MACHINE LEARNING FOR SOFTWARE ENGINEERING

Nome: Baba Adrian Petru
Matricola: 0320578
Università degli studi Tor Vergata anno 2021/2022

Agenda

- Introduzione e scopo del lavoro svolto
- Metodologia adottata
- Presentazione e discussione dei risultati ottenuti
  -Bookkeeper
  -Storm
- Conclusioni

Introduzione

-Una strategia per diminuire **sforzi e risorse**, temporali ed economiche, dell'attività di testing è sfruttare il machine learning per predire dove gli sforzi andrebbero dirottati. Lo scopo del lavoro svolto è di analizzare e di confrontare i parametri prestazionali di tre classificatori- nello spacifico **IBK, NaiveBayes e RandomForest-** con l'utilizzo o meno di strategie differenti atte, possibilmente, a migliorare le loro prestazioni nella predizione della buggyness

-I tre classificatori sono stati addestrati su due dataset contenenti dati che rappresentino in qualche modo la **difettosità** delle classi di due progetti opensource

-I progetti opensource presi in considerazione in questo lavoro sono **Apache Bookkeeper** e **Apache Storm**

-Le tecniche considerate sono tecniche di balancing, nello specifico **undersampling**, **SMOTE** e **oversampling**: si vuole effettuare un confronto tra le tre strategie ponendo enfasi nel confronto specifico tra gli ultimi due in quanto operativamente simili, ovvero ambedue aggiungono istanze per la classe minoritaria facendo in modo che pareggi quella maggioritaria

Metodologia- raccolta dati preliminari

-Per effettuare il lavoro di cui si è parlato, c'è stato bisogno di una fase iniziale di **data mining**, per costruire un **dataset** da utilizzare

-Tale dataset è stato creato con il programma «**DataCollector**»

-Il primo passo è stato ricavare i **ticket** dei progetti da JIRA

-Un'importante numero di ticket è stato scartato perché le informazioni contenute non sono coerenti, nello specifico:

1-Affected version presenti ma Opening Version assente
2-Opening Version = Fixed Version
3-Affected Version >= Fixed Version
4-Fixed Version assente

Metodologia- raccolta dati preliminari

-E’ importante sottolineare che con i vincoli imposti prima, un grandissimo numero di ticket è stato **scartato**

-Scartare un importante numero di ticket rende plausibile l’idea che i dati successivamente raccolti possano essere parzialmente **inaccurati**, in quanto presenti delle mancanze

-Ciò avviene in particolare nel caso di **bookkeeper**: il numero totale di ticket non è eccessivamente grande e con questi filtri precipita di molto

Metodologia- raccolta dati preliminari

-Per i ticket ritenuti validi, se le affected version erano presenti allora la più lontana dal presente è stata considerata come **prima affected version**

-Quando queste non erano presenti, l'affected version è stata calcolata tramite **proportion** con un approccio **cold start**

-Si è preferito l'approccio cold start in quanto è stato considerato l'approccio più **neutro**: gli altri sarebbero potuti essere più **inaffidabili** dato l'elevato numero di ticket scartati

-Con quest'approccio, tenendo conto dei dati disponibili su tutti i progetti apache, il valore proporzionale **P** utilizzato è **1,5**

Metodologia- scelta dati per il Dataset

-All'interno del dataset, per ciascuna classe di ciascuna release, sono stati calcolati i dati specificati in seguito in quanto considerati significativi per definire la buggyness

1- **N. REVISIONI**: numero di volte in cui il file è stato toccato da un commit
un numero alto indica una maggiore probabilità che siano stati aggiungi bug

2- **LOC**: numero di righe di codice
un alto numero di righe implica una maggiore probabilità che, da qualche parte all'interno della classe, ci sia un bug

3- **LOC TOUCHED**: totale righe cambiate nella classe, ovvero la somma tra le aggiunte, cancellate e modificate
un alto valore indica che la classe è stata modificata spesso e/o pesantemente: è facile dunque che siano stati introdotti bug

Metodologia- scelta dati per il Dataset

4- **MAX LOC ADDED**: numero massimo di righe aggiunte tra tutti i commit subiti
Un alto valore indica che la classe ha subito aggiunte pesanti e quindi una più alta probabilità che dei bug siano stati introdotti

5- **AVERAGE LOC ADDED**: numero medio di righe aggiunte tra i vari commit
Un valore alto indica che la classe è stata arricchita pesantemente nel tempo

6-**CHURN**: Differenza in valore assoluto tra righe aggiunte e cancellate
Anche in questo caso un valore alto indica che la classe ha subito pesanti modifiche nel tempo, che siano cancellazioni o aggiunte

7-**MAX CHURN**: valore massimo assunto dal churn tra i vari commit
un alto valore indica che la classe ha subito almeno una volta pesanti aggiunte o cancellazioni

8-**AVERGAE CHURN**: media del valore assunto dal churn tra i vari commit

Metodologia- scelta dati per il Dataset

9-**AGE**: età della classe, in numero di settimane da quando è stata aggiunta
può assumere significati diversi in base alle altre metriche

10-**WEIGHED AGE**: età pesata, calcolata facendo **AGE*TOUCHED LOC**
un valore alto indica che non solo la classe ha una certa età ma è stata anche modificata molto spesso

11-**NUMBER OF AUTHORS**: numero di programmatori distinti che hanno messo mano sulla classe
Un valore alto indica che programmatori diversi hanno lavorato sulla stessa classe, quindi una probabilità più alta che qualcuno che non conoscesse a fondo la classe l'abbia modificata introducendo bug

Metodologia- elaborazione dati

-Ultimo parametro per il dataset è la **buggyness**. Sono state considerate buggy tutte le classi che hanno subito commit a partire dalla affected version arrivando alla fixed version, **esclusa**

-Per contenere il problema dello **snoring**, release- e relativi dati- risalenti ad **oltre la metà del tempo di vita** del progetto sono stati scartati- è stato ritenuto un approccio migliore rispetto allo scartare la metà netta delle realease

-La valutazione è avvenuta applicando l'algoritmo di **walk forward**.
Alcune iterazioni delle iniziali sono state saltate in quanto non si avevano casi di classi buggy. Lo split del dataset è stato fatto sempre in «**DataCollector**»

Metodologia- ottenimento risultati

-I risultati sono stati ottenuti e stampati in un file csv attraverso il programma «**MLtoISW**»

