variabili indipendenti (dette anche X esplicative o regressori)
- Un coefficiente di regressione per ogni variabile esplicativa più un coefficiente per l'intercetta B
- Un termine di errore (e). Questo perché la relazione tra due variabili non è quasi mai perfettamente riassumibile tramite un'equazione matematica per diverse possibili ragioni: la relazione potrebbe non essere lineare; potrebbero esserci altre variabili (non considerate e/o non osservabili) che influiscono sulla Y ; ci potrebbero essere errori di misurazione delle variabili.

Nello specifico, la variabile dipendente Y è determinata dai valori dell'intercetta (B_0) a cui vengono sommati i valori delle variabili esplicative (le X) moltiplicate per i loro coefficienti (B), più un termine d'errore (e). Con p regressori l'equazione quindi è:

$$Y = \beta_0 + \beta_1 * X_1 + \beta_2 * X_2 + ... + \beta_p * X_p + e$$

2.1.1 Regressione lineare semplice

Qualora si abbia una singola variabile indipendente si tratta di **simple linear regression**. Riguarda punti bidimensionali le cui dimensioni sono una la variabile dipendente e l'altra la variabile indipendente, convenzionalmente associati alle coordinate x e y del piano cartesiano. L'obiettivo è trovare una funzione lineare (una linea retta non verticale) che, nel modo più accurato possibile, cerchi di predire il valore della variabile dipendente in funzione della variabile indipendente.

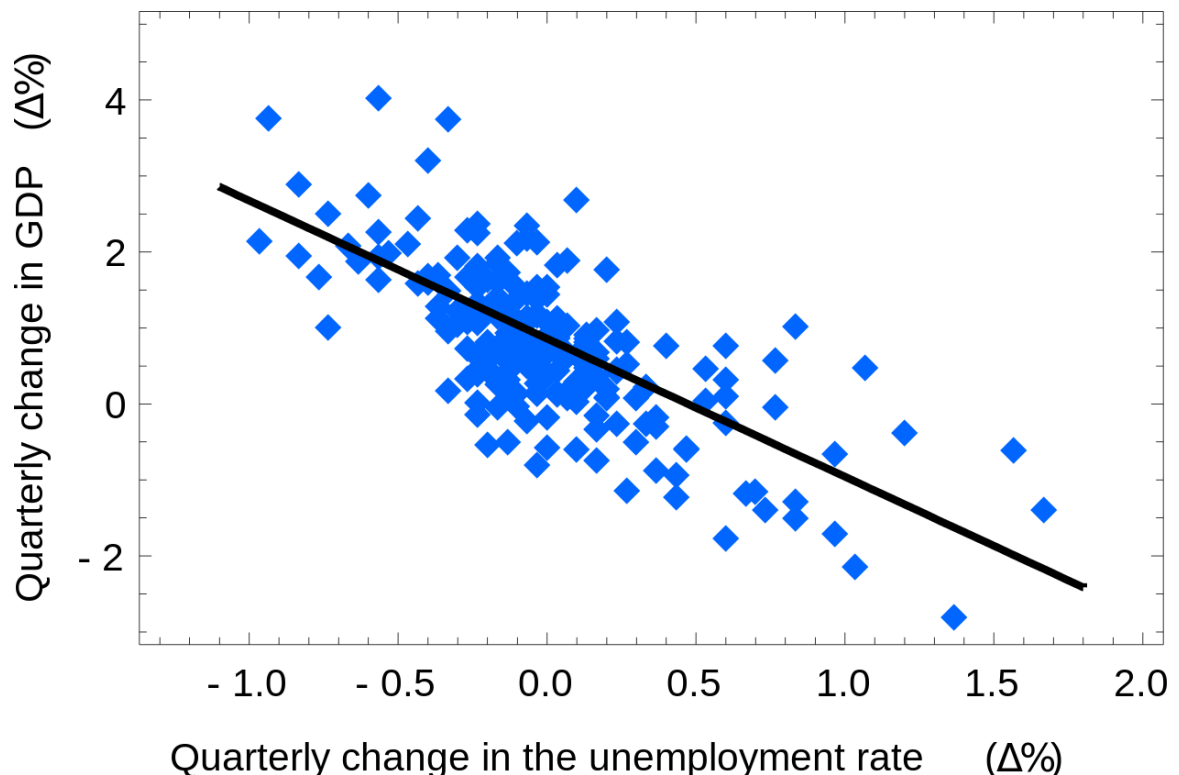


Figura 2.1: Esempio di regressione lineare semplice ove si presume una relazione lineare tra la variabile dipendente (crescita del GDP) e cambiamenti nel tasso di disoccupazione

In genere la regressione lineare semplice prevede il calcolo di proprietà statistiche quali

media, deviazione standard, correlazione e covarianza. Tale tipologia di regressione lineare difficilmente risulta utile nella pratica.

2.1.2 Funzione di costo - Cost function

Prima di continuare con le diverse tipologie di regressione lineare, occorre soffermarsi un attimo sul concetto di 'funzione di costo'. Nel machine learning tale funzione aiuta a determinare lo scarto tra il valore predetto dal modello rispetto al valore effettivo. Esiste poi la **loss function**, utilizzata per quantificare la 'loss' (lo scarto descritto prima) ottenuto nella fase di training. Tali funzioni sono fondamentali per algoritmi che prevedono tecniche di ottimizzazione. Ricapitolando quindi per loss function s'intende l'errore in una singola delle diverse istanze costituenti la fase di training; per cost function si fa riferimento alla media dei coefficienti associati alla loss function sull'intero dataset.

Come già detto prima, una funzione di costo ci consente di determinare quella che è la soluzione ottima poiché offre un criterio presso cui è possibile confrontare diverse soluzioni e determinare la migliore. Ci sono diverse tipologie di funzioni di costo ed è possibile andare a fare una prima classificazione in base alla tipologia del modello su cui si sta lavorando. Per quanto riguarda i modelli di regressione, la funzione di costo prende il nome di **Regression cost function**. Tali funzioni vengono calcolate on the distance-based error con la seguente forma: $\text{Error} = y - y'$ dove y è l'input e y' è il valore predetto in output. Tra le regression cost function più comuni troviamo:

- **Errore medio - Mean Error (ME)** Questa funzione di costo prevede il calcolo dello scarto per ogni istanza del training data, per poi restituire il valore medio.
Gli errori possono assumere valori negativi, ciò implica che andando a fare la media potrebbe annullarsi il valore della funzione di costo, per questo motivo il Mean Error viene più che altro visto come un punto di partenza per le altre tipologie.
- **Errore quadratico medio - Mean Squared Error (MSE)** In questa tipologia di funzione di costo viene applicato il quadrato agli errori, risolvendo il problema di annullamento del risultato della funzione di costo. Una conseguenza dell'applicazione del quadrato ai singoli errori è una maggiore rilevanza attribuita anche alle piccole deviazioni, le quali vengono accentuate. Analogamente però lo stesso discorso regge anche per i cosiddetti 'outliers', i valori anomali considerevolmente più grandi rispetto alla media, assumeranno un'importanza ancora più grande. Per questo motivo si dice che MSE è

più suscettibile agli outliers.

- **Radice dell'errore quadratico medio - Root Mean Squared Error (RMSE)** Il coefficiente RMSE è dato applicando la radice quadrata all'MSE, così facendo si riporta il risultato della funzione di costo alla stessa grandezza della variabile dipendente.

$$MSE = \frac{\sum_{i=0}^n (y - y')^2}{n}$$

Figura 2.2: MSE = (sum of squared errors)/n

- **Errore medio assoluto - Mean Absolute Error (MAE)** Questa funzione di costo risolve il problema di annullamento del valore restituito dal ME, anziché fare il quadrato dei singoli scarti viene invece applicato il modulo. Di conseguenza tale funzione di costo è più prestante dinanzi ai noise e outliers.

$$MAE = \frac{\sum_{i=0}^n |y - y'|}{n}$$

Figura 2.3: MAE = (sum of absolute errors)/n

2.1.3 Ordinary Least Squares

Quando si hanno più di una variabile indipendente input è possibile ricorrere alla seguente tecnica di regressione lineare. Tale procedura consiste nell'andare a minimizzare la somma dei scarti quadratici. In altre parole, data una linea di regressione sui dati, viene calcolata la distanza da ogni singola istanza di dato rispetto alla linea di regressione, calcolarne il quadrato, ripetere il procedimento per tutti le istanze dei dati e calcolarne la somma. Il valore ottenuto è quello che si cerca di minimizzare con la seguente tecnica di regressione. Dal punto di vista geometrico, tale valore corrisponde alla somma quadratica delle distanze, parallele all'asse della variabile dipendente, tra ogni istanza del dataset e il corrispondente punto sulla superficie di regressione, minore è la differenza (distanza), superiore è la capacità predittiva del modello.

2.1.4 Gradient Descent

Per gradient descent s'intende un algoritmo che trova iterativamente la linea di regressione più appropriata da applicare ad un dataset di training; ad ogni iterazione si va a minimizzare il margine d'errore del modello. Inizialmente al modello verranno assegnati per ogni coefficiente dei valori casuali; viene poi calcolata la somma degli scarti quadratici per ogni coppia di valori input/output. Viene selezionato un tasso di crescita (learning rate alpha parameter) che influenzerà la "resistenza" al miglioramento tra un'iterazione e l'altra. Tale processo continuerà finché non si raggiungerà un ottimo o una situazione di stallo, ovvero tra un'iterazione e l'altra non c'è alcun miglioramento.

2.1.5 regolarizzazione

La regolarizzazione è una tecnica nel machine learning che mira a generalizzare il modello, ciò implica che il modello sia performante non solo in fase di training o di testing, ma anche con un generico input che riceverà in futuro. Fondamentalmente la regolarizzazione consiste nel minimizzare the sum of the squared error del modello sul training data (applicando la tecnica dell'Ordinary Least Squares) e al tempo stesso ridurre la complessità del modello.

2.2 Compromesso tra Bias e Varianza

Per **bias** s'intende la differenza tra l'average della predizione del modello e il valore effettivo che stiamo cercando di predire. Lo si può considerare quindi come lo scarto tra il valore predetto e la realtà.

Un modello con alto bias da poco conto alle informazioni presenti nel dataset utilizzato in fase di training, il modello è troppo semplice per poter riuscire ad analizzare sufficientemente la realtà, non è in grado di considerare le variazioni, si tratta di una situazione di **underfitting**. Nel caso della **varianza** siamo di fronte ad una situazione opposta a quella appena descritta nel caso del bias; la varianza descrive la capacità del modello di considerare ed apprendere nel complesso tutte le informazioni contenute nel dataset, includendo informazioni 'scomode' come il rumore e gli outliers. Ne consegue che il modello apprende troppo dal training data a tal punto che, in fase di testing non sarà in grado di offrire predizioni accurate; ci troviamo quindi in una situazione di **overfitting**.

