



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

Phd Valeria Pontillo

Università degli studi di Salerno

CANDIDATO

OLEXIY LYSYTSYA

Matricola: 0123456789

Anno Accademico YYYY-YYYY

INSERIRE QUI UNA DEDICA O UNA CITAZIONE

Sommario

Software testing is the act of examining the artifacts and the behavior of the software under test by validation and verification. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation.

The goal of this paper is to develop a model able to analyze an existing test case and try to predict his impact on the overall project code coverage.

Code coverage is a software testing metric that determines the number of lines of code that is successfully validated under a test procedure, which in turn, helps in analyzing how comprehensively a software is verified.

The model is based on the hypothesis that there is a **direct correlation between some or any of the characteristics of each test case and his overall project code coverage**. Le analisi statistiche trattate nel seguente articolo così come il training del modello sono state fatte su un dataset consistente in una raccolta di casi di test inerenti a diciotto diversi progetti software. L'analisi delle specifiche e la descrizione del suddetto dataset sarà approfondita in seguito. Il frutto di questo articolo consiste quindi in un modello tale che, prese in input diverse feature statiche, restituisce in output una variabile dipendente indice della coverage associata.

Indice	ii
1 Introduzione	1
1.1 Motivazioni e Obiettivi	1
1.2 Risultati	1
1.3 Struttura della tesi	1
2 Stato dell'arte	2
2.1 Regressione lineare	2
2.1.1 Simple Linear Regression	3
2.1.2 Ordinary Least Squares	3
3 Data analysis	5
3.1 Descrizione delle features	7
3.2 Analisi delle features	9
3.3 Variabile dipendente	10
3.4 Violin plot	12
3.5 Pearson correlation coefficient	13
3.6 Random Forest	14
3.7 SelectKBest	15
3.8 Recursive feature elimination	15
3.9 Recursive feature elimination with cross validation	17

4 Conclusioni	19
Ringraziamenti	20
Bibliografia	21

1.1 Motivazioni e Obiettivi

Software testing can provide objective, independent information about the quality of software and risk of its failure to users or sponsors.[3] Software testing represents the most important way developers have to check software reliability, especially when testing that newly committed code changes do not introduce new defects

1.2 Risultati

1.3 Struttura della tesi

Questo capitolo illustra lo stato dell'arte e i lavori presenti in letteratura sugli aspetti di ricerca trattati nel nostro studio, oltre che una panoramica teorica sui concetti fondamentali per l'interpretazione del seguente articolo.

2.1 Regressione lineare

La regressione lineare nasce come algoritmo statistico più di duecenti anni fa, tale metodo è stato preso in prestito dal mondo del machine learning sicché esso consente di andare a predire il valore di una variabile sulla base del valore di un'altra variabile. Quest'ultima la si può definire come variabile obiettivo, quello che si sta cercando di andare a predire e viene riferita come **variabile dipendente**. la singola o le molteplici variabili utilizzate invece per andare a predire il valore della variabile dipendente vengono definite come **variabili indipendenti**.

La regressione lineare viene definita come **modello lineare**, ossia un modello che assume una relazione lineare tra le variabili input (x) e la singola variabile output (y). Nello specifico y può essere calcolata come combinazione lineare delle x .

In altre parole, si può dire che, la regressione lineare va a modellare la relazione tra diverse variabili applicando un'equazione lineare.

Avendo origini alquanto lontane, nel corso degli ultimi secoli si può dire che la regressione lineare è stata ampiamente studiata ed analizzata sotto quasi ogni possibile forma e

sfaccettatura, di seguito verranno analizzate quattro delle principali tecniche più diffuse;

2.1.1 Simple Linear Regression

Qualora si abbia una singola variabile esplicativa 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, cerca di predire il valore della variabile dipendente in funzione della variabile indipendente. In genere la regressione lineare semplice prevede il calcolo di proprietà

