Spolaor Riccardo (864877)
Web Intelligence [CT0428]

Who will win the Australian Open 2020 Tournament?

Università
Ca'Foscari
Venezia

Introduzione

Il progetto si prefigge di predire l'**outcome** del torneo di tennis **Australian Open 2020** definendo un modello di predizione a partire dai dati delle singole partite forniti dal sito http://www.tennis-data.co.uk/ e simulando la contesa match per match.
Il dataset inizialmente proposto conteneva per ogni match informazioni riguardo le caratteristiche tecniche del torneo e del suo sviluppo, le valutazioni dei bookmaker e le statistiche agonistiche dei giocatori.
Ho deciso di impostare il lavoro in parti distinte, dividendo i singoli processi di analisi in vari Notebook affinché fosse chiara la visualizzazione. Sono stati creati dei file Python separati all'interno del folder *./python files* per gestire funzioni richiamate più volte o utilizzate in Notebook differenti o che contengono molte righe di codice.
Il dataset è stato successivamente implementato tramite l'aggiunta di nuove feature come dettagliatamente specificato al punto *05 Adding New Features.*

00 Dataset Download and Examination

Il sito http://www.tennis-data.co.uk/ contiene i dati dei match suddivisi per anno all'interno di file nel formato *.csv*, in ognuno di essi i tornei sono ulteriormente ordinati per data. Ho ignorato il file riguardante l'anno 2000 in quanto non conteneva informazioni relative agli odd dei bookmaker, dato ritenuto fondamentale per la predizione.
Ho importato i singoli file in ordine di data in oggetti *DataFrame* della libreria *pandas* scaricandoli direttamente da internet e aggiungendo una feature `csvID` progressiva per distinguerli. Ho poi unito ogni *DataFrame*, mantenendo l'ordine temporale dei match.

In seguito ho svolto un lavoro preliminare di **data pre-processing**, riscontrando che alcune feature contenenti valori numerici erano state assegnate erroneamente al tipo *object* e sono intervenuto correggendo le anormalità.
Dando seguito al processo di analisi ho rilevato la presenza di dati sporchi per le feature `Winner` e `Loser`, che ho corretto eliminando gli spazi vuoti riscontrati prima e dopo i nomi, trasformando tutte le stringhe in maiuscolo e procedendo ad altre correzioni manuali.

Ho infine realizzato i grafici delle occorrenze dei valori per ogni feature categoriale per capire come potessero essere successivamente gestite.

01 Data Cleaning – Missing Values Filling

Il passo successivo è stato quello di riempire i **valori mancanti** del dataset. Le feature con tali caratteristiche erano quelle del Rank del vincitore e del perdente (`WRank, LRank`), dei loro punti (`WPts, LPts`), degli odd dei bookmaker e dei game e i set vinti dai contendenti (`W1, L1, Wsets, Lsets…`).
Dove sono stati inferiti i missing values, eccetto che per i game ed i set vinti dai giocatori, è stata aggiunta una nuova feature contenente valori binari per indicare la precedente mancanza del dato.
Da un'analisi delle statistiche ho riscontrato che `MaxL` conteneva un valore massimo anomalo e la sua deviazione standard era molto alta. Ho quindi modificato i valori anomali con *NaN* in modo da poterli riempire con dati più verosimili nei passi successivi.

Inizialmente avevo deciso di riempire i valori del rank e dei punti dei giocatori utilizzando la media del valore di quella feature e di eliminare ogni riga rimanente che contenesse ancora dati *NaN*. Successivamente ho considerato di riempire i valori rimanenti dopo il primo filling con il dato del rank massimo + 1 e quello dei punti minimi – 1, ipotizzando che il giocatore in questione fosse una new entry quando non aveva rank o punti assegnati.