Nell'apprendimento supervisionato, ci si potrebbe imbattere in una situazione di underfitting quando il modello non riesce a concettualizzare i pattern intrinseci nel data. Questo genere

di modelli avranno un alto bias ed una bassa varianza. In genere ciò accade quando non si ha un quantitativo sufficiente di data a disposizione per il training oppure ad esempio quando si sta cercando di costruire un modello di regressione lineare utilizzando però data non lineare. Una situazione di overfitting è frutto di un dataset rumoroso che confonde l'apprendimento del modello facendogli prendere in considerazione rumore o pattern fuorvianti; in genere questi modelli hanno un basso bias ed un'alta varianza.

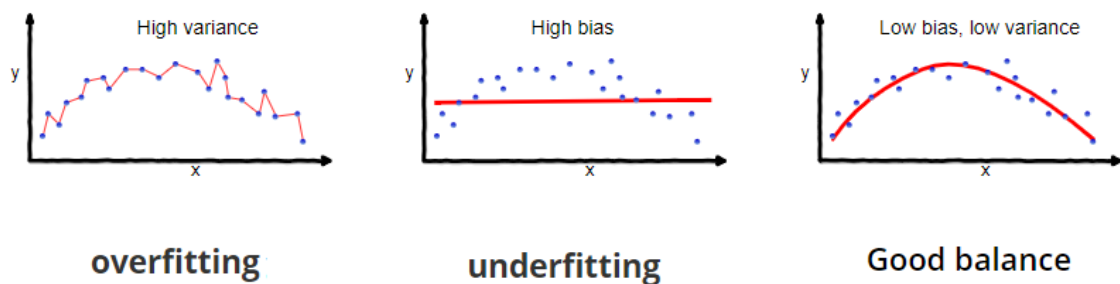


Figura 2.4

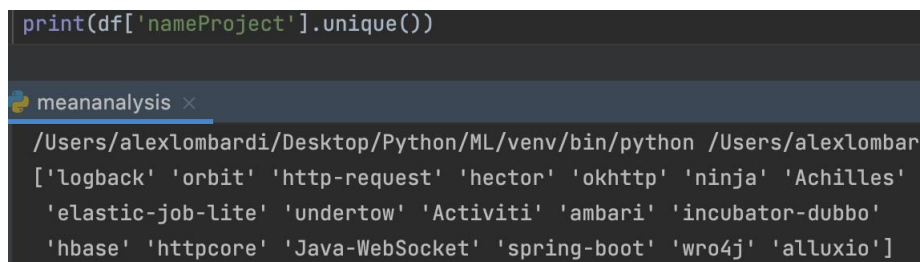
Il bias e la varianza sono due concetti uno l'opposto dell'altro; se il modello è troppo semplice e con pochi parametri, c'è il rischio che s'incomba in una situazione ad alto bias e bassa varianza; viceversa se il modello ha un alto numero di parametri potremmo avere un'alta varianza ed un basso bias. Occorre trovare un equilibrio prevenendo sia l'overfitting che l'underfitting sui dati. Questo compromesso nella complessità è il cosiddetto tradeoff between bias and variance: un algoritmo non può essere più complesso e meno complesso allo stesso tempo.

Il dataset descritto nel seguente capitolo è già stato protagonista di altri articoli scientifici; ad esempio uno studio sulla flakiness dei test cases, in altre parole un'analisi sulla non deterministicità riguardante l'esito dei suddetti test cases [23].

Considerando la struttura del suddetto dataset, è possibile svolgere un'analisi sulle diverse caratteristiche di codifica del singolo caso di test, permettendo quindi un'analisi statistica sulla natura di tali caratteristiche o features, e sul loro impatto a proposito della coverage conseguente.

Costituito da un numero complessivo di quasi 10000 singole unità di test, vengono coinvolti 18 diversi progetti software, la distribuzione dei casi di test rispetto ai diversi progetti è descritta dai seguenti grafici:

```
print(df['nameProject'].unique())
```



```
['logback' 'orbit' 'http-request' 'hector' 'okhttp' 'ninja' 'Achilles'  
'elastic-job-lite' 'undertow' 'Activiti' 'ambari' 'incubator-dubbo'  
'hbase' 'httpcore' 'Java-WebSocket' 'spring-boot' 'wro4j' 'alluxio']
```

Figura 3.1: i nomi dei progetti

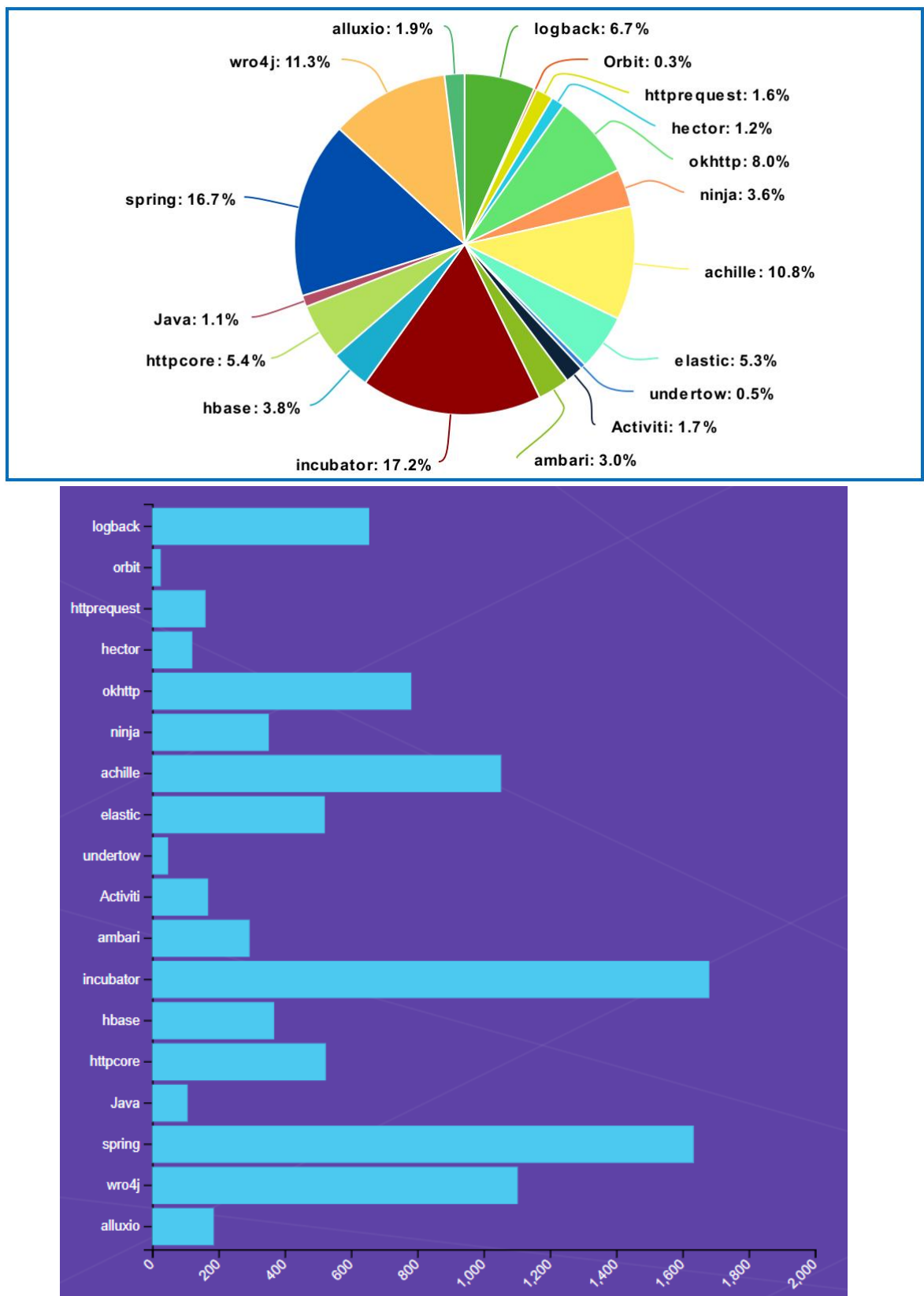


Figura 3.2: distribuzione dei test sui diversi progetti

3.1 Descrizione delle features

Metriche o features dei casi di test	
Nome	Descrizione
TLOC	Numero di linee di codice
TmcCabe	Indice sulla complessità del codice adottando il metodo della complessità ciclomatica
Lcom2	Indice sulla coerenza nei metodi del test case v2.
Lcom5	Indice sulla coerenza nei metodi del test case v5
CBO	Coupling tra gli oggetti, il numero di dipendenze che una classe ha con le altre [7]
WMC	La somma della complessità di tutti i metodi del test case
RFC	Il numero di metodi che potenzialmente potrebbero essere invocati da altre classi [7]
MPC	Numero di messaggi scambiati tra gli oggetti della classe
Halstead Vocabulary	Somma complessiva dei distinti operatori e operandi in una funzione.
Halstead Lenght	Somma complessiva delle occorrenze di operatori e operandi.
Halstead Volume	Spazio richiesto in bits per la memorizzazione del programma.
numCoveredLines	Numero di linee di codice ricoperte dal test
executionTime	Tempo di esecuzione del test
projectSourceLinesCovered	Numero di classi di produzione ricoperte dal test
hIndexModPerCoverLine _{tw} X	con X un valore tra [5, 10, 25, 50, 75, 100, 500, 10,000], almeno X linee di codice sono state modificate X volte nel caso di test
num third party libs	librerie di terze parti impiegate nel test

Code smells	
Nome	Descrizione
classDataShouldBePrivate	Quando una classe espone i suoi attributi, violando i principi di information hiding.
complexClass	Quando una classe ha un indice di complessità elevato.
functionalDecomposition	Quando in una classe tecniche di polimorfismo ed ereditarietà sono utilizzati impropriamente
godClass	Quando una classe ha una grande dimensione e implementa diverse responsabilità.
spaghettiCode	Quando una classe non ha una struttura e dichiara metodi prolissi senza parametri.

Text smells	
Nome	Descrizione
Assertion density	percentuale di asserzioni nel codice del test case
Assertion roulette	asserzioni non documentate nel codice del test case
Mystery Guest	Il test presenta materiale esterno [8]
Eager test	Il test analizza più metodi contemporaneamente [8]
Sensitive equality	Il test presenta un confronto sul toString [8]
Resource Optimism	Il test fa uso di risorse esterne potenzialmente non disponibili[8]
Conditional test logic	Il test prevede un if statement condizionale
Fire and forget	Il test lancia attività secondarie in background

3.2 Analisi delle features

L'obiettivo è quello di andare a costruire un modello tale che, preso in input un test case, rappresentato sotto forma di variabili decimali, un sott'insieme delle feature statiche appena descritte, tale modello cercherà di predire la coverage conseguita dal test in questione. Nello sviluppo di un modello di machine learning, eventualmente si arriverà nella fase di **training**. Nella suddetta fase al modello verranno 'date in pasto delle informazioni' sicché esso possa comprendere indirettamente gli aspetti e le tematiche costituenti il problema trattato. Nel nostro caso le informazioni sono tutto ciò che è presente all'interno del dataset. Si potrebbe quindi supporre che all'aumentare del quantitativo di informazioni fornite al modello in fase di training, aumenterà di conseguenza anche la comprensione del modello dello scenario e quindi anche le sue performance. Tuttavia questa ipotesi è errata in quanto si, fino ad un certo limite, aumentando la profondità dell'informazione ne risentirà in positivo l'addestramento del modello. Raggiunto questo limite però si incomberà in diverse problematiche spiacevoli, prima fra tutti l'**overfitting** [20]: situazione in cui un modello si adatta troppo bene ai dati di training e, di conseguenza, non riuscirà a predire in modo accurato i dati di test non visualizzati; in altre parole il modello ha un riferimento 'teorico' troppo forte e non riesce ad affrontare una situazione che non ha già visto nel training. Una problematica conseguente ad un training troppo profondo è l'incremento della complessità del modello: per quanto possibile, è nel nostro interesse che il modello sia semplice e comprensibile, pena l'inutilizzabilità del risultato finale.