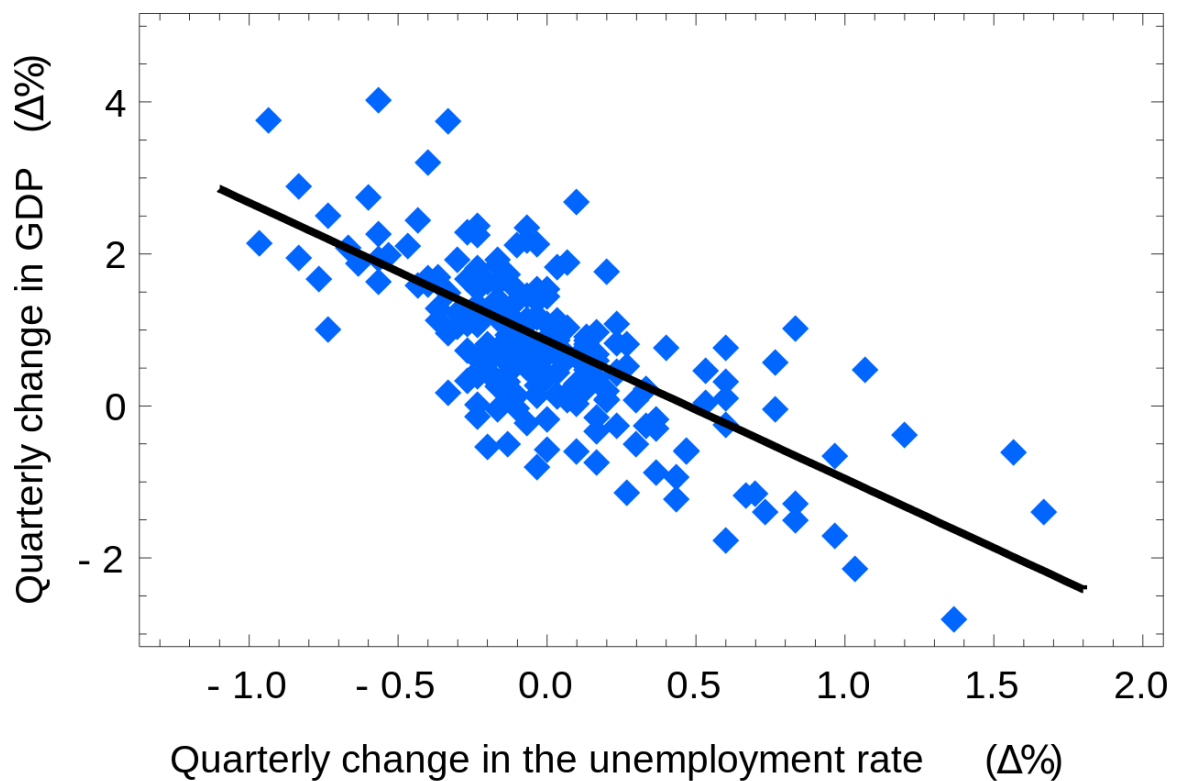


Figura 2.1: Esempio di regressione lineare semplice ove si presume una relazione lineare tra la variabile dipendente (crescita del GDP) e cambiamenti nel tasso di disoccupazione

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

2.1.2 Ordinary Least Squares

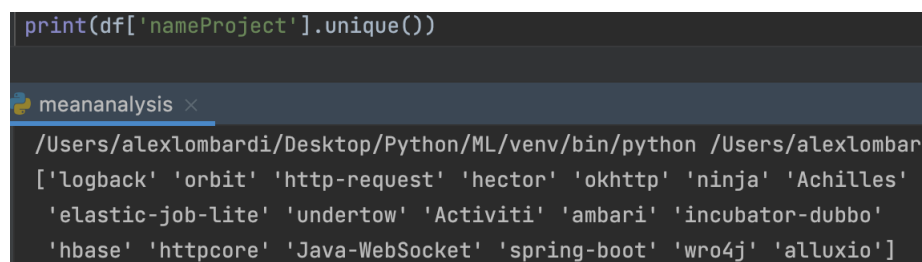
Quando si hanno più di una variabile indipendente input è possibile ricorrere alla seguente tecnica di regressione lineare. Tale procedura consiste nell'andare a minimizzare la somma

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. In genere i dati vengono approcciati sotto forma di una matrice e viene

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 [6].

Per la natura 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



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

meananalysis x
/Users/alexlombardi/Desktop/Python/ML/venv/bin/python /Users/alexlombardi/Desktop/Python/ML/venv/bin/python
['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 diversi nomi dei progetti