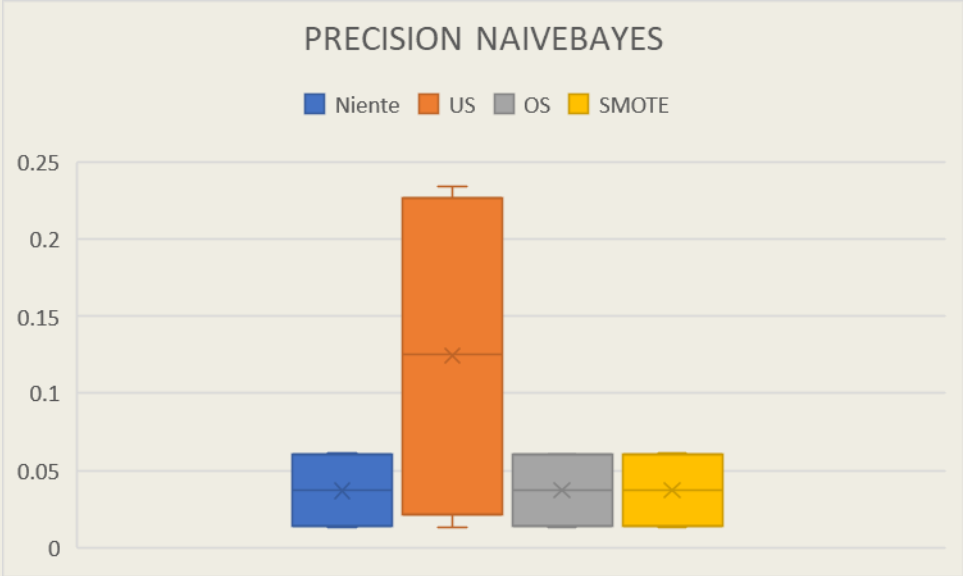
-Il programma calcola tutte le combinazioni studiate a lezione per le varie tecniche di balancing, di feature selection e di modifica alla cost sensitivity

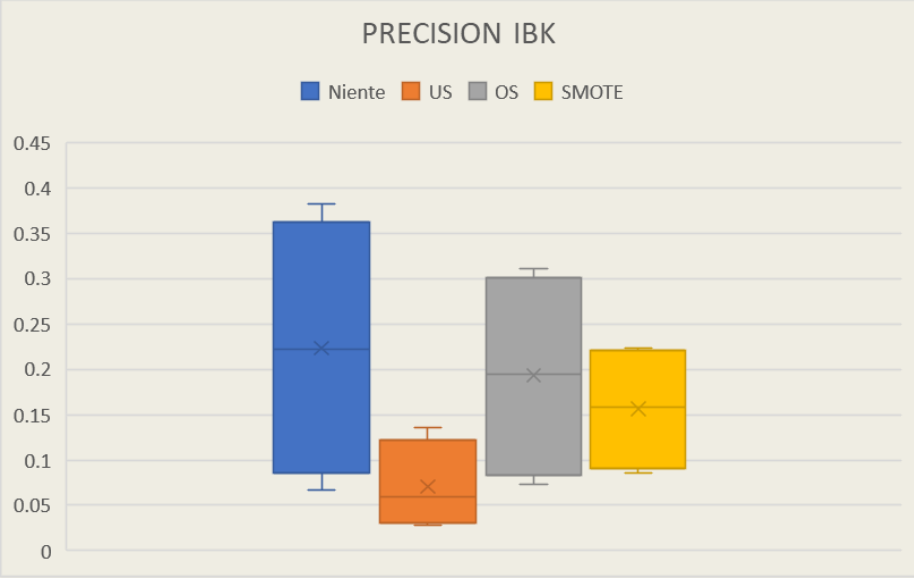
-Per lo scopo di questa presentazione, tuttavia, verrà effettuato un confronto applicando esclusivamente le tecniche di **balancing**

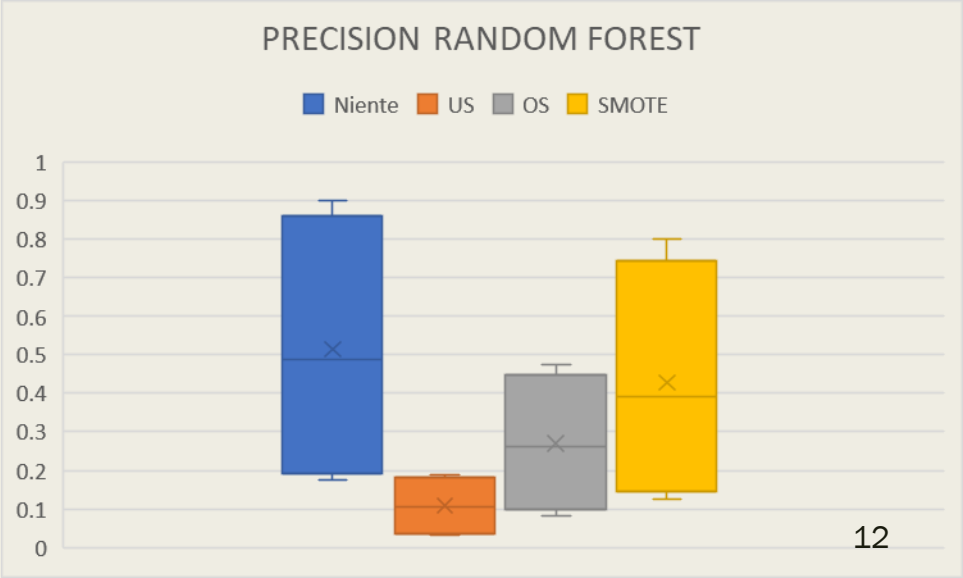
Risultati BK- Precision

Di seguito l'analisi delle prestazioni sul progetto BookKeeper.
Legenda per le seguenti slide:
Niente: classificatore senza balancing applicato
US: Undersampling
OS: Oversampling
SMOTE: Smote

In questa slide nello specifico abbiamo le varie **precision** per ciascuno dei classificatori con ciascun filtro applicato







Risultati BK- Precision

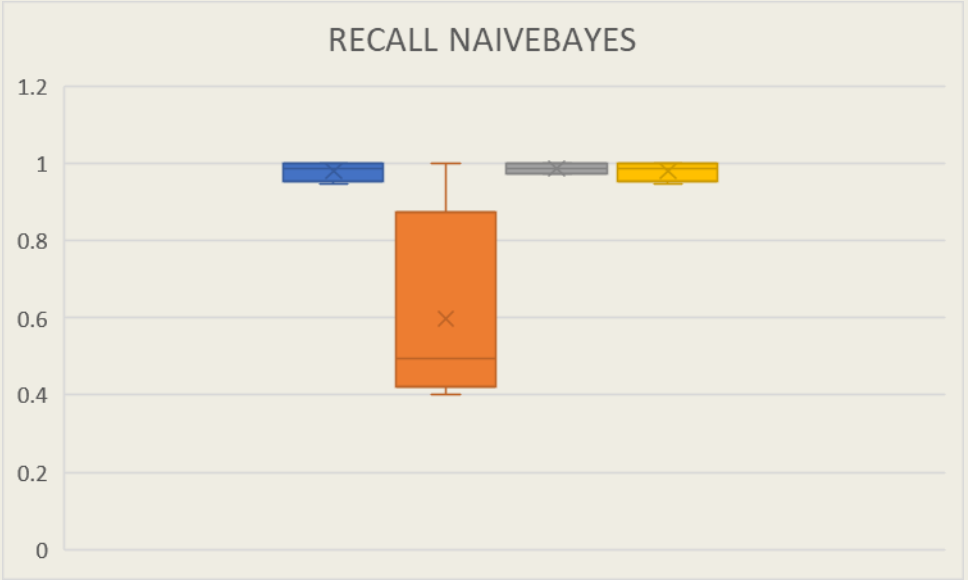
-La precision media **diminuisce** anche notevolmente quando applicata una qualsiasi tecnica di balancing, tranne nel caso dell'undersamplig con Naive Bayes dove **aumenta** notevolmente, mentre resta uguale per gli altri approcci