Per queste motivazioni occorre andare a fare un'analisi delle 38 feature presenti nel dataset, per poi selezionarne un sott'insieme: quelle più adatte al training del modello, in altre parole quelle feature che riescono a descrivere il problema in maniera più efficiente rispetto alle altre [15].

3.3 Variabile dipendente

La variabile dipendente ha un ruolo cruciale in quanto esprime in se stessa l'informazione che si sta cercando di far predire al modello. Considerando il paragrafo precedente in cui vengono elencate le diverse features del dataset, si può notare che il ruolo della variabile dipendente ha due possibili candidati: "numCoveredLines" e "projectSourceLinesCovered"

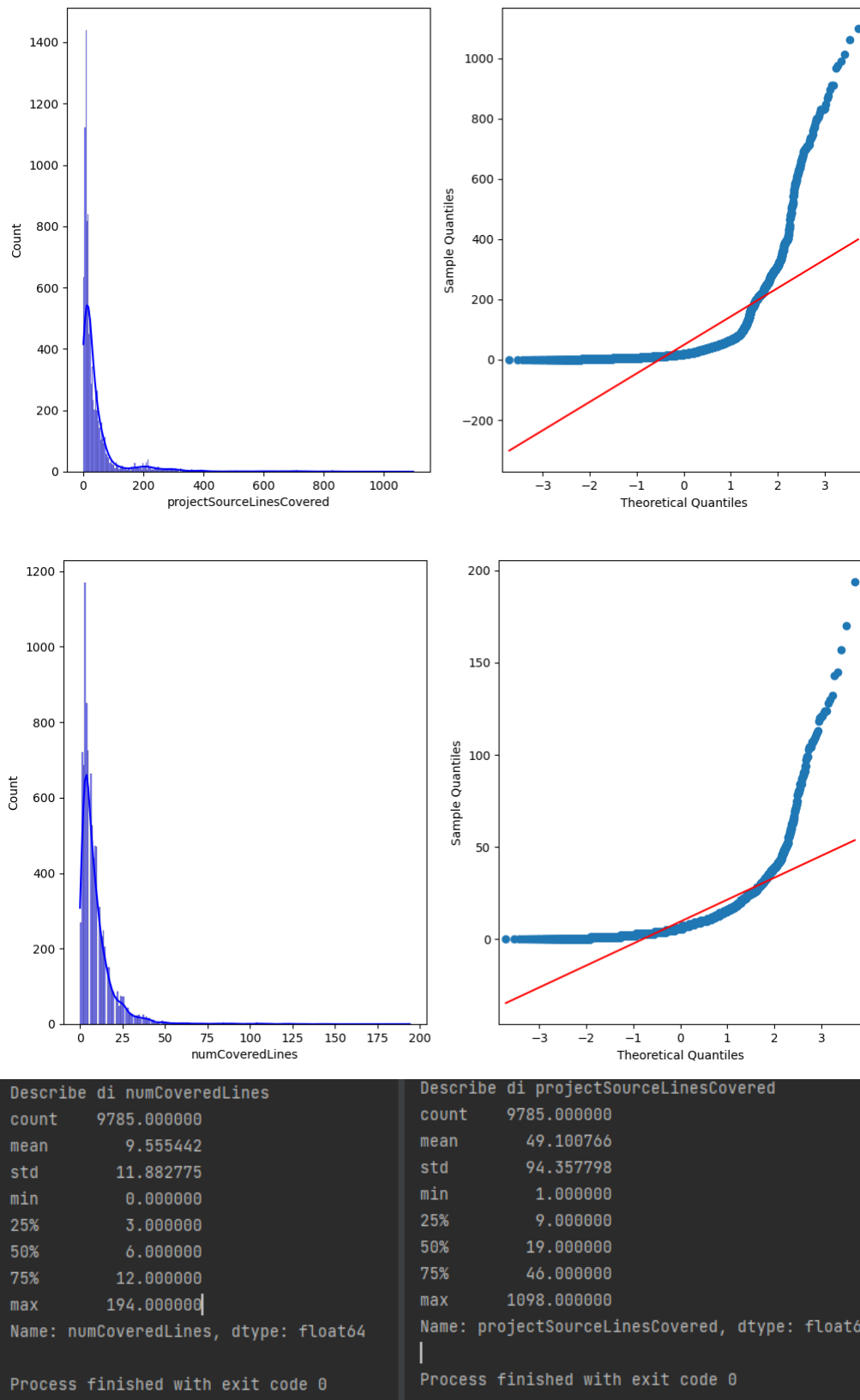


Figura 3.3: probability plot e descrizione di "projectSourceLinesCovered" e "numCoveredLines"

Considerando le Figure 3.3 e 3.4 è possibile analizzare la distribuzione dei valori di queste due feature. Per cominciare la prima informazione che si può dedurre dai due grafici è che la stragrande maggioranza dei casi di test tendono a raggiungere i medesimi valori, sono però presenti diversi outliers, delle istanze anomale che si discostano dal valore medio, ad esempio per quanto riguarda la variabile "projectSourceLinesCovered" alcune istanze raggiungono valori come 1098 che si discosta molto dal valore medio di 48. Occorre quindi andare a fare una pulizia di questi outliers che potrebbero ingannare il modello durante il suo apprendimento.

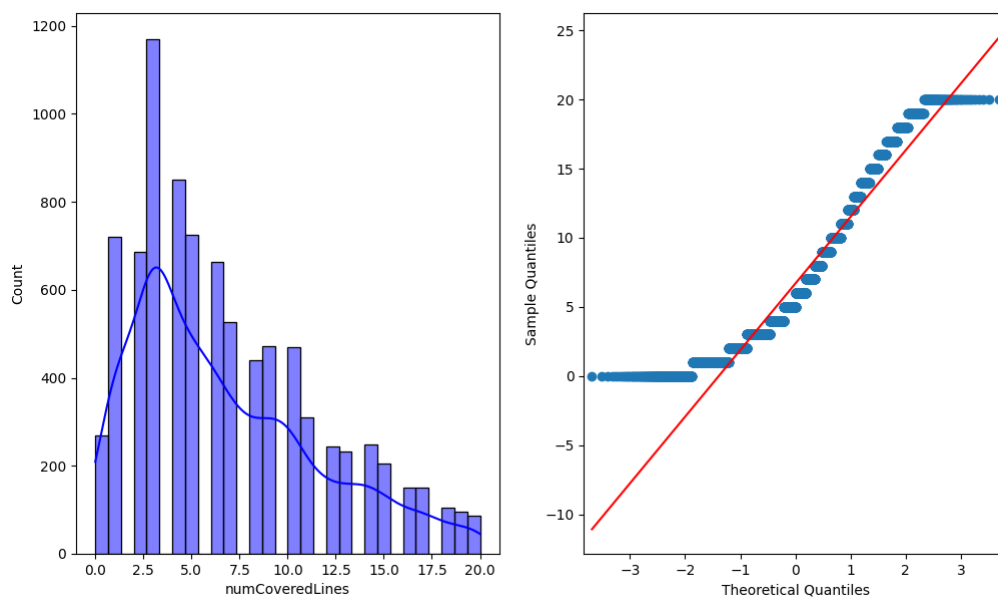


Figura 3.4: probability plot di "numCoveredLines" dopo aver rimosso gli outliers

durante lo sviluppo del modello verrà presa in considerazione la feature "numcovered-lines" per una semplice questione di attinenza diretta allo scopo di questa tesi: riuscire a predire il numero di righe di codice coperte da un singolo caso di test.

3.4 Violin plot

In questa sezione verrà descritta l'applicazione di uno strumento che consente di andare a descrivere le differenze di una data feature, a seconda se si tratta di un'istanza positiva o negativa. In questo caso per positivo intendiamo quelle istanze di test che ricoprono un quantitativo di linee di codice superiore al valore medio, negativo il contrario. Sotto il punto di vista teorico, una feature con una maggiore discrepanza tra i casi positivi e negativi, riuscirà a descrivere meglio al modello le caratteristiche di un test case positivo e negativo.

Quello che si sta cercando quindi sono tutte quelle feature sicché i due possibili stati della variabile dipendente (positivo o negativo), in riferimento alla feature in analisi, assuma valori il più diverso possibili per stato, una feature tale che il suo violin plot corrispondente abbia le righe orizzontali centrali rappresentanti la mediana, il più lontane possibili sull'asse delle ordinate. Un buon esempio è quello che troviamo nella Figura 3.5 ovvero la violin plot della feature tloc. Dal grafo è possibile notare che la distribuzione dei valori raggiunti dalle singole istanze presenta uno scarto significativo nei due possibili stati della variabile dipendente, questo è un indizio a proposito della capacità di questa feature a scindere e a trovare una disgiunzione tra i test soddisfacenti da quelli non soddisfacenti.

Quello che invece non si sta cercando è una situazione in cui le due mediane quasi si incontrano, sintomo che secondo tale feature, i test positivi sono indistinguibili da quelli negativi, si tratta quindi di una feature potenzialmente pessima da dare in training al nostro modello.