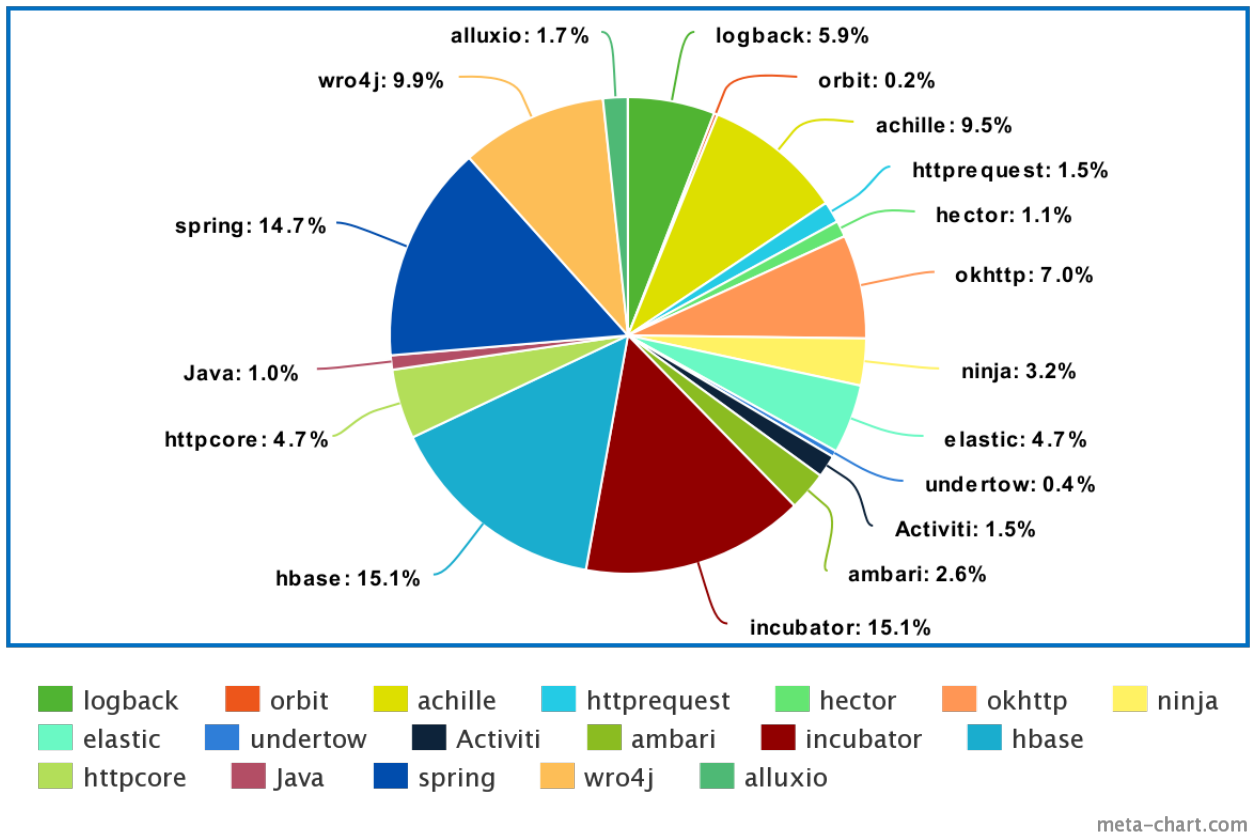


Figura 3.2: distribuzione dei test sui diversi progetti

3.1 Descrizione delle features

Production and Test Code Metrics	
Nome	Descrizione
TLOC	Number of lines of code of the test suite
TmcCabe	McCabe cyclomatic complexity, indica la complessità del codice
Lcom2	Lack of Cohesion of Methods version 2, i.e., the percentage of methods that do not access a specific attribute averaged over all attributes in the class.
Lcom5	Lack of Cohesion of Methods version 5, i.e., the density of accesses to attributes by methods.
CBO	Coupling Between Object, i.e., the number of dependencies a class has with other classes[1]
WMC	Weighted Methods per Class, i.e., the sum of the complexities (i.e., McCabe's Cyclomatic Complexity) of all the methods in a class [16]. Note that Chidamber and Kemerer [16] did not define a predefined complexity metric to consider for the computation of WMC. In our case, we opted for the McCabe metric to account for the individual complexity of methods.[1]
RFC	Response For a Class, i.e., the number of methods (including inherited ones) that can potentially be called by other classes[1]
MPC	Message Passing Coupling, measures the numbers of messages passing among objects of the class.
Halstead Vocabulary	The total number of distinct operators and operands in a function
Halstead Length	The total number of operator occurrences and the total number of operand occurrences.
Halstead Volume	Proportional to program size, represents the size, in bits, of space necessary for storing the program.
numCoveredLines	Total number of lines of code covered by the test
executionTime	Running time for the test execution
projectSourceLinesCovered	Total number of production classes covered by each test
hIndexModPerCoverLine	hIndex capturing churn of covered lines in past 5, 10, 25, 50, 75, 100, 500, and 10,000 commits. Each value h indicates that at least h lines were modified at least h times in that period.

Code smells	
Nome	Descrizione
classDataShouldBePrivate	When a class exposes its attributes, violating the information hiding principle.
complexClass	When a class has a high cyclomatic complexity.
functionalDecomposition	When in a class inheritance and polymorphism are poorly used
godClass	When a class has huge dimension and implementing different responsibilities.
spaghettiCode	When a class has no structure and declares long method without parameters.

Text smells	
Nome	Descrizione
Assertion density	percentage of assertion statements in the test code
Assertion roulette	undocumented assertions in the test code
Mystery Guest	Il test presenta materiale esterno[2]
Eager test	Il test analizza più metodi contemporaneamente[2]
Sensitive equality	Il test presenta un confronto sul toString[2]
Resource Optimism	Il test fa uso di risorse esterne potenzialmente non disponibili[2]
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

Feature Selection and classification have previously been widely applied in various areas like business, medical and media fields. High dimensionality in datasets is one of the main challenges that has been experienced in classifying data, data mining and sentiment analysis.[4]

L'obiettivo è quello di andare a costruire un modello tale che, preso in input un test case descritto con feature statiche, 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 dati in pasto delle informazioni sicché esso possa comprendere ed essere in grado le tematiche relative al problema. 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 sì, 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**: situazione in cui un modello si adatta troppo bene ai dati di training e, di conseguenza, non può prevedere 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. Un'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, la validità caratteristica è indirettamente proporzionale alla profondità del dataset usato nel training.