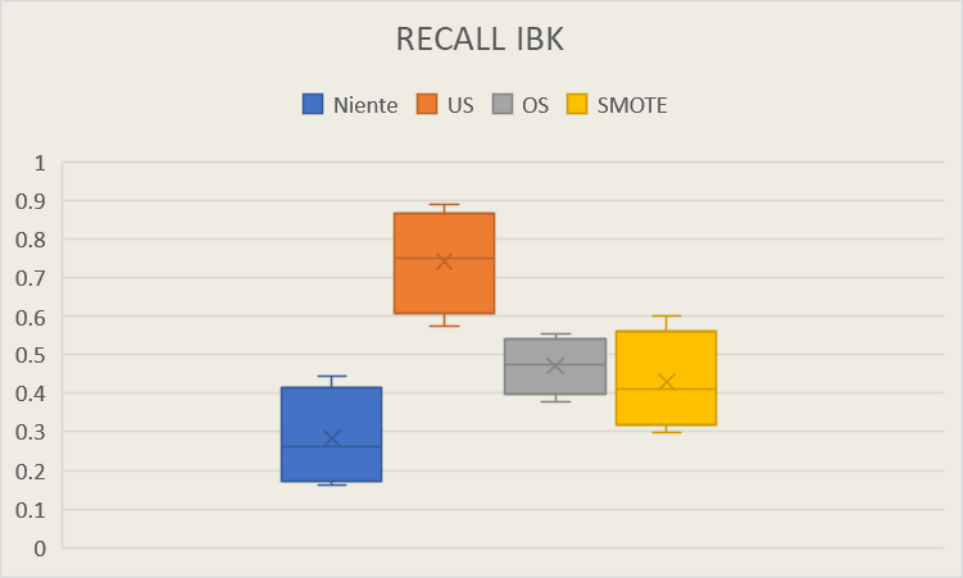
-Va nuovamente sottolineato come i dati di questo progetto nonché le iterazioni di walkforward **sono piuttosto pochi**: l'affidabilità dei dati in output è questionabile

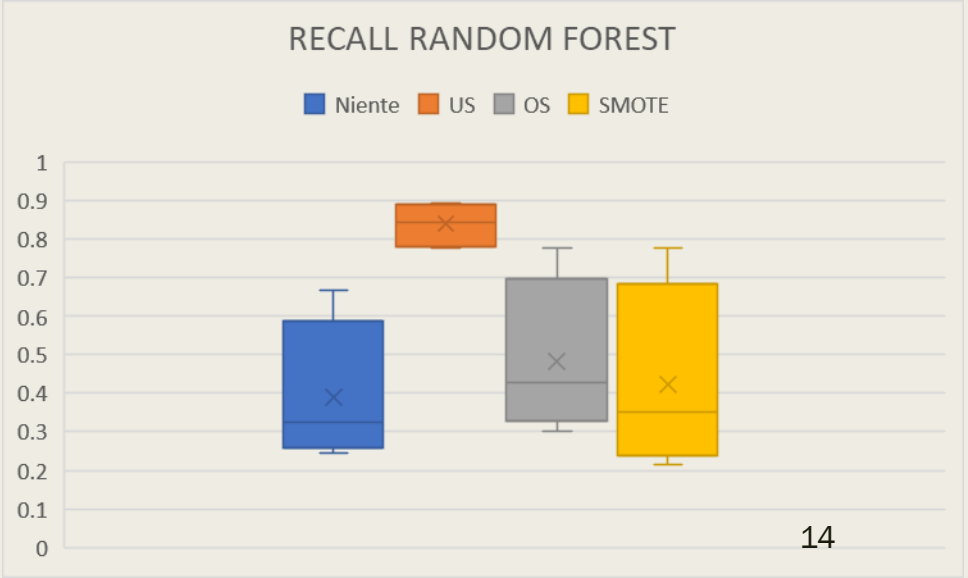
-Tra smote e oversampling si ha un **bilanciamento** delle prestazioni medie tra i tre classificatori: nel caso NaiveBayes sono praticamente le stesse, nel caso IBK l'oversampling ha un valore medio migliore a fronte di una maggiore varianza, nel caso Random Forest è SMOTE a dare un valore medio migliore seppure con una maggiore varianza

Risultati BK- Recall

Nel caso della recall l'undersampling- tranne per il NaiveBayes- **migliora** le prestazioni ed è migliore rispetto agli altri approcci.
Tra SMOTE e Oversmapling, nel caso della recall, **quest'ultimo** è migliore per tutti e 3 i classificatori, con una media maggiore e minore varianza.
Ambedue gli approcci **migliorano sempre** le prestazioni del classificatore senza balancing.