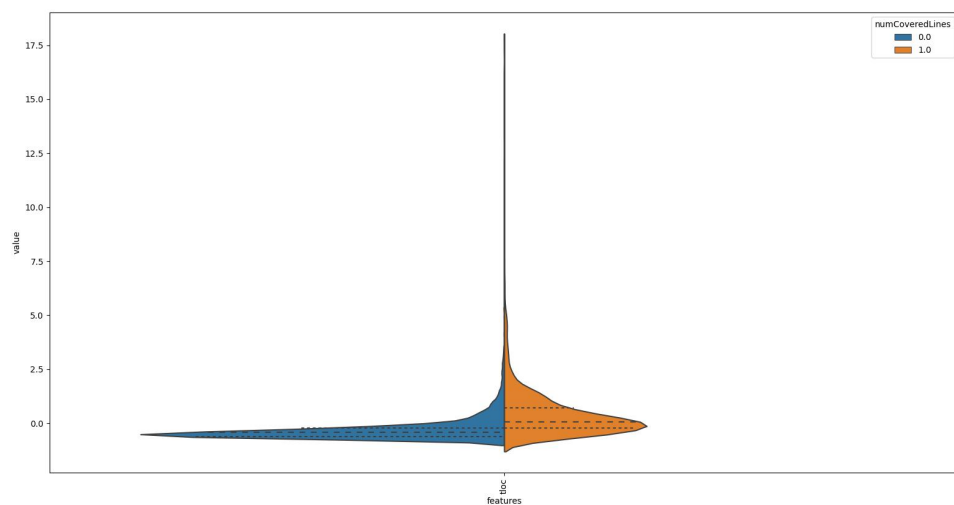


Figura 3.5: violin plot della feature 'tloc'

3.5 Indice di correlazione di Pearson

$$r = \frac{\sum (x - \bar{x})(y - \bar{y})}{\sqrt{\sum (x - \bar{x})^2 \sum (y - \bar{y})^2}}$$

Figura 3.6: formula del coefficiente di correlazione di Pearson

Il coefficiente di correlazione lineare di Pearson è tra gli strumenti più rilevanti per determinare la correlazione lineare tra due variabili [5]. Il risultato di tale metodo restituisce un valore compreso tra -1 e 1 ove

- **-1 indica una correlazione negativa massima**
- **0 indica un'assenza di alcuna correlazione**
- **1 indica una correlazione positiva massima**

Una correlazione positiva implica che all'aumentare di X con X la variabile dipendente 'numCoveredLines', aumenta anche Y con Y la variabile indipendente (i.e una ipotetica metrica del caso di test), una negativa invece indica che all'incrementare di X, Y invece decresce.

Date due variabili statistiche X e Y, l'indice di correlazione di Pearson è definito come la loro covarianza divisa per il prodotto delle deviazioni standard delle due variabili

- **H0 - ipotesi nulla:** la variabile dipendente non è linearmente correlata alla variabile dipendente
- **H1 - ipotesi alternativa:** la variabile dipendente è linearmente correlata alla variabile dipendente

Il test consiste nel calcolo della correlazione lineare tra le due variabili. Supponendo l'ipotesi nulla vera, quindi che la variabile dipendente non sia linearmente correlata alla variabile indipendente, il valore del coefficiente di correlazione tra i due operandi deve tendere allo zero. **Più l'indice di correlazione si avvicina ad 1, meno è probabile la validità dell'ipotesi nulla.**

Quello che si sta cercando è una feature (variabile indipendente) del dataset che sia il più correlata possibile alla variabile dipendente 'numCoveredLines': **tale correlazione può essere utilizzata dal modello come indizio per la predizione del valore della variabile dipendente che si sta cercando.** Un'ipotesi nulla indica quindi la presenza di una variabile indipendente utile all'apprendimento del modello.

Per determinare la veridicità dell'ipotesi nulla, si fa un confronto tra il coefficiente di correlazione lineare ed il **p-value** [25].

Nella statistica inferenziale il **Valore P** o anche livello di significatività osservato è la probabilità, per una ipotesi supposta vera (ipotesi nulla), di ottenere risultati ugualmente o meno compatibili, di quelli osservati durante il test, con la suddetta ipotesi.

In altri termini, il valore p aiuta a capire se la differenza tra il risultato osservato e quello ipotizzato è dovuta alla casualità introdotta dal campionamento, oppure se tale differenza è statisticamente significativa, cioè difficilmente spiegabile mediante la casualità dovuta al campionamento.

In sintesi si stanno cercando tutte quelle variabili indipendenti aventi una forte correlazione con la variabile dipendente selezionata.

Il test è stato elaborato in Python mediante l'utilizzo della libreria `scipy.stats` [1]

Nello snippet di codice che segue viene caricato il dataset, vengono rimosse le variabili indesiderate, viene quindi calcolato e poi stampato il risultato dell'applicazione del Pearson a tutte le coppie x, y tale che almeno uno tra x e y siano la variabile dipendente

```
import pandas as pd
from scipy.stats import pearsonr, spearmanr

data = pd.read_csv("csvume/dataset.csv")
print(len(data.columns))
list = ['nameProject', 'testCase', "Unnamed:_0", "projectSourceLinesCovered", ]
data = data.drop(list, axis = 1 )
print(data.columns)
data2 = data.numCoveredLines

for i in range(1, 36):
    data1 = data.iloc[:, i]
    stat, p = pearsonr(data1, data2)
    print("Variabile:_ " + data.columns[i])
    print('stat={0:.3f}, p={0:.3f}'.format(stat, p))
    if p > 0.05:
        print('Probably_independent\n')
    else:
        print('Probably_dependent\n')
```

3.5.1 Risultati

Variabile	coeff. di correlazione	esito probabilistico
tloc	0.543	dependent
tmcCabe	0.190	dependent
assertionDensity	0.189	dependent
assertionRoulette	0.359	dependent
mysteryGuest	0.113	dependent
eagerTest	0.091	dependent
sensitiveEquality	0.058	dependent
resourceOptimism	-0.004	Independent
conditionalTestLogic	0.196	dependent
fireAndForget	0.2	dependent
loc	0.051	dependent
lcom2	0.005	Independent
lcom5	0.002	Independent
cbo	0.092	dependent
wmc	0.063	dependent
rfc	0.082	dependent
mpc	0.088	dependent
halsteadVocabulary	0.088	dependent
halsteadLength	0.058	dependent

Variabile	coeff. di correlazione	esito probabilistico
halsteadVolume	0.057	dependent
classDataShouldBePriv	-0.025	dependent
complexClass	0.022	dependent
functionalDecomp	-0.039	dependent
godClass	0.024	dependent
spaghettiCode	0.018	Independent
ExecutionTime	0.089	dependent
hIndexMod..dLine5	0.277	dependent
hIndexMod..dLine10	0.233	dependent
hIndexMod..dLine25	0.200	dependent
hIndexMod..dLine50	0.158	dependent
hIndexMod..dLine75	0.128	dependent
hIndexMod..dLine100	0.145	dependent
hIndexMod..dLine500	0.351	dependent
hIndexMod..dLine10000	0.296	dependent

I risultati descrivono come la variabile 'floc' sia la variabile presente nel dataset avente la correlazione più forte rispetto alle altre, risultato coerente con quanto constatato coi violin plot. Lo stesso non si può dire per le altre variabili individuate nella fase precedente (CBO, LOC, RFC, etc.), tali feature pur superando la soglia minima del p value del 0.05 non si avvicinano ai livelli di correlazione di tloc. A seguire ci sono altre variabili interessanti come 'assertionRoulette' e 'hIndexMod..dLine500'. Viceversa variabili come 'lcom2' e 'lcom5' risultano non presentare una correlazione significativa rispetto al numero di righe di codice ricoperte dall'istanza del test case. In conclusione è possibile indurre che all'aumentare del numero di righe di codice da cui è composto uno specifico test case, aumenterà anche il quantitativo di righe di codice ricoperte dal test case in questione.

In questo capitolo è descritta una collezione di risultati ottenuti dall'elaborazione in predizione di distinte tipologie di modelli. Un generico modello prenderà in input un insieme di metriche descrittive di un caso di test (feature indipendenti) e restituirà in output una predizione **indotta** sul numero di righe di codice coperte dal suddetto test case.

Nell'algoritmo che segue, in un primo momento vengono rimosse dal dataset le feature non ritenute non adeguate ai fini dell'apprendimento di un modello (ad esempio l'id o il nome del progetto) così come le feature ritenute 'troppo interessanti' quali `projectSourceLinesCovered`; con il metodo `train-test-split` della libreria `sklearn` si il dataset viene partizionato in due sott'insiemi uno per il training ed uno per il testing.

La spartizione di 'df' la struttura dati contenente il dataset in quattro elementi `x_train` `y_train`, `x_test`, `y_test` impiegati nell'addestramento e testing dei modelli. Una composizione casuale di occorrenze del dataset pari al 70% sarà impiegata per l'addestramento, il restante 30% per il test.

```

df = pd.read_csv("csvume/dataset.csv")
#df = df[df['nameProject'].str.match('logback')]
list = ['nameProject', 'testCase', "Unnamed:_0",
        "projectSourceLinesCovered", "numCoveredLines"]
y = df.numCoveredLines
df = df.drop(list, axis = 1 )
# print(df.columns)
# split data train 70 % and test 30 %
x_train, x_test, y_train, y_test = train_test_split(df, y, test_size=0.3,
                                                    random_state=42)
y_train = np.log1p(y_train)

```

4.1 Data Engineering sulla variabile dipendente

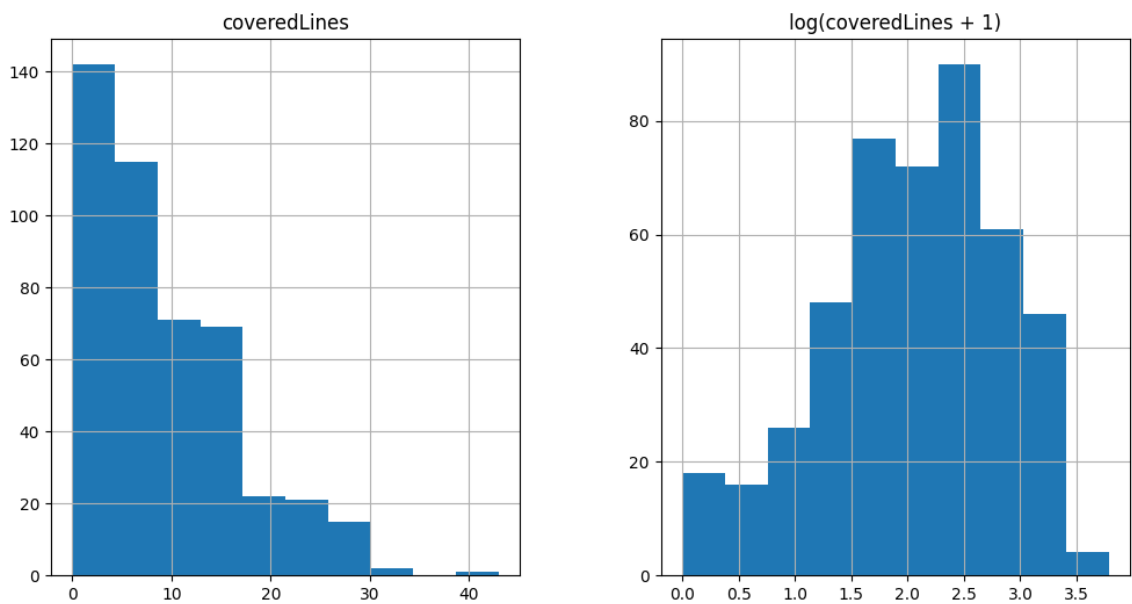


Figura 4.1: distribuzione delle istanze dei test case sul coefficiente associato alla variabile dipendente.