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.

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", analizziamole entrambe

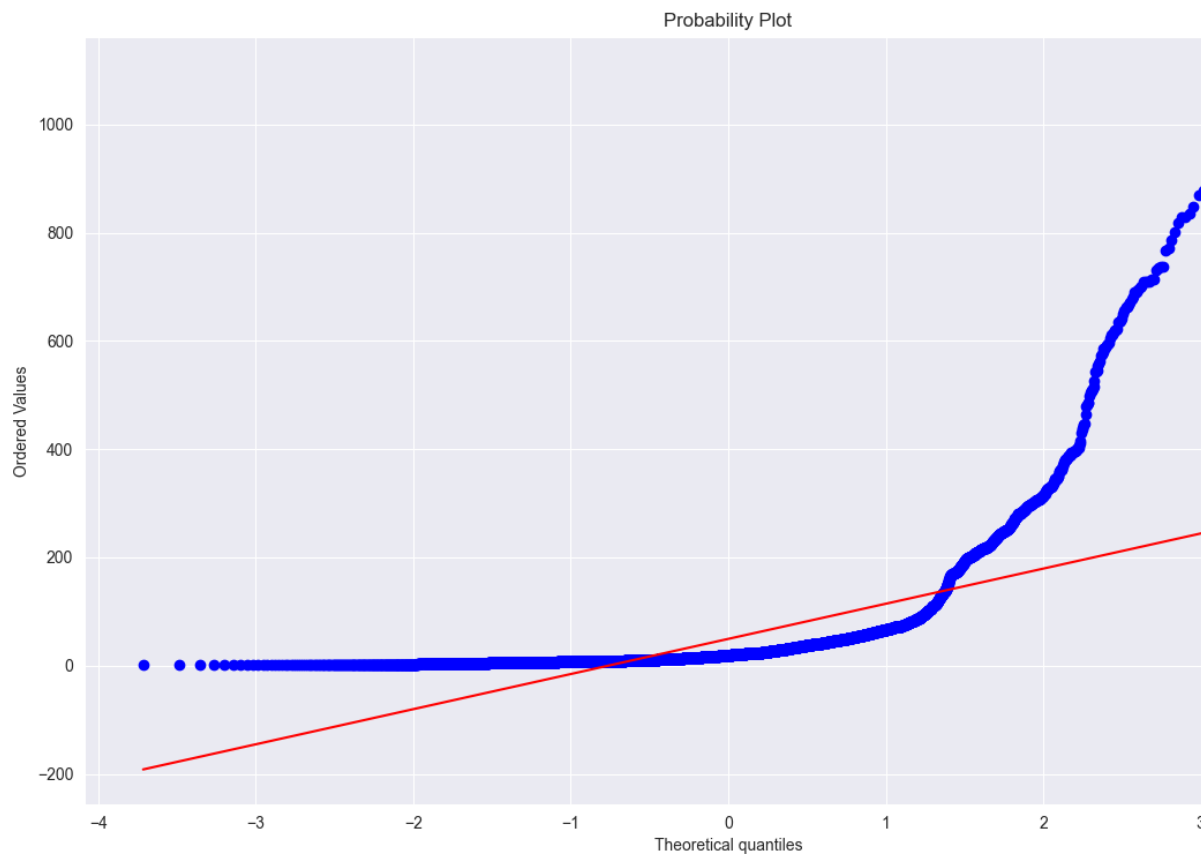


Figura 3.3: probability plot di 'projectSourceLinesCovered'

Considerando le figure 3.3 e 3.4 è possibile analizzare la distribuzione dei valori di queste due feature. Ci si accorge subito che numCoveredLines ha una distribuzione più uniforme rispetto alla sua controparte, ciononostante entrambe le variabili hanno una distribuzione asimmetrica verso destra, in gergo right skewness [CITA].