Visto che molte odd dei singoli bookmaker mancavano, ho deciso di considerare solo le feature che specificavano la scommessa massima e media dei bookmaker (`MaxW` e `AvgW`).
Avevo deciso di riempire i missing values, quando possibile, sfruttando i valori delle scommesse dei singoli bookmaker per quel giocatore e per quella specifica partita per poi eliminare le righe che contenevano ancora dati *NaN*. Ho in seguito deciso, in sostituzione all'eliminazione, di inserire un ulteriore filling che sfruttasse la media dei valori di quella feature per ogni match.

Ho infine utilizzato il numero 0 al posto dei valori mancanti dei game e dei set vinti dal giocatore ad indicare che non ne aveva vinto nessuno. L'unico dato non presente dei set vinti dal vincitore di una partita completata (secondo la feature `Comment`) era di un match al meglio di 5 ed è stato riempito con il valore 3.

Ho ritenuto in un secondo momento di non eliminare nessuna riga, ma di procedere al filling di tutti i valori mancanti in quanto l'accuracy delle previsioni, testata nei passi successivi mantenendo il dataset con tutti i match, risultava maggiore rispetto a quella ottenuta attraverso l'eliminazione di alcune righe.

02 Data Cleaning – Categorical Data Transformation

In questo passo del processo di analisi ho gestito la **trasformazione delle variabili categoriali** affinché fossero utilizzabili dai modelli di previsione, senza incorrere in particolari criticità.

Dummy Features
Ho trasformato in binaria l'unica feature contenente due soli valori:
- `Court`.

One-Hot-Encoding
Ho utilizzato la seguente tecnica per trasformare feature categoriali n-arie prive di relazioni gerarchiche:
- `Surface; Comment`.

One-Hot-Encoding Parziale
Visto che trasformare le colonne `Winner` e `Loser` con lo one-hot-encoding avrebbe allargato troppo il dataset, dato il numero elevato di valori, ho deciso di ordinare i giocatori in base alla differenza tra vittorie e sconfitte considerando i migliori 25. Ho poi creato per ognuno di essi due feature binarie per indicare se il giocatore A (`Winner`) e il giocatore B (`Loser`) fossero congruenti. Ho poi aggiunto le colonne `OtherA` e `OtherB` qualora vincitore o perdente non rientrassero tra i 25 migliori giocatori.

Grading
Grazie anche all'analisi delle occorrenze delle variabili categoriali fatta al passo *00 Dataset Download and Examination*, ho assegnato un valore crescente in base all'ordine delle seguenti variabili:
- `Round`;
- `Series` (in base all'importanza del tipo di Torneo);
- `Location` (in base al numero di match giocati in quel luogo).

Grading Parziale:
La feature `Tournament` presentava un numero di valori troppo elevato per utilizzare la tecnica del one-hot-encoding o un grading totale, ho quindi ordinato solo i 25 tornei più importanti classificandoli in base al loro prestigio da 1 a 25 e destinando valore 0 a tutti gli altri. Le fonti utilizzate sono le seguenti:
- https://www.tennis365.com/tennis-top-10/top-10-atp-tour-500-series-events-where-does-queens-club-rank/
- https://www.tennis365.com/tennis-top-10/top-10-biggest-non-grand-slam-tournaments-on-the-tennis-tour/
- https://www.forbes.com/sites/monteburke/2012/05/30/what-is-the-most-prestigious-grand-slam-tennis-tournament/#2370e5097696

03 Building The Prediction Dataset

Successivamente ho adeguato il dataset per costruire il modello di predizione. Ogni entry era costruita in modo da individuare il vincitore ed il perdente di ogni match. Ho rinominato `Winner` in `PlayerA` e `Loser` in `PlayerB` e le feature a loro collegate di conseguenza. Ho aggiunto una nuova feature binaria (`Winner`) da utilizzare come previsore per l'output che assumesse valore 0 se la partita fosse stata vinta da `PlayerA` e 1 se la partita fosse stata vinta da `PlayerB`. Per come era costruito il dataset, ogni valore della nuova feature `Winner` è stato inizializzato a 0. Per poter allenare un modello è stato necessario invertire le informazioni dei giocatori di alcuni match per ottenere alcuni valori che indicassero la vittoria del `PlayerB` (`Winner` = 1).