Pur rimuovendo gli outliers, resta comune una forte skewness a sinistra nella distribuzione delle occorrenze. Circa due terzi delle istanze totali appartengono al range 0-10 (figura a sinistra). Nella figura di destra è descritto il medesimo scenario dopo l'applicazione del logaritmo alla variabile dipendente, il risultato è una distribuzione normale [2]

Andando ad analizzare la distribuzione della nostra variabile dipendente è evidente come la maggioranza dei test case presenti nel dataset abbiano una tendenza ad assumere un valore

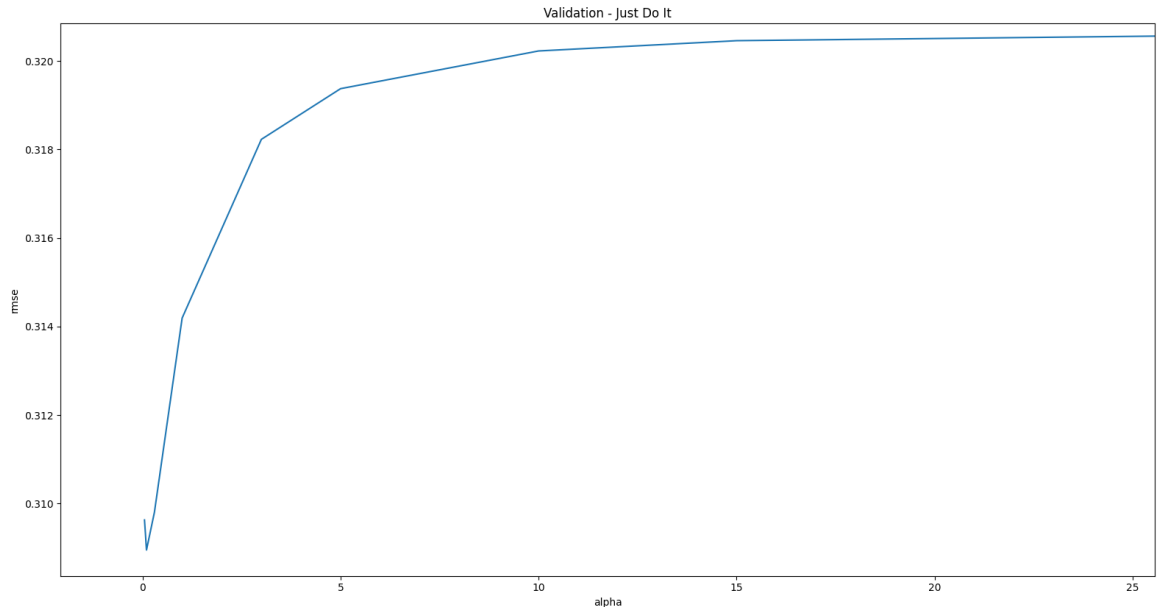
nel range 0-10. il che implica una distribuzione non lineare dei valori raggiunti dalla variabile dipendente nel dataset. Al fine di agevolare l'apprendimento del nostro modello, c'è interesse nel rendere il comportamento della variabile dipendente più lineare, è possibile fare questo applicando banalmente il logaritmo all'intera colonna `coveredLines` (la nostra variabile dipendente) la quale assumerà valori distribuiti in maniera più equilibrata verso il valore medio.

4.2 Strategia nelle selezione del modello

Sono stati implementati quattro diversi modelli di regressione. I primi tre modelli sono strutturalmente identici essendo tutti modelli di regressione lineare, ciò che li differenzia è la tecnica di regolarizzazione adottata che caratterizzerà il loro metodo di apprendimento e da cui prendono il nome, nello specifico: 'lasso', 'ridge' e XGB. L'ultimo modello invece è della tipologia 'Random Forest Regressor', si tratta di una tipologia fortemente diversa dalle altre pertanto la sua struttura verrà approfondita in seguito. E' inoltre descritta la strategia adottata per l'iperparametrizzazione dei diversi modelli. Infine i quattro modelli saranno confrontati sul coefficiente del RMSE o 'Radice dell'errore quadratico medio'

4.3 Ridge model

Il primo modello adotterà il metodo di regolarizzazione noto come Ridge [13]. Tale metodo di regolarizzazione ha la peculiarità di ricorrere ad un parametro di configurazione definito α che inciderà sulla flessibilità del modello. Maggiore sarà la regolarizzazione minore sarà la probabilità di andare in overfit, ciononostante il modello perderà di flessibilità incombendo nel rischio di non riuscire a captare tutte le informazioni presenti nei dati. L'unico iperparametro da configurare è quindi questo coefficiente α , il valore ottimale è stato determinato testando diversi valori [0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 50, 75] e confrontandone i risultati Figura 4.3 .



Dal grafico si evince che il coefficiente ideale da assegnare ad α è poco superiore allo 0, superato tale valore la radice dell'errore quadratico medio non farà altro che aumentare. Trovato il coefficiente α , una volta effettuato il training sul modello ed eseguite le predizioni sulle istanze del dataset dedicate al testing, si avrà un RMSE del **0.6272**. In altre parole avremo uno scarto assoluto medio (MAE) di 6.15 linee di codice di differenza tra il valore predetto e quello effettivo; analizzando le predizioni però si evince che lo scarto tra valore predetto ed effettivo è direttamente proporzionale alla grandezza del valore reale. Andando a scartare tutti i test le cui linee di codice corrispondenti superino 20 loc, lo scarto tra valore predetto e reale sarà di 3.22.

4.4 Lasso model

Il secondo modello adotterà la tecnica di regressione nota come Lasso. Questa tipologia di regressione ha la peculiarità di andare automaticamente a determinare quali sono le feature interessanti o meno ai fini dell'addestramento del modello. Sebbene sia una tecnica che potrebbe offrire spunti interessanti ai fini di questa tesi, non si andrà troppo nel dettaglio in questo paragrafo sicché il valore dell'RMSE è pari a **0.6279** leggermente superiore a quello del modello precedente e anche qui lo scarto assoluto medio tra il valore predetto ed effettivo è di 6.15. In conclusione si evince che tra Ridge e Lasso come tecniche di regolarizzazione il risultato è funzionalmente identico.

4.5 Extreme Gradient Boosting (XGBoost)

Il terzo modello è stato costruito ricorrendo alla libreria open source XGBoost adatta allo sviluppo di modelli di regressione supervisionati. Tale modello adotta la tecnica dell'**ensemble**: nella fase di apprendimento saranno combinati più modelli individuali (base learners) ognuno di essi farà la sua predizione. Esistono diverse tipologie di apprendimento ensemble, in questo caso verrà adottata la tecnica del **boosting** ovvero ciasun base learner influirà sulla predizione finale con un certo peso. Il coefficiente del peso viene calcolato in base all'errore commesso in fase di learning.

Una caratteristica è il formato del dataset utilizzato dal modello ovvero il DMatrix. Una tipologia di struttura dati che va ad incrementare le performance e l'efficienza.

Un modello XG durante la fase di training analizza la complessità di ogni base learner, l'obiettivo è quello di ottenere come risultato un modello semplice ed accurato, qualora tramite una loss function uno dei diversi base learner divenisse troppo complicato, per prevenire il rischio di overfitting vengono combinate tecniche di regolarizzazione quali LASSO e Ridge. Il terzo modello offre un leggero miglioramento se confrontato con i suoi predecessori, riesce a predire la variabile dipendente con un RMSE del 0.4825 e quindi uno scarto medio assoluto di 3.45 linee di codice oppure 2.33 considerando solo i casi dal LOC inferiore a 25

```
df = pd.read_csv("csvume/dataset.csv")
list = ['nameProject', 'testCase', "Unnamed:_0",
        "projectSourceLinesCovered", "numCoveredLines"]
y = df.numCoveredLines
df = df.drop(list, axis = 1)
x_train, x_test, y_train, y_test = \
    train_test_split(df, y, test_size=0.3, random_state=42)
y_train = np.log1p(y_train)
from sklearn.model_selection import cross_val_score
def rmse_cv(model):
    rmse= np.sqrt(-cross_val_score(model, x_train,
                                    y_train, scoring="neg_mean_absolute_error", cv = 5))
    return (rmse)
```

#XGB

```
import xgboost as xgb
dtrain = xgb.DMatrix(x_train, label = y_train)
params = {"max_depth":2, "eta":0.1}
model = xgb.cv(params, dtrain,
               num_boost_round=500, early_stopping_rounds=100)
model_xgb = xgb.XGBRegressor(n_estimators=360,
                             max_depth=2, learning_rate=0.1) #the params were tuned using xgb.cv
print(rmse_cv(model_xgb).mean())
model_xgb.fit(x_train, y_train)
xgb_preds = np.expm1(model_xgb.predict(x_test))
predizioni = pd.DataFrame({"id":x_test.id, "coveredLines":xgb_preds})
predizioni.to_csv("csvume/xgb.csv", index = False)
```

#RIDGE

```
from sklearn.linear_model import Ridge, RidgeCV, ElasticNet, LassoCV, LassoL
model_ridge = Ridge()
alphas = [0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 50, 75]
cv_ridge = [rmse_cv(Ridge(alpha = alpha)).mean()
            for alpha in alphas]
cv_ridge = pd.Series(cv_ridge, index = alphas)
cv_ridge.plot(title = "Validation_Just_Do_It")
plt.xlabel("alpha")
plt.ylabel("rmse")
print("Ridge")
print(cv_ridge.min())
model_ridge = Ridge(alpha = 0.05).fit(x_train, y_train)
ridge_preds = np.expm1(model_ridge.predict(x_test))
predizioni = pd.DataFrame({"id":x_test.id, "coveredLines":ridge_preds})
predizioni.to_csv("csvume/ridge.csv", index = False)
```

#LASSO

```
model_lasso =\
    LassoCV(alphas = [1, 0.1, 0.001, 0.0005]).fit(x_train, y_train)
```

```
print("Lasso")
print(rmse_cv(model_lasso).mean())
coef = pd.Series(model_lasso.coef_, index = x_train.columns)
lasso_preds = np.expm1(model_lasso.predict(x_test))
predizioni = pd.DataFrame({"id": x_test.id, "coveredLines": lasso_preds})
predizioni.to_csv("csvume/lasso.csv", index = False)
```

4.6 Random Forest Regression

L'ultimo modello è il più preciso in quanto è in grado di predire il numero delle covered lines con uno scarto assoluto medio di 2.31 linee di codice. E' stato usato il modulo sklearn [21] per lo sviluppo e il training del random forest regression model, nello specifico la funzione RandomForestRegressor.

La 'foreste casuale' è una tipologia di apprendimento **ensemble** adottabile sia dai modelli di regressione che di classificazione, il training consiste nella costruzione di molteplici alberi decisionali durante l'apprendimento. Nel machine learning un albero di decisione è un modello predittivo, dove ogni nodo interno rappresenta una variabile, un arco verso un nodo figlio rappresenta un possibile valore per quella proprietà e una foglia il valore predetto per la variabile obiettivo a partire dai valori delle altre proprietà, che nell'albero è rappresentato dal cammino (path) dal nodo radice (root) al nodo foglia. In questo caso quindi le foreste casuali sono un insieme di molti alberi decisionali individuali.