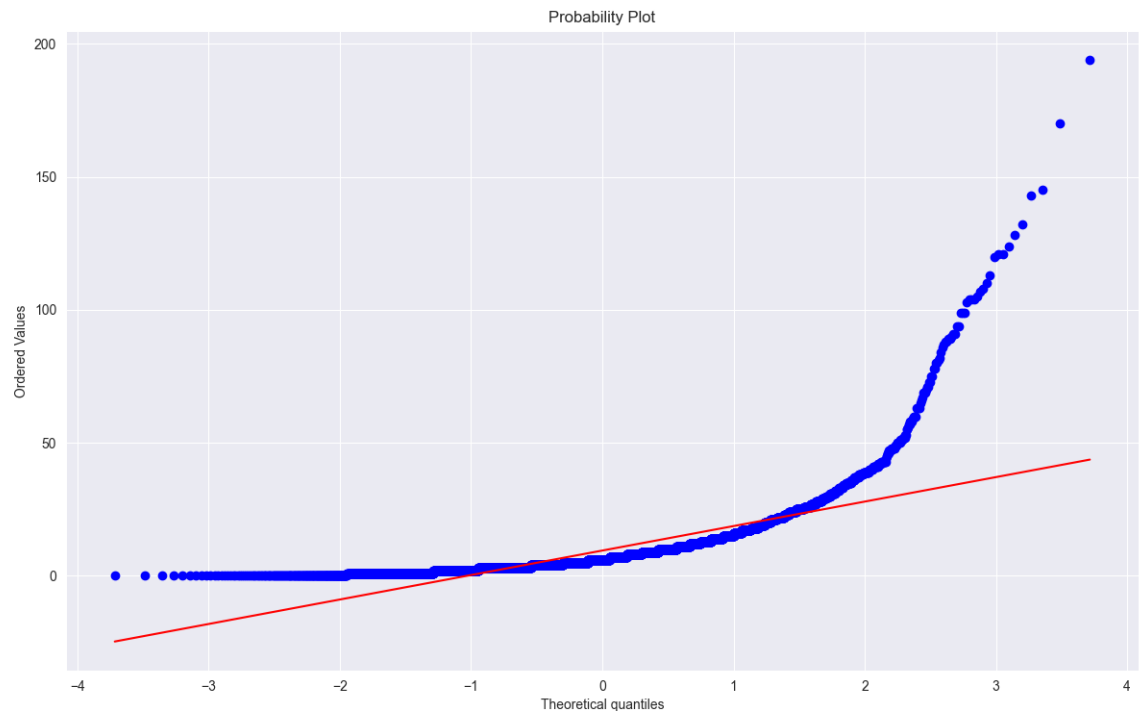


Figura 3.4: probability plot di 'numCoveredLines'

Sarebbe più interessante ed utile ai fini dell'addestramento del modello, se la variabile dipendente assumesse un'andamento uniforme lineare (normal distribution) pertanto si andrà a fare un lavoro di feature engineering per migliorare la nostra variabile dipendente.

Prima di partire con l'analisi occorre soffermarsi un attimo su cosa stiamo cercando di fare: fare una scrematura delle informazioni contenute all'interno del dataset per prendere quelle più polarizzanti, in questo caso i due 'poli' sono: il test ha una coverage soddisfacente, il test non ha una coverage soddisfacente.

Per riuscire ad analizzare graficamente i dati, alcuni dei plots che andremo ad utilizzare prevedono l'utilizzo di una variabile dipendente binaria e quindi che possa assumere esattamente due valori. come si evince dalla figura 3.2 però, la natura delle due features è incompatibile a quanto appena detto, per ovviare questa problematica banalmente andrò a ristrutturare l'intera colonna facendo in modo che il valore della singola istanza sia uguale a 0 se il numero di righe coperte è inferiore alla media, 1 altrimenti (figura 3.3).

```
.describe di 'numCoveredLines'
count      9785.000000
mean        9.555442
std         11.882775
min          0.000000
25%         3.000000
50%         6.000000
75%        12.000000
max        194.000000
Name: numCoveredLines, dtype: float64
.describe di 'projectSourceLinesCovered'
count      9785.000000
mean       49.100766
std        94.357798
min         1.000000
25%         9.000000
50%        19.000000
75%        46.000000
max       1098.000000
Name: projectSourceLinesCovered, dtype: float64

Process finished with exit code 0
```

Figura 3.5: descrizione delle due feature candidate

```
data = pd.read_csv("dataset.csv")
# y e j sono le potenziali variabili dipendenti
y = data.numCoveredLines
j = data.projectSourceLinesCovered
y.loc[y < 9] = 0
y.loc[y >= 9] = 1
```

Figura 3.6: polarizzazione della variabile dipendente

3.4 Violin plot

In questa e nelle successive sezioni, saranno mostrati ed analizzati una serie di ‘plots’ comunemente utilizzati nel mondo del data science, il primo di cui usufruiremo è il violin plot.

Quello che stiamo cercando è una o più feature sicché i due possibili stati della variabile dipendente, in riferimento alla feature in analisi, assuma valori il più diverso possibili per stato. Quello che stiamo cercando quindi è una feature tale che il suo violin plot corrispondente abbia le righe orizzontali centrali, ovvero quelle rappresentanti la mediana, il più lontane possibili sull’asse delle ordinate. Un buon esempio è quello che troviamo nella figura 3.4 ovvero la violin plot della feature floc. Dal grafo è possibile notare che la distribuzione dei