Per un dataset bilanciato era necessario avere un numero equo di righe con valori 0 e 1 per `Winner`. Un'opzione considerata è stata quella di duplicare il dataset, invertire le righe della copia e riunire i due dataset mantenendo l'ordine temporale dei match.
Per una questione computazionale, dato che la cardinalità dell'insieme delle entry del dataset era molto elevata già in principio, ho deciso di invertire solo le righe pari, in modo da avere comunque un modello bilanciato, ma non esageratamente grande, visto che non c'erano variazioni di attendibilità rispetto alle prove effettuate anche con la prima tecnica descritta.

Ho poi eliminato le feature che contenevano dati non utilizzabili per la previsione, in quanto indicavano le statistiche del match o non erano a mio avviso significative:
- `Date; PlayerA; PlayerB; 1A; 1B; 2A; 2B; 3A; 3B; 4A; 4B; 5A; 5B; setsA; setsB; Awarded; Completed; Disqualified; Retired; Sched; Walkover`.

04 Validating The Prediction Models

Ho quindi proceduto ad una verifica primaria dell'accuratezza tramite due modelli di previsione: ***decision tree classifier*** e ***random forest classifier***. La divisione del dataset in **train** e **validation** è stata gestita in modo da mantenere l'ordine temporale dei dati. Il training dataset è stato costruito con i primi ⅔ dei dati ed il test dataset con i rimanenti.
Non è stato necessario applicare tecniche di bilanciamento quali smote oversampling in quanto le classi erano bilanciate per come le avevo costruite al passo precedente.
Prima di procedere alla vera e propria costruzione e validazione dei modelli ho ricercato da quale anno in poi (`csvID`) l'accuracy fosse migliore per un decision tree classifier a cui è stato applicato un tuning sulla profondità. È risultato che l'accuracy massima fosse migliore per i dati dal 2002 in poi:

```
Best Max csvID: 1 - Accuracy: 0.6928549334982987
```

Per gestire il tuning degli **iperparametri** dei modelli ho deciso di non applicare una esaustiva grid search per motivi computazionali, ma di gestire il tuning di ognuno di essi singolarmente.
Il processo di scelta degli iperparametri è stato eseguito a tentativi ed ho notato che, per ognuno dei modelli, dopo il tuning di un paio di essi l'accuracy non migliorava più in modo significativo anche aggiungendo ulteriori aggiustamenti agli iperparametri.

Decision Tree Classifier

Gli iperparametri considerati per il tuning dell'***albero classificatore*** sono stati:

- `max_depth; max_features` (Vedere *Allegato 1*).

Dall'analisi della **decomposizione della varianza e distorsione quadratica** delle previsioni del modello basata sulla profondità dell'albero è visibile come, all'aumentare di questa, l'albero faccia **overfitting**, diminuendo la distorsione, ma aumentando la varianza (Vedere *Allegato 2*).

```
Best Max Depth: 5 - Accuracy: 0.6928549334982987
Best Max Features: 33 - Accuracy: 0.6931642437364677
```

Random Forest Classifier

Gli iperparametri considerati per il tuning della ***random forest di classificazione*** sono stati:

- `n_estimators; max_depth` (Vedere *Allegato 3*).

Dall'analisi della **decomposizione della varianza e distorsione quadratica** delle previsioni del modello basate solo sul numero di stimatori della foresta è visibile come, all'aumentare del numero di essi, la varianza diminuisca e la distorsione rimanga stabile. Una random forest con 275 stimatori presenta un errore più basso rispetto ad un albero fully grown, ma più alto di quello di un albero con il tuning, in quanto riesce a mantenere bassa la distorsione, ma non riesce a gestire in modo altrettanto efficiente la varianza, aumentando conseguentemente l'errore totale.
Considerando anche la profondità della foresta si evidenzia che, all'aumentare di questa, la varianza cresce e la distorsione diminuisce. Una random forest con 275 stimatori e profondità 10 predice i risultati con variazioni minimali positive rispetto ad un albero con il tuning degli iperparametri applicato (Vedere *Allegato 4* e *Allegato 5*).

```
Best Estimators Number 275 - Accuracy: 0.6823383854005568
Best Depth 10 - Accuracy: 0.694463346736777
```

05 Adding New Features

Ho ripreso il dataset precedente al punto *03 Building The Prediction Model* per aggiungere nuove feature ritenute importanti per aumentare la qualità della previsione.