Nell'esempio rappresentato dalla Figura 4.5, partendo dal nodo in testa, analizzando il valore della variabile Var_1 la singola istanza decisionale può prendere due possibili percorsi alternativi sulla base di una condizione specifica. Se tale condizione è verificata viene intrapreso il percorso affine e viceversa. Tale procedimento viene ripetuto finché si arriverà ad un nodo foglia contenente il valore predetto.

Per addestrare ogni singolo albero, viene selezionato un campione casuale dell'intero set di dati, questo procedimento è chiamato campionamento con sostituzione o bootstrap. L'algoritmo di bootstrapping Random Forest combina la tecnica d'apprendimento ensemble con il framework degli alberi decisionali, ottenendo quindi alberi considerando partizioni randomiche del dataset.

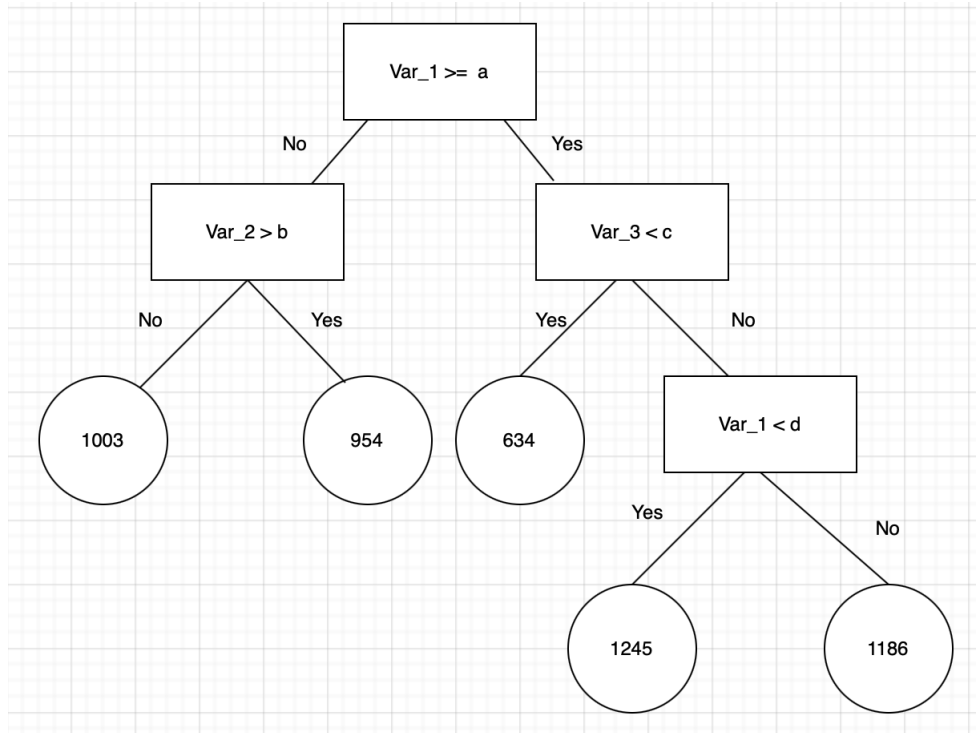


Figura 4.2: esempio di albero decisionale

```

df = pd.read_csv("csvume/dataset.csv")
list = ['nameProject', 'testCase', "Unnamed:_0", "projectSourceLinesCovered"]
df = df.drop(df[df.numCoveredLines > 18].index)
y = df.numCoveredLines
df = df.drop(list, axis = 1 )
# split data train 70 % and test 30 %
x_train, x_test, y_train, y_test =
    \train_test_split(df, y, test_size=0.3, random_state=42)
rf = RandomForestRegressor(n_estimators = 400, min_samples_split=2,
    \min_samples_leaf=1, max_features='sqrt', max_depth = None,
    \,bootstrap=False)
rf.fit(x_train, y_train)
prediction = rf.predict(x_test)
mae = mean_absolute_error(y_test, prediction)
print(mae)

```

Come risultato un MAE del 2.31 è frutto del training eseguito su tutto il dataset per intero, andando a partizionare il dataset e a considerare i singoli progetti che lo compongono si otterranno i risultati descritti nella tabella che segue.

Progetto	Scarto assoluto medio	Percentuale del DS
logback	2.882	6.7%
orbit	4.152	0.3%
http-request	1.167	1.6%
hector	4.303	1.2%
okhttp	1.593	8.0%
ninja	1.117	3.6%
Achilles	1.220	10.8%
elastic-job-lite	1.376	5.3%
undertow	10.093	0.5%
Activiti	6.065	1.7%
ambari	6.361	3%
incubator-dubbo	1.968	17.2%
hbase	4.774	3.8%
httpcore	2.487	5.4%
Java-WebSocket	2.406	1.1%
spring-boot	1.606	16.7%
wro4j	1.645	11.3%
alluxio	3.304	1.9%

Dalla tabella si evince che lo scarto assoluto medio è inversamente proporzionale alla percentuale del progetto nel dataset. In altre parole, maggiore è il numero dei singoli test case associati al singolo progetto, maggiore sarà la capacità predittiva del modello nel determinare la coverage associata ad un nuovo test case.

Ciononostante ci sono comunque degli outliers a questa supposizione, considerando i due progetti 'http-request' e 'Activiti' la leggera differenza nella percentuale non giustifica la forte differenza nello scarto assoluto medio, ne consegue che per alcuni progetti è più facile predire la codecoverage rispetto che ad altri, ma perché?

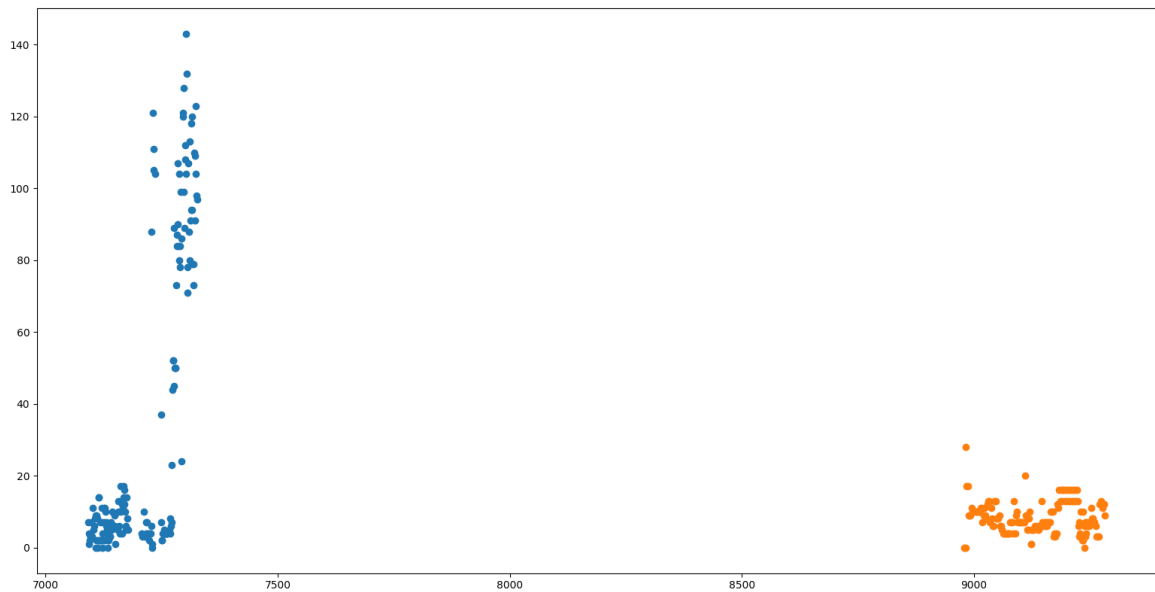


Figura 4.3: scatter plot distribuzione variabile dipendente arancio = progetto http blu = progetto attiviti

nella Figura 4.3 viene descritta la distribuzione dei coefficienti raggiunti della variabile dipendente `numCoveredLines` nei due progetti 'Activiti' in blu e 'http-request' in arancio. Dall'immagine è possibile notare che nel primo progetto i valori raggiunti dalla variabile dipendente hanno una varianza superiore rispetto che a quelli nel secondo progetto.

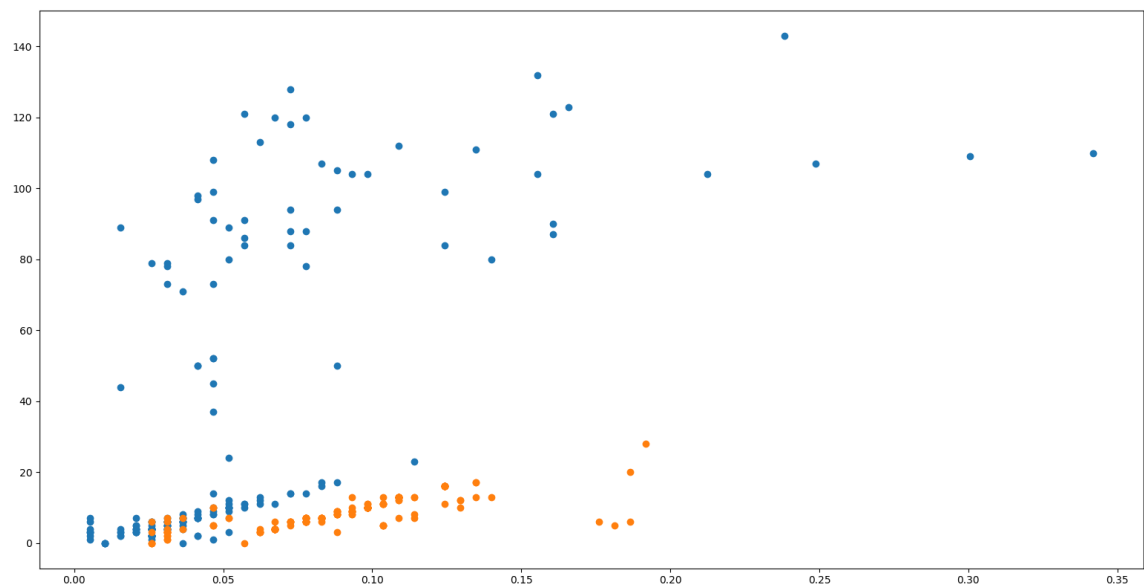


Figura 4.4

nella Figura 4.4 è mostrato un boxplot tra la variabile indipendente più significativa trovata durante la costruzione del modello 'tloc' e la variabile dipendente 'numCoveredLines'.

E' evidente come vi è una linearità più definita tra le variabili del progetto in arancione 'http-request' rispetto che alla sua controparte. Si suppone che questo comportamento del progetto 'Attività' è dovuto a istanze di test unit con una particolarità descritta da degli outliers nel data, nello specifico un picco nei valori raggiunti dalla variabile dipendente non descritto nelle variabili indipendenti considerate.