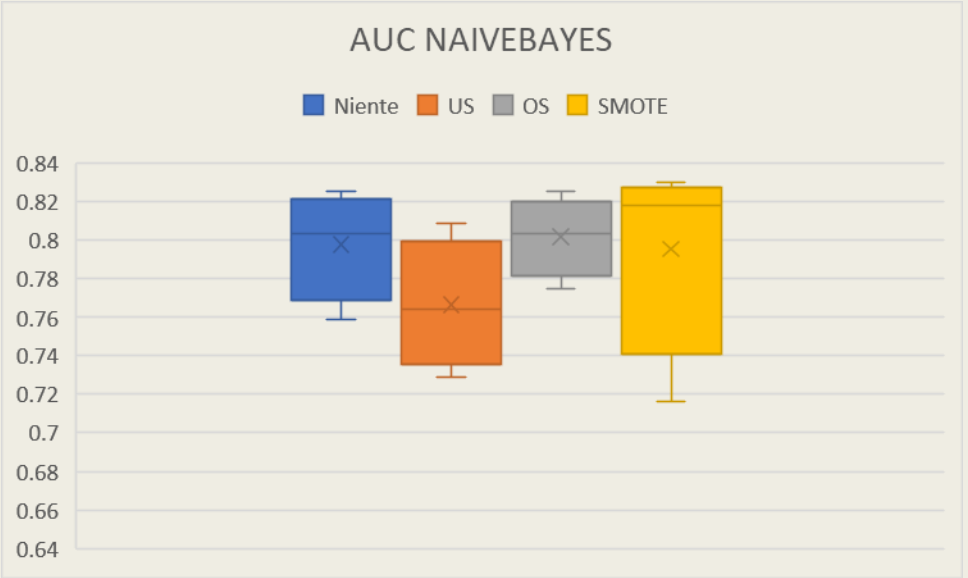


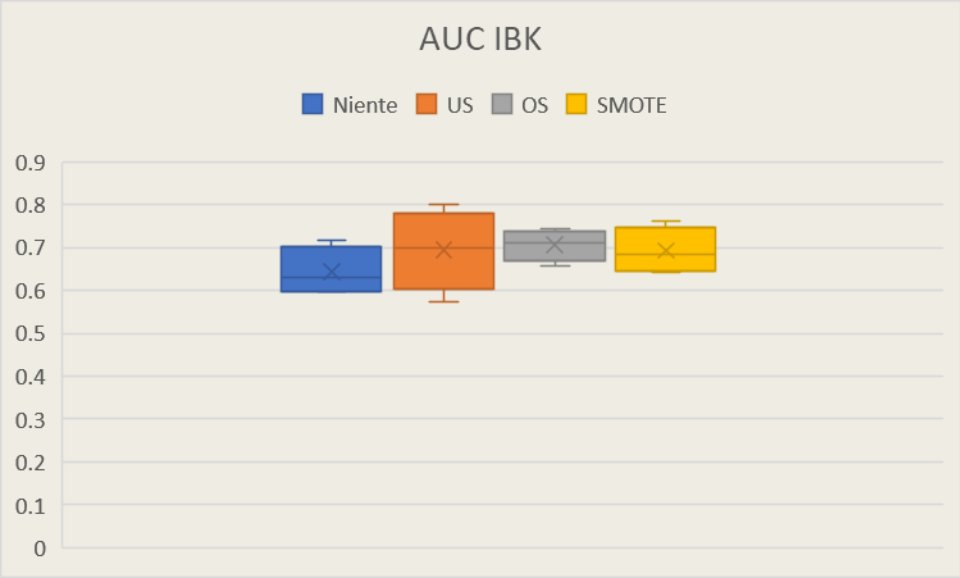


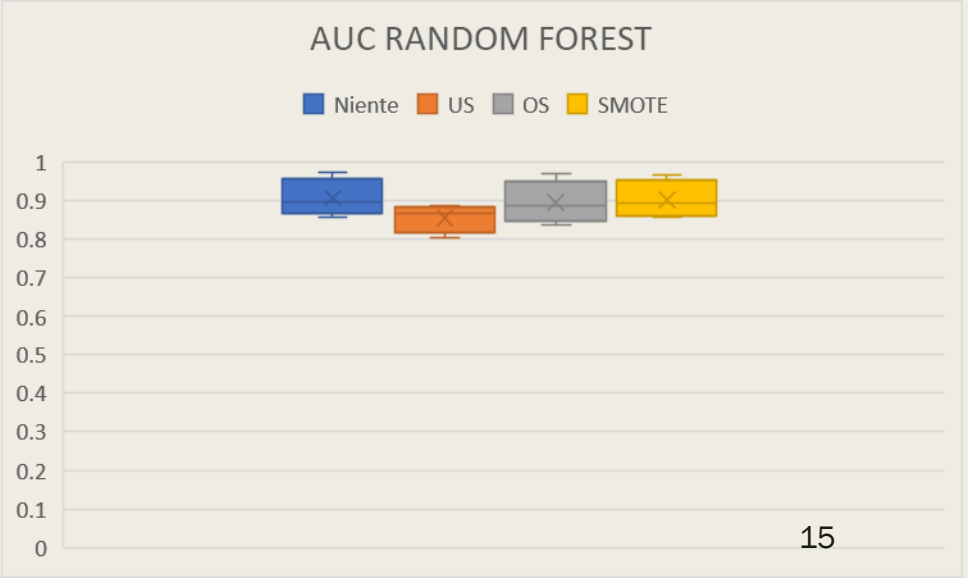


Risultati BK- Area Under ROC

Nel caso dell' AUC l'undersampling è **peggiore** rispetto agli altri approcci e generalmente peggiore rispetto al caso senza balancing.
Tra SMOTE e Oversampling, per i classificatori IBK e RandomForest le prestazioni sono **simili**, mentre per il NaiveBayes lo SMOTE **migliora** le prestazioni, a fronte di una grande variabilità