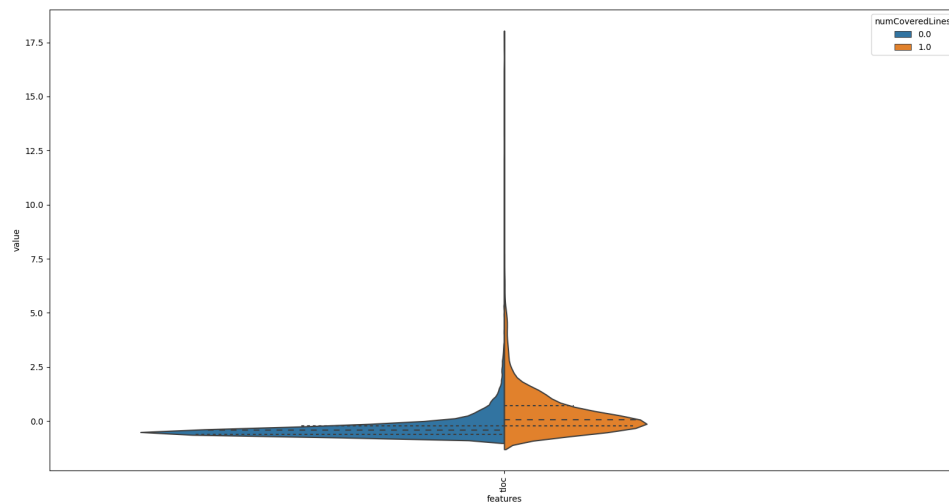


Figura 3.7: violin plot della feature 'tloc'

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 stiamo cercando è il caso presente nella figura 3.5: qui si può notare che 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. Altre feature interessanti sono in ordine di potenziale: CBO +++, LOC, RFC, MPC, halsteadVocabulary, halsteadLength, halsteadVolume

3.5 Pearson correlation coefficient

In statistics, the Pearson correlation coefficient is a measure of linear correlation between two sets of data. It is the ratio between the covariance of two variables and the product of their standard deviations; thus, it is essentially a normalized measurement of the covariance, such that the result always has a value between -1 and 1. [7]

Judging by the graph, there is a perfect correlation between 'rft' and 'mpc', and also between the three 'halstead' features. Let's simplify the dataset by removing any feature with high correlation; in other words let's drop any features which isn't going to give any new information regarding the situation to the model during the training; let's drop for example: 'lcom2', 'mpc', 'halstead length' and 'halstead volume'. There are still going to be some features with high correlation (0.9+) but we can fix that later

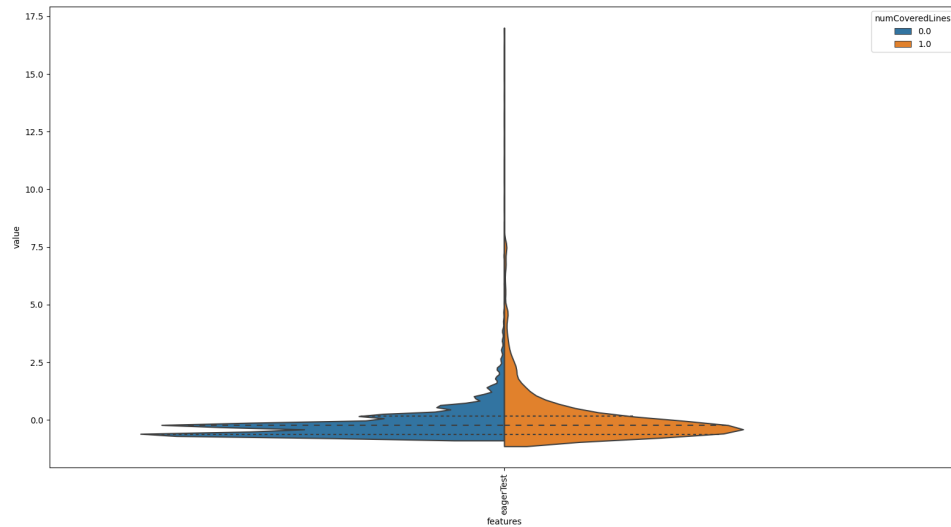


Figura 3.8: violin plot della feature 'EagerTest'