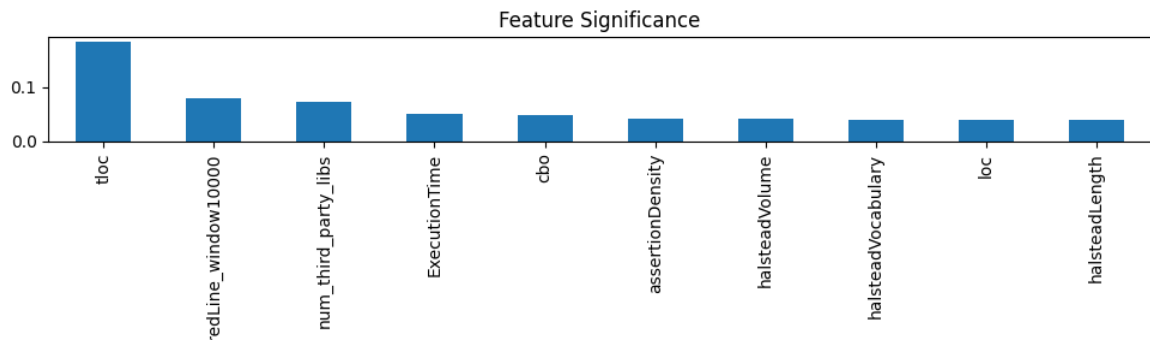
Andando a rimuovere gli outliers su tutto il dataset l'accuratezza del modello più preciso raggiunge uno scarto assoluto medio di 1.5

4.7 Analisi delle feature

Di seguito è proposta un'analisi nell'impatto che le diverse variabili indipendenti hanno avuto nell'addestramento del modello ricorrendo a tre diverse metodologie

4.7.1 Random Forest Built-in Feature Importance

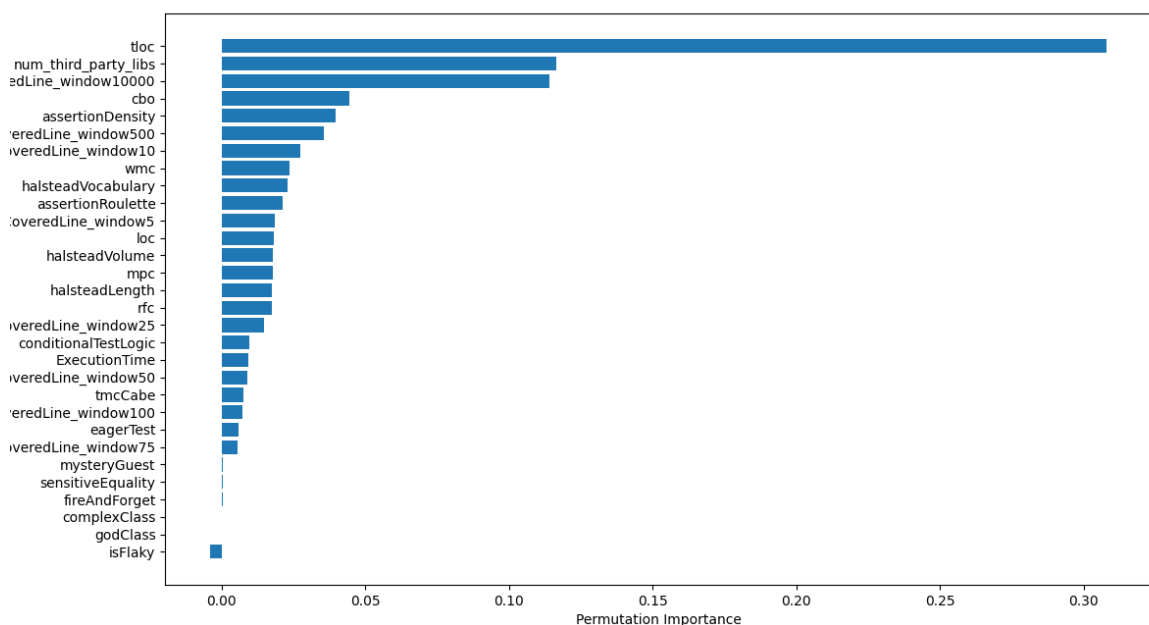
L'algoritmo Random Forest implementato nella libreria sklearn offre un metodo build-in per il calcolo delle feature importance. La feature importance attribuita alle singole variabili è determinata calcolando il Gini importance [18; 19] un coefficiente appartenente alla struttura random forest. Come esplicitato in precedenza, un random forest regressor è composto da molteplici alberi decisionali, all'interno di ogni nodo, una feature specifica è utilizzata per effettuare la decisione sulla base della quale andare a partizionare le istanze del dataset. Questa feature specifica viene selezionata sulla base di un criterio: per problemi di classificazione tale criterio è il gini impurity o information gain, per problemi di regressione è la variance reduction. E' possibile misurare come ogni variabile indipendente va a ridurre l'impurità della suddivisione delle istanze (la variabile con il decremento maggiore è quella che verrà poi selezionata per il nodo interno). Per ogni feature è possibile determinare come incide mediamente (su tutti gli alberi decisionali costruiti) sul decremento dell'impurity. Il valore medio su tutti gli alberi è la misura della feature importance. Uno degli svantaggi di questo metodo è la tendenza a preferire feature numeriche, oppure nel caso di feature con alta correlazione tra loro, viene selezionata una di esse escludendo le altre.



si evince che la variabile indipendente più significativa nel training del modello è TLOC ovvero il quantitativo di linee di codice da cui è costituito il singolo test case. Andando a rieffettuare il training del modello con il medesimo dataset ma senza la colonna 'TLOC' si ottiene uno scarto assoluto medio di **1.72**, un incremento dello 0,2 rispetto al valore ottenuto in precedenza. Ciò implica che sì la variabile TLOC è significativa nella descrizione dei singoli test case, ma non fondamentale o almeno non è l'unica in grado di fornire una descrizione della realtà trattata al modello.

4.7.2 Permutation Based Feature Importance

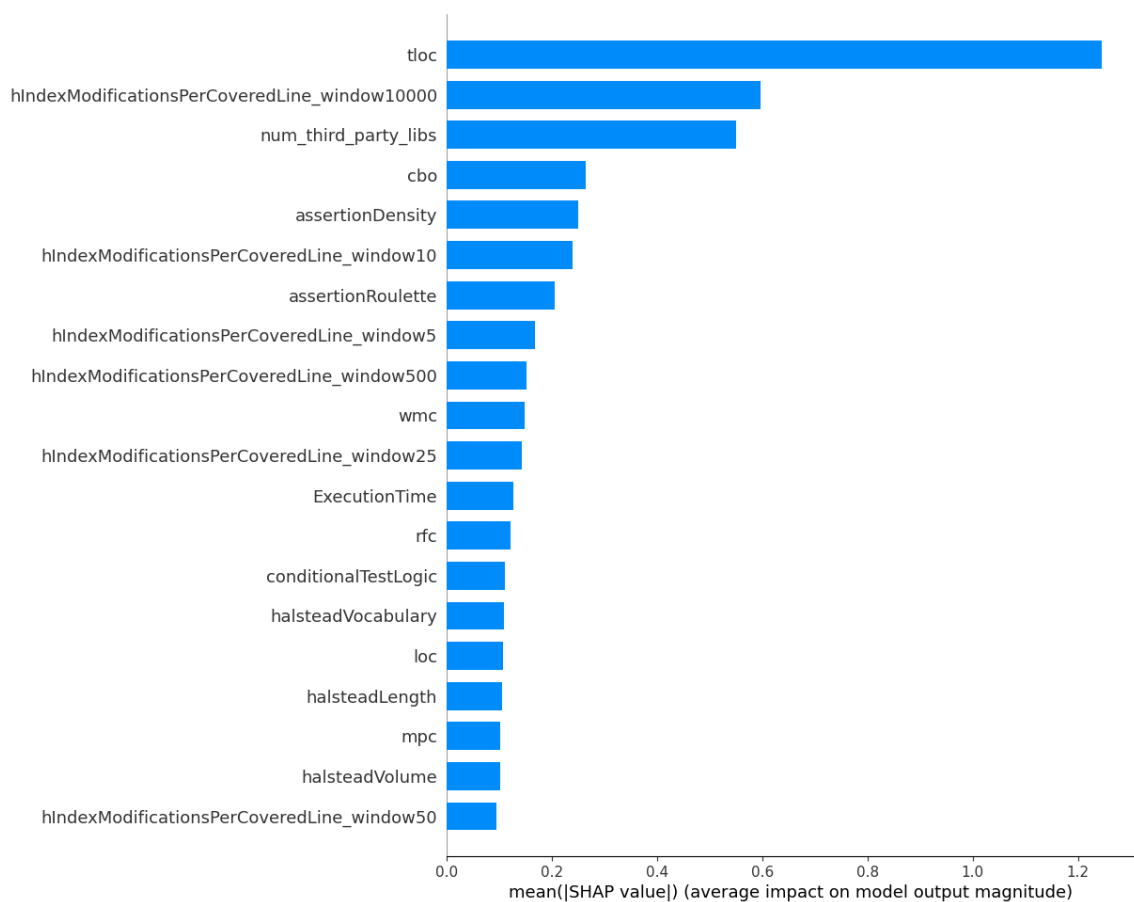
Questo metodo consiste nella selezione randomica delle feature caratterizzanti il modello e determinare l'impatto conseguente sulle performance del modello. Le features che impattano maggiormente in negativo le performance del modello quando rimosse, sono le più importanti con questo metodo nelle prime dieci feature più significative troviamo le stesse di prima con l'eccezione di wmc e assertionRoulette



4.7.3 SHAP Values

SHAP (SHapley Additive exPlanations) è un approccio che adotta la teoria dei giochi per spiegare l'output di un modello di machine learning [16].

Il primo grafo mostra l'impatto di ogni feature secondo il coefficiente SHAP, il secondo grafo invece esplicita l'impatto delle feature sulla predizione del modello in base al coefficiente delle feature stesso (blu coefficiente inferiore, rosso coefficiente superiore). Ad esempio la variabile 'tloc' è più incisiva nella predizione del modello se assume valori più alti, viceversa assumendo valori più bassi è meno utile alla predizione.