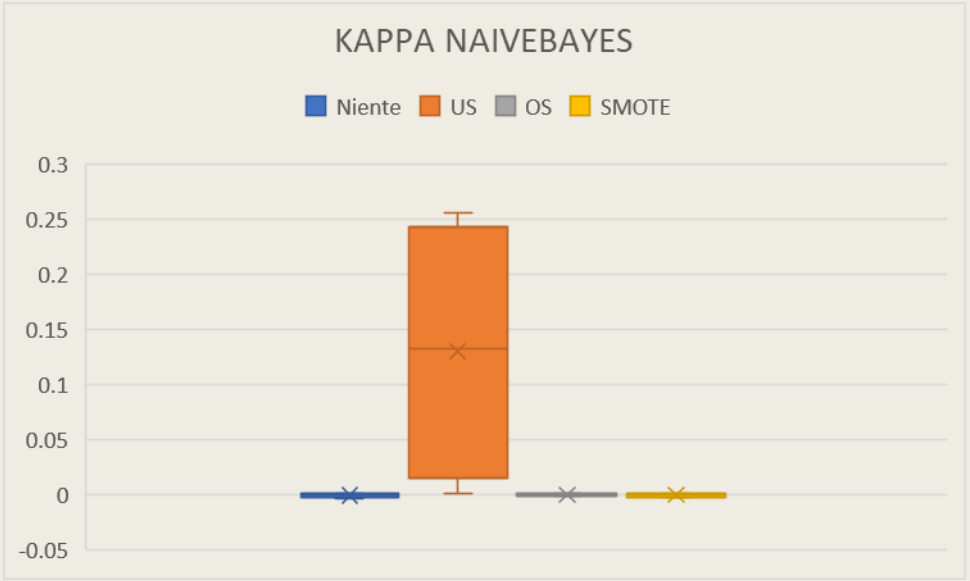


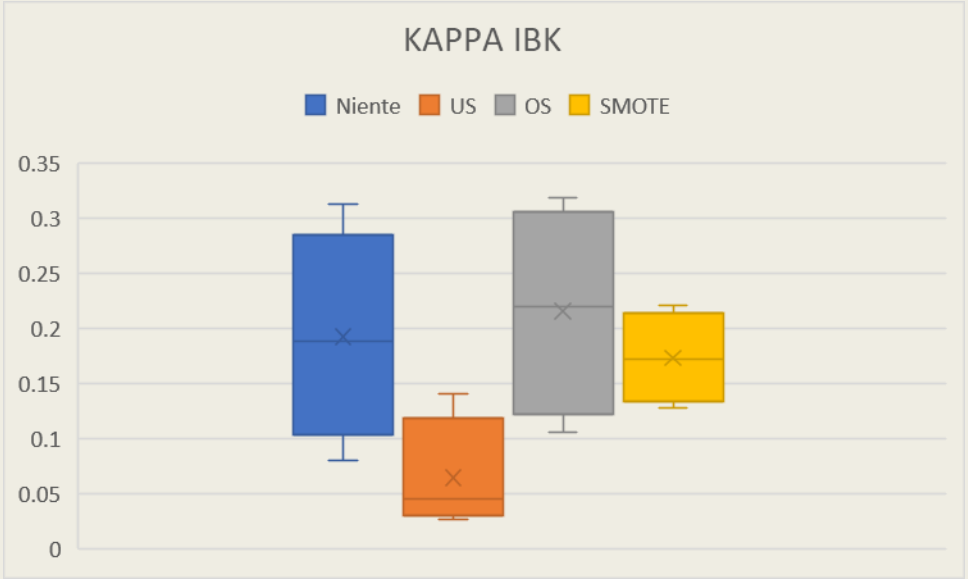


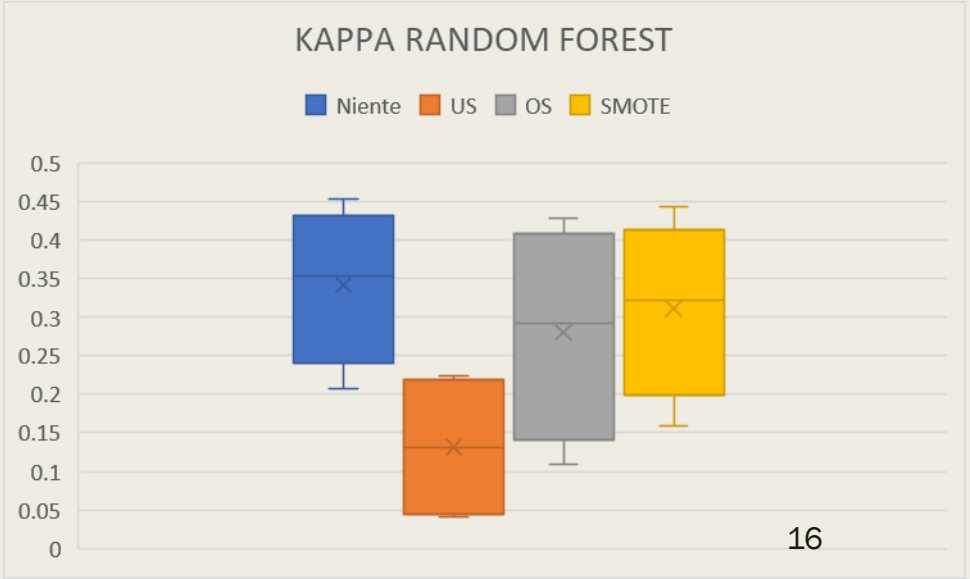


Risultati BK- Kappa

Nel caso del kappa, l'undersampling produce un **netto peggioramento** per IBK e Random Forest e l'opposto per Naive Bayes, sia rispetto al caso senza balancing che agli altri due approcci.
Tra SMOTE e Oversampling nello specifico, con NaiveBayes **non abbiamo differenze**, con RandomForest SMOTE migliora leggermente sia la media diminuendo la variabilità mentre con IBK è l'Oversampling ad avere la media migliore, a fronte di una molto maggiore variabilità.



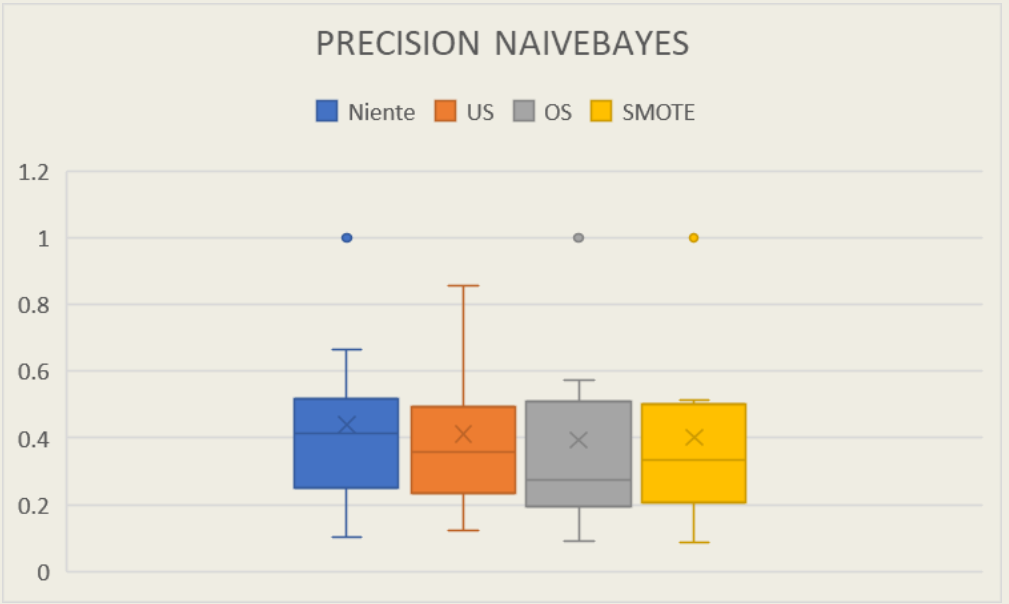


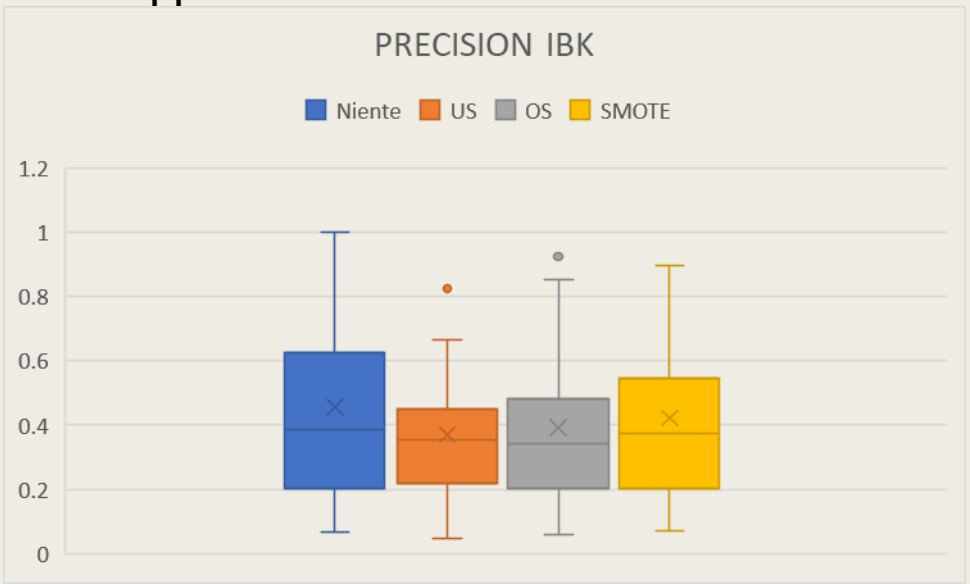


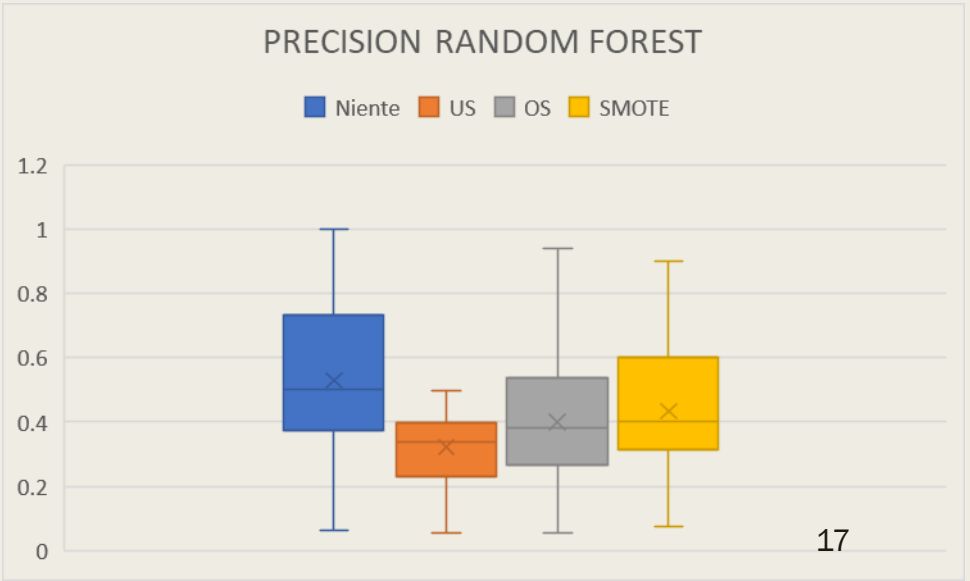
Risultati STORM- Precision

Di seguito l'analisi delle prestazioni sul progetto STORM.
Nelle seguenti slide:
Niente: classificatore senza balancing applicato
US: Undersampling
OS: Oversampling
SMOTE: Smote