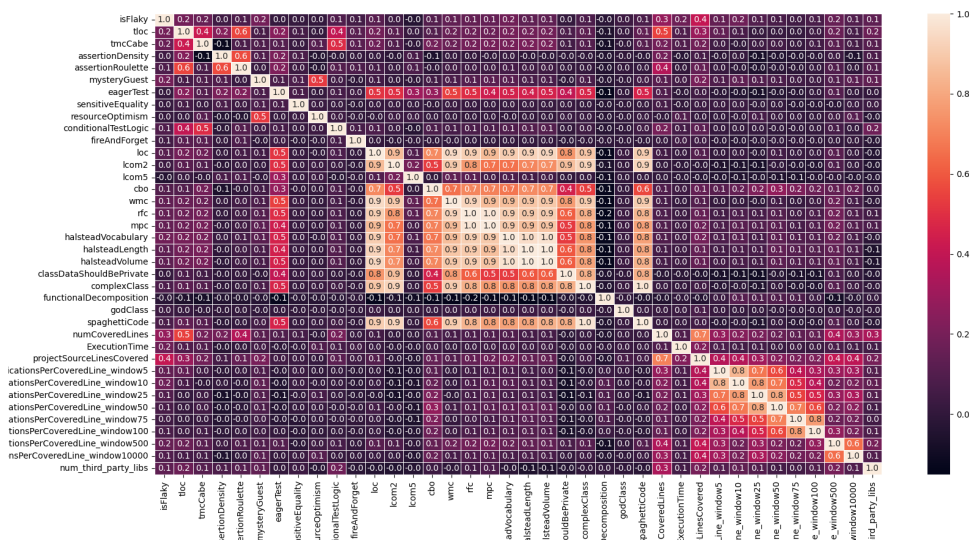


Figura 3.9: pearson applicato all'intero dataset

3.6 Random Forest

A random forest is a machine learning technique that's used to solve regression and classification problems. It utilizes ensemble learning, which is a technique that combines many classifiers to provide solutions to complex problems. Here the random forest algorithm is used as an index to the efficiency of our analysis.

The dataset is going to be partitioned so that a random 30% of entries are going to be used for testing only. After compiling a few tries we can see that:

```

17 # split data train 70 % and test 30 %
18 x_train, x_test, y_train, y_test = train_test_split(df, y, test_size=0.3, random_state=42)
19
20 #random forest classifier with n_estimators=10 (default)
21 clf_rf = RandomForestClassifier(random_state=43)
22 clf_rf = clf_rf.fit(x_train, y_train)
23
24 ac = accuracy_score(y_test, clf_rf.predict(x_test))
25 print('Accuracy is: ', ac)

```

Figura 3.10: Python code snippet

- By polarizing the dependent variable (so we're guessing if the input test unit has a coverage greater or lower than the median value, in this case 9 loc), the accuracy of the random tree forest reaches 89%
- If we instead consider the regression problem, so we are trying to guess the distinct amount of lines covered by the test, we get an accuracy of 40%
by just increasing the n-estimators to 100 inside our RandomForestClassifier, we reach 43% accuracy, not an amazing result but this could be seen as our starting point

3.7 SelectKBest

Univariate feature selection works by selecting the best features based on univariate statistical tests. It can be seen as a preprocessing step to an estimator. Scikit-learn exposes feature selection routines as objects that implement the transform method: SelectKBest removes all but the k highest scoring features[5]

the result implies that the best five feature are: "tloc", "assertionRoulette", "hIndexModificationsPerCoveredLinewindow10000" "hIndexModificationsPerCoveredLinewindow500", "numthirdpartylibs" so let's try training our model featuring just those features. The resulting accuracy this time reaches 38% of correct guesses, that's lower than our last result [Figure 3.9].

3.8 Recursive feature elimination

Now let's try with Feature ranking with recursive feature elimination. Given an external estimator that assigns weights to features (e.g., the coefficients of a linear model), the goal of recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of features and

```

select_feature = SelectKBest(f_regression, k=5).fit(x_train, y_train)

topfeature = "nome " + x_train.columns + "val: " + select_feature.scores_.astype(str)
print('Score list:', topfeature)
print(select_feature.scores_)

```

randomTree ×

C:\Users\xlits\PycharmProjects\ML1\venv\Scripts\python.exe C:/Users/xlits/PycharmProjects/
Accuracy is: 0.7503405994550408
Score list: Index(['nome isFlakyval: 575.7776011580305', 'nome tlocval: 2776.936896345742',
'nome tmcCabeval: 261.9708661738497',
'nome assertionDensityval: 268.6751612477164',
'nome assertionRouletteval: 943.22598802096',
'nome mysteryGuestval: 100.24009219547432',
'nome eagerTestval: 58.94377501431061',
'nome sensitiveEqualityval: 18.726333959898188',
'nome resourceOptimismval: 0.02518566314527672',
'nome conditionalTestLogicval: 238.64308077916922',
'nome fireAndForgetval: 4.361820485024651',
'nome locval: 19.93383303243937', 'nome lcom5val: 0.31073321050458913',
'nome cboval: 59.62580713510306', 'nome wmcval: 27.795897631491687',
'nome rfcval: 51.338071227277396',
'nome halsteadVocabularyval: 62.922676614434856',
'nome classDataShouldBePrivateval: 3.939096771194413',
'nome complexClassval: 3.4714810428868157',
'nome functionalDecompositionval: 11.081978186972792',
'nome godClassval: 0.0019506730251395259',
'nome spaghettiCodeval: 2.842055075471971',
'nome numCoveredLinesval: 1.5418073324302887e+19',
'nome ExecutionTimeval: 45.63576021011903',
'nome hIndexModificationsPerCoveredLine_window5val: 565.0504048788076',
'nome hIndexModificationsPerCoveredLine_window10val: 406.73593871725285',
'nome hIndexModificationsPerCoveredLine_window25val: 296.9067463231873',
'nome hIndexModificationsPerCoveredLine_window50val: 184.50275607866192',
'nome hIndexModificationsPerCoveredLine_window75val: 120.00211199329614',
'nome hIndexModificationsPerCoveredLine_window100val: 142.61756621707158',
'nome hIndexModificationsPerCoveredLine_window500val: 959.0765997030578',
'nome hIndexModificationsPerCoveredLine_window1000val: 641.0403255168083',
'nome num_third_party_libsval: 872.267334767079'],