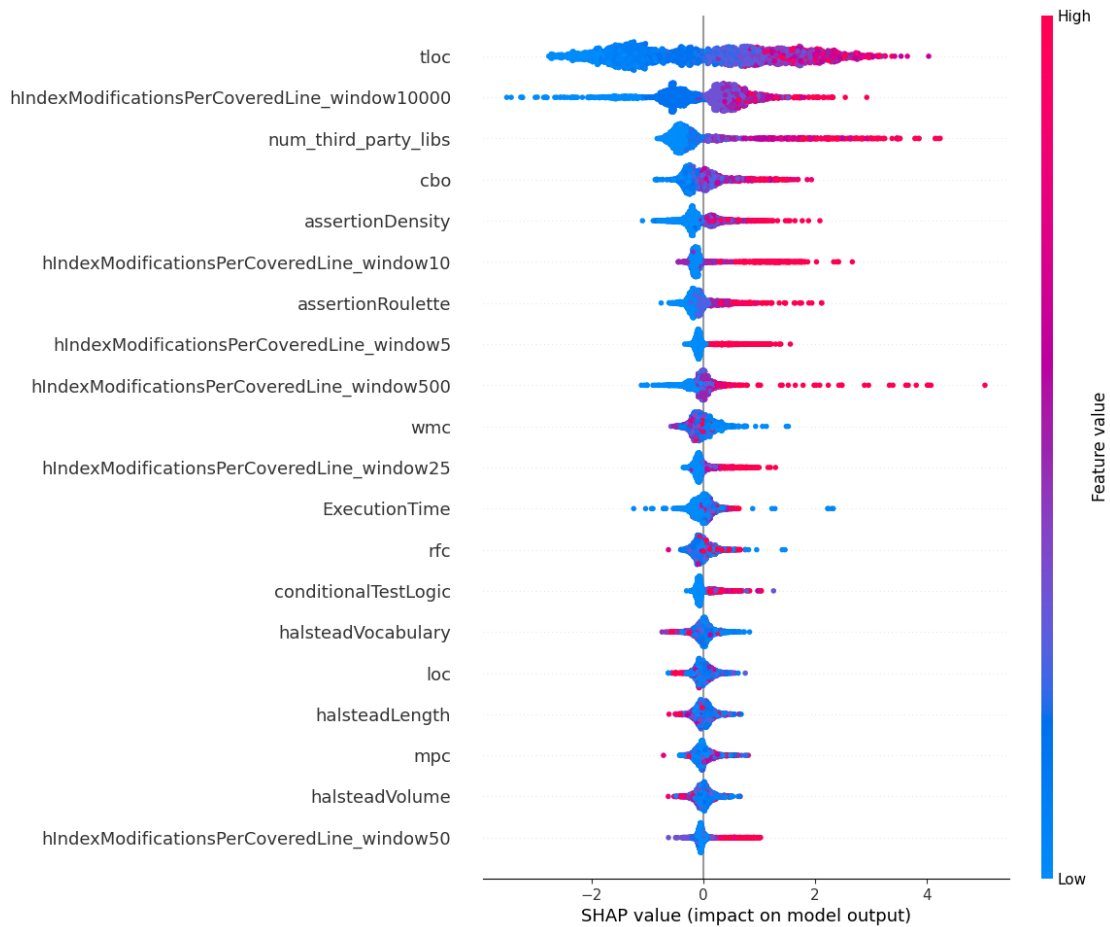


Figura 4.5

4.8 Ottimizzazione degli iperparametri

Una volta stabilita la superiorità della tipologia nel contesto, è necessario trovare la combinazione degli iperparametri ottimale. La metodologia di ricerca adottata è riconducibile a quella descritta nel caso del 'Lasso' ma aumentato leggermente il livello di profondità. Il primo passo consiste nella creazione di una griglia composta dai diversi iperparametri configurabili del modello:

1. **numero di alberi in ogni foresta**
2. **numero di feature da considerare prima dello 'split'**
3. **numero massimo della profondità di ogni albero**
4. **numero minimo of samples prima dello split in un nodo**
5. **numero minimo di elementi in un nodo foglia**
6. **bootstrap, metodo per il sampling dei data points**

Costruita la griglia, è stato eseguito il training del modello tante volte quante le combinazioni possibili, una volta terminato il processo e confrontati i risultati.

Il modello ideale ha gli iperparametri così configurati:

```
(n_estimators = 400, min_samples_split=2, min_samples_leaf=1, max_features='sqrt', max_depth = None, bootstrap=False)
```

conferendo una predizione con uno scarto di 1.40

#TUNING PARAMETRI

```
from sklearn.model_selection import RandomizedSearchCV
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
max_features = ['sqrt']
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
bootstrap = [True, False]
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
```

Nell'arco di questa tesi di laurea sono stati descritti i retroscena caratterizzanti lo sviluppo di un modello di regressione tale che, preso in input una combinazione di metriche descriventi un caso di test, restituisca in output una predizione sul numero di righe di codice ricoperte dal caso di test in questione.

Il fine primario è costituito dalla ricerca di quelle che risultano essere le metriche più preziose per il raggiungimento di tale obiettivo. Nello specifico si è constatato che a valle del lavoro compiuto, la metrica più generosa per la descrizione dello scenario al modello, risulta essere 'tloc'. In altre parole il numero di righe di codice costituenti il test case stesso è uno degli aspetti principali da considerare, se si vuole cercare di predire l'impatto del test case sul codice sorgente.

E' stato sviluppato un modello di regressione della tipologia RandomTreeForest [6] in grado di predire la variabile dipendente 'numCoveredLines' con un margine di errore assoluto di 1,41.

Occorre specificare però che al fine di massimizzare le capacità predittive del modello in questione, è stato posto un limite massimo alla dimensione del test case da analizzare. Questo poiché nel dataset studiato vi era una carenza di test case dalle grandi dimensioni, ma molto più semplicemente, all'aumentare della dimensionalità del test case risulta più ostico ottenere una predizione valida.

A valle delle considerazioni siffatte, si può dire che in questo contesto si potrebbe andare ad aumentare la dimensionalità dei test case trattati, costruendo un modello maggiormente contestualizzato alla realtà, e quindi in grado di individuare il numero di righe di codice ricoperte da un caso di test anche se tale valore risulti essere a tre cifre intere.

ricapitolando i risultati raggiunti da questa tesi:

- **uno studio statistico sulle metriche caratterizzanti il generico test case**

grazie all'applicazione del metodo di correlazione di Pearson, sono state individuate quelle che risultano essere le variabili dipendenti con una forte correlazione lineare rispetto alla variabile dipendente. In altre parole le metriche maggiormente adatte al training del modello

- **utilizzo di metodologie di explainable AI per esplorare il 'ragionamento' del modello dietro ogni sua predizione**

usufruendo di librerie come SHAP e scikit learn, è stato possibile implementare metodi in grado di fornire una panoramica sui retroscena delle predizioni eseguite dal modello. Nello specifico, un'analisi su quali metriche e sul come esse abbiano influenzato la singola predizione.

- **web app e repository**

è possibile replicare autonomamente i risultati descritti nella tesi, facendo riferimento ai contenuti reperibili nella seguente repository github: '<https://github.com/Alex31y/Thesi>' ove è possibile trovare:

1. dataset.csv il dataset trattato nella tesi

2. /codice/Random Forest Regressor.py

qui è presente lo script Python in cui viene costruito ed addestrato il modello finale, descritto nella tesi. Per il corretto funzionamento del programma occorre andare ad aggiornare il path del dataset (rigo 14)

3. /codice/test

questo script è utile per testare le predizioni del modello ottenuto, confrontando un file csv contenente le predizioni ed il file csv del dataset di partenza

inoltre, come accompagnamento alla tesi, è stata prodotta una web app reperibile al seguente link '<https://alex31y.github.io/data/>'

- [1] scipy stats libreria python. <https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.pearsonr.html>. Accessed: 2022. (Citato a pagina 21)
- [2] Mohammad Ahsanullah, BM Kibria, and Mohammad Shakil. Normal distribution. In *Normal and Student st Distributions and Their Applications*, pages 7–50. Springer, 2014. (Citato a pagina 26)
- [3] Dhaya Sindhu Battina. Artificial intelligence in software test automation: A systematic literature review. *International Journal of Emerging Technologies and Innovative Research (www.jetir.org | UGC and issn Approved)*, ISSN, pages 2349–5162, 2019. (Citato a pagina 1)
- [4] Boris Beizer. *Software Testing Techniques (2nd Ed.)*. Van Nostrand Reinhold Co., USA, 1990. (Citato a pagina 1)
- [5] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer, 2009. (Citato alle pagine 4 e 20)
- [6] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. (Citato alle pagine 5 e 40)
- [7] S.R. Chidamber and C.F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994. (Citato a pagina 14)
- [8] Arie Van Deursen, Leon Moonen, Alex Bergh, and Gerard Kok. Refactoring test code. In *Proceedings of the 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2001)*, pages 92–95, 2001. (Citato a pagina 15)

- [9] David P Doane and Lori E Seward. Measuring skewness: a forgotten statistic? *Journal of statistics education*, 19(2), 2011.
- [10] Christof Ebert, James Cain, Giuliano Antoniol, Steve Counsell, and Phillip Laplante. Cyclomatic complexity. *IEEE software*, 33(6):27–29, 2016. (Citato a pagina 3)
- [11] Rahul Gopinath, Carlos Jensen, and Alex Groce. Code coverage for suite evaluation by developers. In *Proceedings of the 36th International Conference on Software Engineering*, pages 72–82, 2014. (Citato a pagina 3)
- [12] B. Hailpern and P. Santhanam. Software debugging, testing, and verification. *IBM Systems Journal*, 41(1):4–12, 2002. (Citato a pagina 1)
- [13] Trevor Hastie. Ridge regularization: An essential concept in data science. *Technometrics*, 62(4):426–433, 2020. (Citato a pagina 27)
- [14] Cem Kaner. Exploratory testing. In *Quality assurance institute worldwide annual software testing conference*, pages 1–14, 2006.
- [15] Kenji Kira and Larry A Rendell. A practical approach to feature selection. In *Machine learning proceedings 1992*, pages 249–256. Elsevier, 1992. (Citato a pagina 16)
- [16] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017. (Citato a pagina 37)
- [17] Brian Marick, John Smith, and Mark Jones. How to misuse code coverage. In *Proceedings of the 16th Interational Conference on Testing Computer Software*, pages 16–18, 1999. (Citato a pagina 3)
- [18] Bjoern H Menze, B Michael Kelm, Ralf Masuch, Uwe Himmelreich, Peter Bachert, Wolfgang Petrich, and Fred A Hamprecht. A comparison of random forest and its gini importance with standard chemometric methods for the feature selection and classification of spectral data. *BMC bioinformatics*, 10(1):1–16, 2009. (Citato a pagina 35)
- [19] Stefano Nembrini, Inke R König, and Marvin N Wright. The revival of the gini importance? *Bioinformatics*, 34(21):3711–3718, 2018. (Citato a pagina 35)

- [20] Erick Odhiambo Omuya, George Onyango Okeyo, and Michael Waema Kimwele. Feature selection for classification using principal component analysis and information gain. *Expert Systems with Applications*, 174:114765, 2021. (Citato a pagina 16)
- [21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. (Citato a pagina 31)
- [22] Strategic Planning. The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology*, page 1, 2002. (Citato a pagina 1)
- [23] Valeria Pontillo, Fabio Palomba, and Filomena Ferrucci. Toward static test flakiness prediction: A feasibility study. In *Proceedings of the 5th International Workshop on Machine Learning Techniques for Software Quality Evolution, MaLTESQuE 2021*, page 19–24, New York, NY, USA, 2021. Association for Computing Machinery. (Citato alle pagine 4 e 12)
- [24] Martin Shepperd. A critique of cyclomatic complexity as a software metric. *Software Engineering Journal*, 3(2):30–36, 1988. (Citato a pagina 3)
- [25] Ronald L Wasserstein and Nicole A Lazar. The asa statement on p-values: context, process, and purpose, 2016. (Citato a pagina 21)