In questa slide nello specifico abbiamo le varie **precision** per ciascuno dei classificatori con ciascun filtro applicato







Risultati STORM- Precision

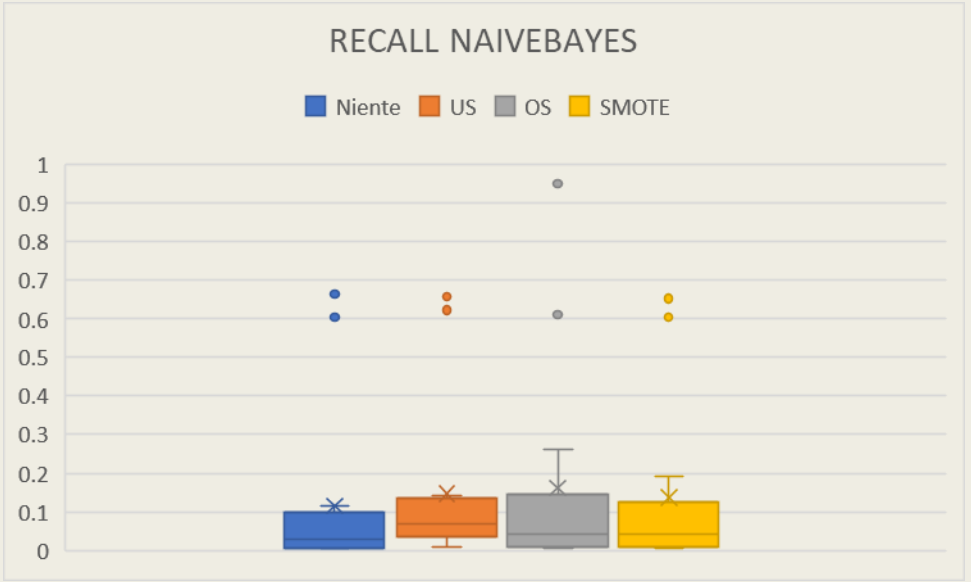
-Nel caso di questo progetto **la quantità di dati** a disposizione e di iterazioni nell'applicazione del walk forward **è di molto maggiore**: ci si aspettano dati più vicini alla realtà e alle aspettative

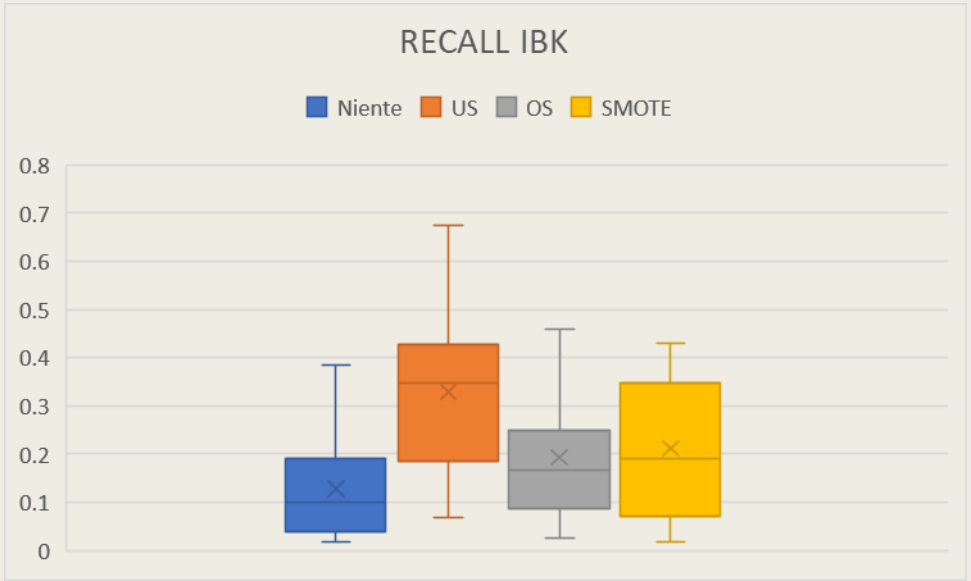
-L'utilizzo di una qualsiasi tra le tre techinche di balancing **non produce alcun miglioramento** nelle prestanzioni, per nessuno dei tre classificatori. L'undersampling ha generalmente le prestazioni peggiori, tranne nel caso del Naive Bayes in cui, rispetto alle altre tecniche, ha una mediana leggermente superiore ma con stesso valor medio

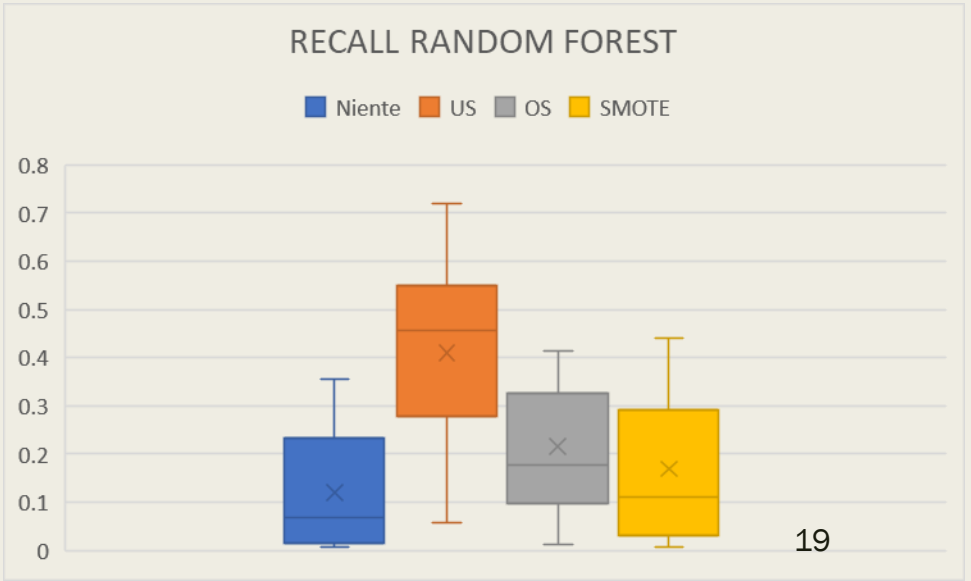
-Nello specifico tra SMOTE e Oversampling, in tutti e tre i **il primo vince**: ha una mediana e una media maggiore rispetto al secondo, anche se di poco

Risultati STORM- Recall

-In questo caso l' undersampling produce un **miglioramento** più o meno netto per tutti e tre i classificatori, molto più degli altri due approcci.
-Nello specifico tra Oversampling e SMOTE il primo generalmente ha una variabilità maggiore ma i miglioramenti su medie e mediane sono uguali, quindi SMOTE **potrebbe essere preferibile**