Figura 3.11: Python code snippet

the importance of each feature is obtained either through any specific attribute or callable. Then, the least important features are pruned from current set of features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.[5]

Basically, it uses one of the classification methods (random forest in our example), assign weights to each of features. Whose absolute weights are the smallest are pruned from the

current set features. That procedure is recursively repeated on the pruned set until the desired number of features

```
#Recursive feature elimination
from sklearn.feature_selection import RFE
# Create the RFE object and rank each pixel
clf_rf_3 = RandomForestClassifier()
clr_rf_3 = clf_rf_3.fit(x_train,y_train)
rfe = RFE(estimator=clf_rf_3, n_features_to_select=8, step=1)
rfe = rfe.fit(x_train, y_train)

print('Chosen best 5 feature by rfe:',x_train.columns[rfe.support_])
ac_3 = accuracy_score(y_test,rfe.predict(x_test))
print('Accuracy is: ',ac_3)
```

randomTree x

```
/Users/alexlombardi/Desktop/Thesi/codice/venv/bin/python /Users/alexlombardi/Desktop/Thesi/cod
Accuracy is: 0.4315395095367847
Chosen best 5 feature by rfe: Index(['tloc', 'loc', 'cbo', 'wmc', 'rfc', 'halsteadVocabulary',
      'ExecutionTime', 'num_third_party_libs'],
      dtype='object')
Accuracy is: 0.4087193460490463

Process finished with exit code 0
```

Figura 3.12: Python code snippet

This time the resulting optimal features are: 'tloc', 'loc', 'cbo', 'wmc', 'rfc', 'halsteadVocabulary', 'ExecutionTime', 'numThirdPartyLibs' but still the accuracy of our randomTree is lower than if we use all of the available features [Figure 3.10].

3.9 Recursive feature elimination with cross validation

As of now we gave as an input a limit to the optimal amount of features we're looking for. But actually we have no idea what's the optimal amount of features to consider for training our model; to help answer this question we could use `sklearn.feature-selection.RFECV` [Figure 3.11]

```
#Recursive feature elimination with cross validation
from sklearn.feature_selection import RFECV

# The "accuracy" scoring is proportional to the number of correct classifications
clf_rf_4 = RandomForestClassifier()
rfecv = RFECV(estimator=clf_rf_4, step=1, cv=5, scoring='accuracy') #5-fold cross-validation
rfecv = rfecv.fit(x_train, y_train)

print('Optimal number of features :', rfecv.n_features_)
print('Best features :', x_train.columns[rfecv.support_])
```

randomTree ×

```
Optimal number of features : 28
Best features : Index(['isFlaky', 'tloc', 'tmcCabe', 'assertionDensity', 'assertionRoulette',
                     'mysteryGuest', 'eagerTest', 'sensitiveEquality', 'resourceOptimism',
                     'conditionalTestLogic', 'loc', 'lcom5', 'cbo', 'wmc', 'rfc',
                     'halsteadVocabulary', 'functionalDecomposition', 'spaghettiCode',
                     'ExecutionTime', 'hIndexModificationsPerCoveredLine_window5',
                     'hIndexModificationsPerCoveredLine_window10',
                     'hIndexModificationsPerCoveredLine_window25',
                     'hIndexModificationsPerCoveredLine_window50',
                     'hIndexModificationsPerCoveredLine_window75',
                     'hIndexModificationsPerCoveredLine_window100',
                     'hIndexModificationsPerCoveredLine_window500',
                     'hIndexModificationsPerCoveredLine_window10000',
                     'num_third_party_libs'],
                     dtype='object')
```

Process finished with exit code 0

Figura 3.13: Python code snippet

CAPITOLO 4

Conclusioni

BREVE SPIEGAZIONE CONTENUTO CAPITOLO

Ringraziamenti

INSERIRE RINGRAZIAMENTI QUI

Bibliografia

- [1] 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 7)
- [2] 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 8)
- [3] Cem Kaner. Exploratory testing. In *Quality assurance institute worldwide annual software testing conference*, pages 1–14, 2006. (Citato a pagina 1)
- [4] 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 9)
- [5] 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 alle pagine 15 e 16)
- [6] 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 a pagina 5)
- [7] Wikipedia contributors. Pearson correlation coefficient — Wikipedia, the free encyclopedia, 2022. [Online; accessed 18-July-2022]. (Citato a pagina 13)

Siti Web consultati

- Wikipedia – www.wikipedia.org