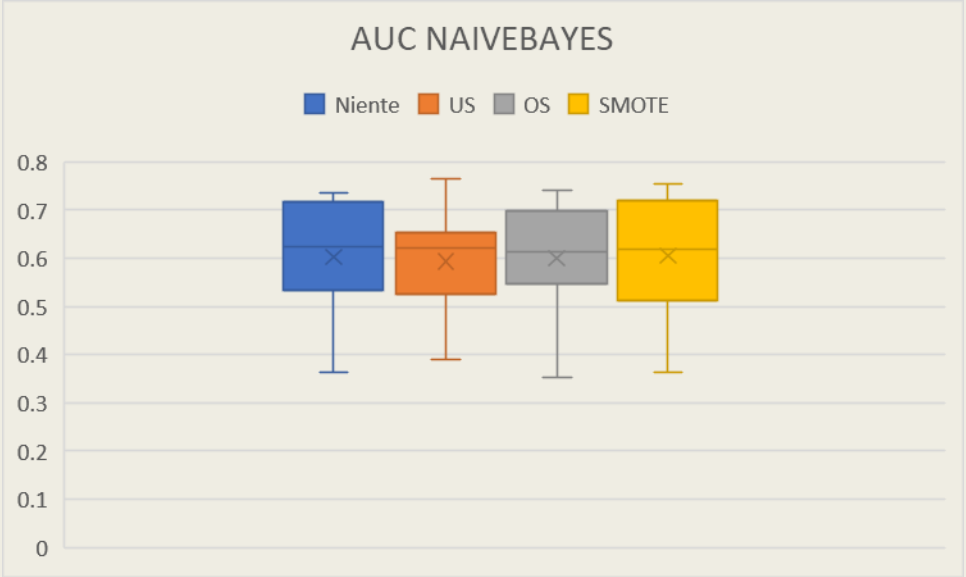


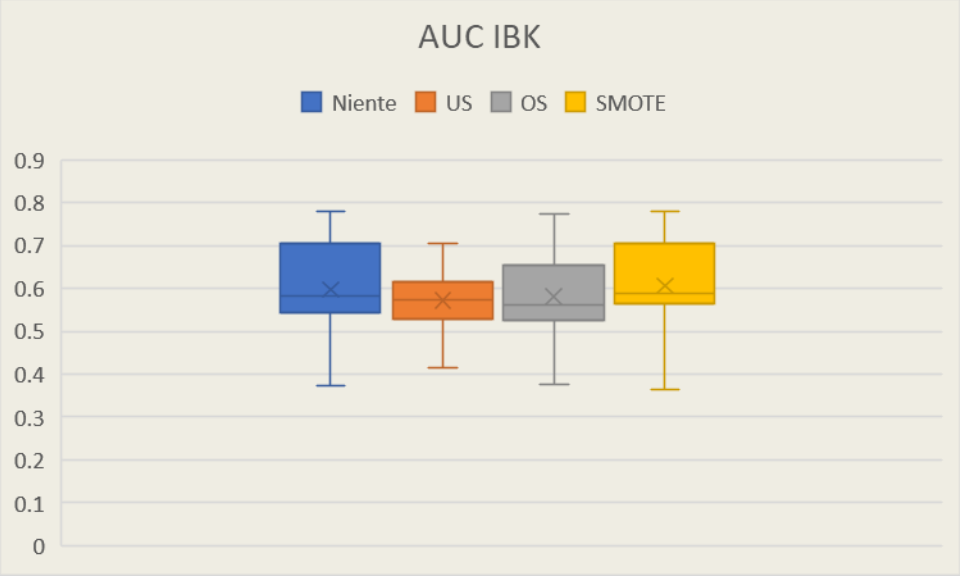


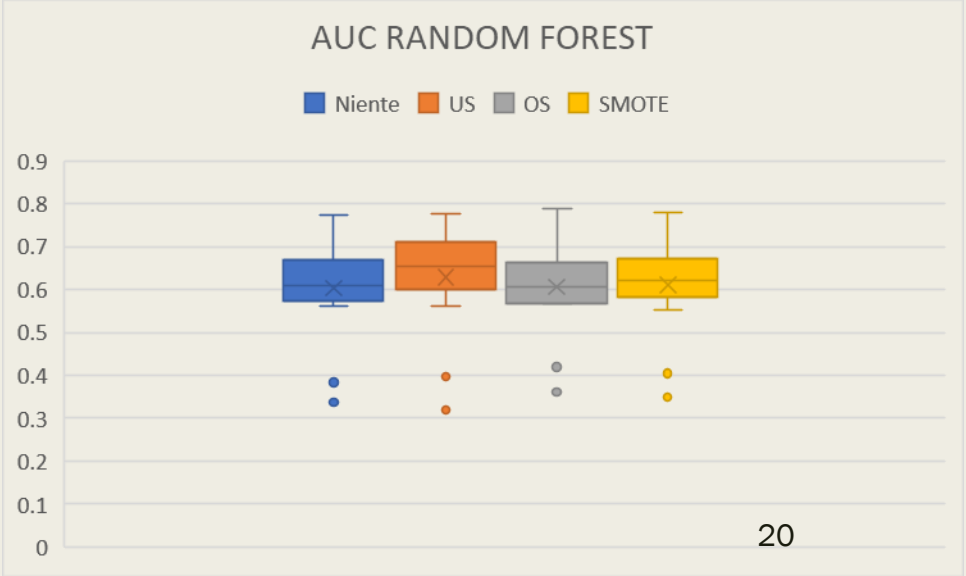


Risultati STORM- Area Under ROC

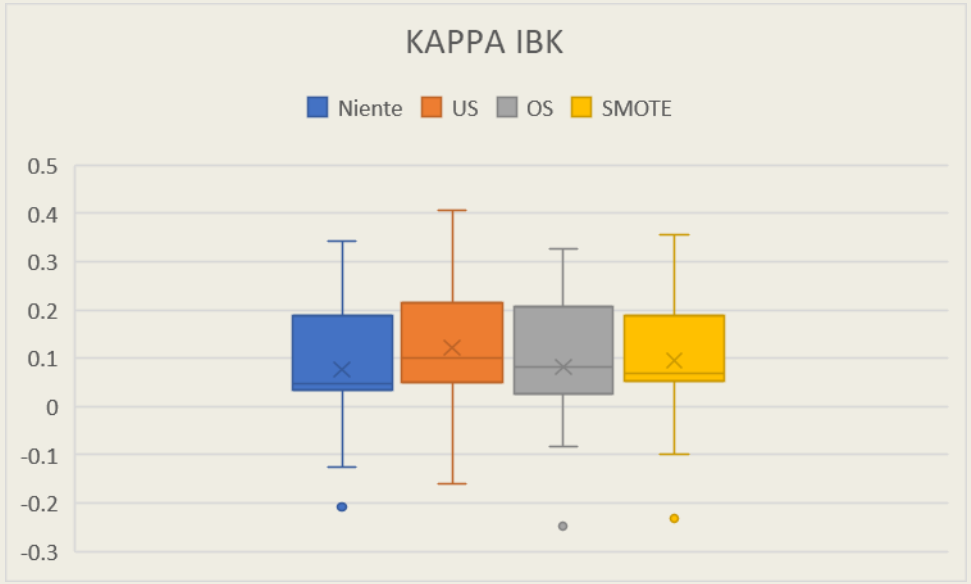
-In questo caso **non ci sono variazioni** davvero notevoli per nessun classificatore, indipendentemente dall'applicazione o meno di balancing.
Tra Oversampling e SMOTE nello specifico, **quest'ultimo** sembra produrre una media e mediana leggermente migliori per tutti e tre i classificatori

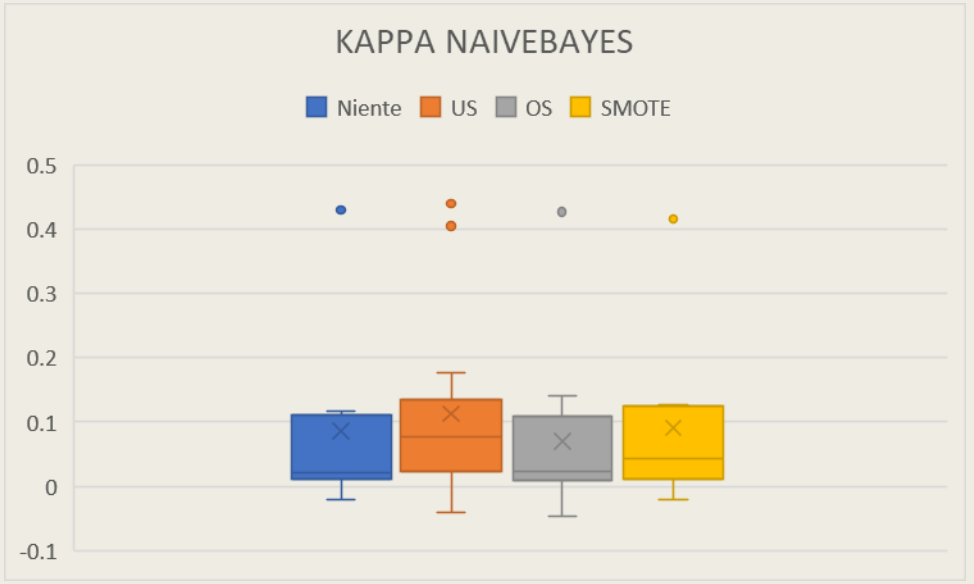


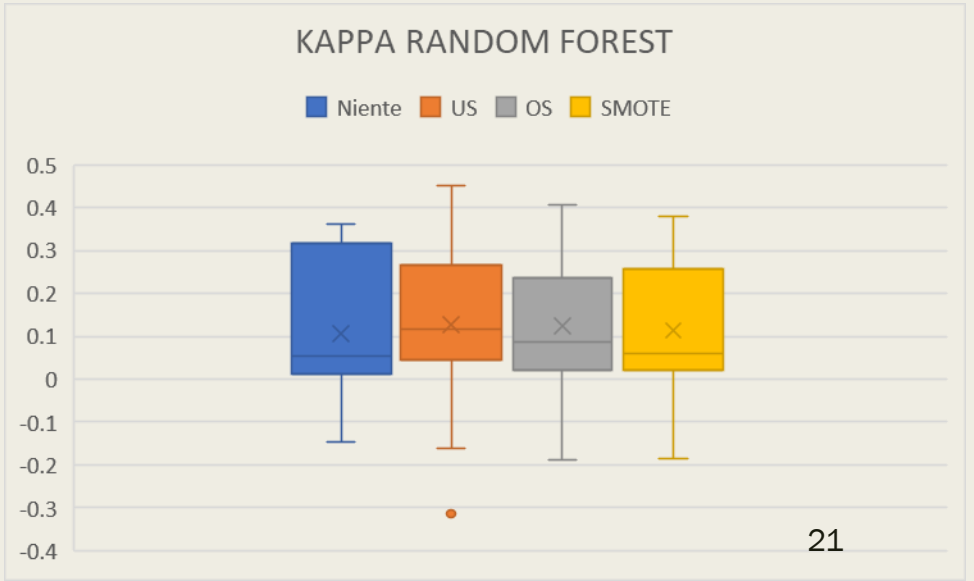




Risultati STORM- Kappa







Risultati STORM- Kappa

-Per il kappa, la tecnica dell' **undersampling** sembra produrre mediamente i risultati migliori: la media e la mediana sono, quando applicato l'undersampling, maggiori rispetto alle prestazioni del classificatore senza balancing, e questo per ogni classificatore

-L'undersampling **ha le prestazioni migliori** anche se confrontato con le altre due tecniche

-Tra SMOTE e Oversampling, il primo **migliora** maggiormente le prestazioni medie di ibk e NaiveBayes, mentre le **peggiora** nel caso Random Forest. La variabilità con SMOTE è leggermente minore rispetto al caso del classificatore senza balancing, per tutti e tre, ed è inferiore anche a quella del caso in cui viene applicato l'oversampling

Discussione Risultati- recap conclusivo

-**Tra l'undersampling e gli altri due approcci**, si hanno risultati discordanti. Per Bookkeeper l'undersampling è mediamente la tecnica peggiore peggiorando anche le prestazioni del classificatore senza balancing applicato, tranne un paio di casi specifici- come la precision di NaiveBayes

-**Nel caso di storm**, invece, **l'undersampling è** quasi sempre **migliore** rispetto agli altri due. Eccetto per la precision, migliora le prestazioni di ciscun classificatore

-**Tra SMOTE e Oversampling**: per bookkeeper l'oversampling risulta migliore con valori medi maggiori e variabilità equiparabili.

-per Storm avviene l'esatto opposto: **smote migliora le prestazioni** più di quanto non faccia l'oversampling con una variabilità minore

-Nel confronto bisogna anche tener conto della **limitatezza** nel complesso del dataset di bookkeeper rispetto a quello Storm

Conclusioni

- **I risultati** evidenziati dai due progetti **sono discordanti**, quindi non è possibile definire una tecnica di balancing in assoluto migliore rispetto all'altra

-Lo stesso vale per la scelta specifica tra smote e oversampling

-Sicuramente una delle motivazioni principali di tali differenze è data dalla dimensione e qualità del dataset che fanno molta differenza nella determinazione di soluzioni ottime per i casi specifici

-Mediamente **Random Forest** è il classificatore con i numeri migliori. Sembrerebbe la scelta migliore, tuttavia questo dipende dallo scopo: se per esempio si preferisse avere il maggior numero di istanze positive identificate, indipendentemente dai falsi positivi, allora Naive Bayes sarebbe la scelta migliore

-Le scelte apparentemente **migliori** per questo caso di studio, con i dati a disposizione: per bookkeeper sono RandomForest con undersampling, per Storm RandomForest con SMOTE

Grazie per l'attenzione

Link:

DataCollector github: https://github.com/MegAtonBoom/DataCollector

DataCollector sonarcloud: https://sonarcloud.io/project/overview?id=MegAtonBoom_MLtoISW

MLtoISW github: https://github.com/MegAtonBoom/MLtoISW

MLtoISW sonarcloud: https://sonarcloud.io/project/overview?id=MegAtonBoom_MLtoISW2