Feature basate su quelle fornite dal Dataset

Sono state aggiunte le seguenti feature:

- `OpponentsPlayed`: Il numero di partite disputate tra i due contendenti prima del match;
- `OpponentsWRatio(A/B)`: Il rapporto tra partite vinte da A o B e giocate tra i due contendenti prima del match;
- `FadigueTournGames(A/B)`: Il numero di game giocati da A o B nell'attuale torneo prima del match, come misura di stanchezza.
- `FadigueTournSets(A/B)`: Il numero di set giocati da A o B nell'attuale torneo prima del match, come misura di stanchezza.
- `WalkoverLast(A/B)`: feature binaria per indicare se il giocatore aveva subito un infortunio l'ultimo match (`Walkover` = 1)
- `RetiredLast(A/B)`: feature binaria per indicare se il giocatore si era ritirato l'ultimo match (`Retired` = 1)
- `Played(A/B)`: Il numero di partite giocate dal giocatore A o B prima di quel match
- `WonRatio(A/B)`: Il rapporto tra partite vinte dal giocatore A o B e giocate prima del match
- `PlayedCourt(A/B)`: Il numero di partite giocate da A o B nello stesso tipo di campo del match prima della partita;

- `WonRatioCourt(A/B)`: Il rapporto tra partite vinte da A o B e giocate prima della partita nello stesso tipo di campo del match;
- `5_setsMean(A/B)`: La media del numero di game vinti dal giocatore A o B nei 5 match precedenti alla partita;
- `5_gamesMean(A/B)`: La media del numero di set vinti dal giocatore A o B nei 5 match precedenti alla partita;

Data Integration
Ho deciso di integrare i dati aggiungendo una feature che indicasse la mano con cui gioca l'atleta A (`Winner`) `HandA`, e quella con cui gioca l'atleta B (`Loser`) `HandB`.
Le informazioni sono state recuperate dal dataset presente al seguente link:
- https://www.kaggle.com/sijovm/atpdata

Essendoci informazioni mancanti per alcuni giocatori, le ho recuperate presso i seguenti link:
- https://www.ultimatetennisstatistics.com/
- https://www.tennisexplorer.com/

È stato necessario un lavoro di **data pre-processing** per rendere i nomi dei giocatori nello stesso formato.
Oltre alla mano con cui giocano i contendenti sono state aggiunte le seguenti feature:
- `PlayedVsSameHanded(A/B)`: Il numero di partite disputate dal giocatore A o B contro giocatori che usano la stessa mano dell'avversario prima del match;
- `WonRatioVsSameHanded(A/B)`: Il rapporto tra partite vinte dal giocatore A o B e giocate contro giocatori che usano la stessa mano dell'avversario prima del match;

Feature di Confronto
Sono state infine aggiunte delle feature binarie per esprimere un raffronto quantitativo tra le caratteristiche del giocatore A e quelle del giocatore B (`"Caratteristica"A>"Caratteristica"B`).

Ho aggiunto le nuove feature riga per riga, sfruttando il fatto che le partite sono ordinate in base alla data. Utilizzando il metodo *`itertuples()`*, che itera tra le righe come `namedtuples`, sono riuscito ad ottenere tempi di computazione migliori rispetto a quelli risultanti utilizzando il metodo *`iterrows()`*.

---

06 Rebuilding The Prediction Dataset

Eseguo nuovamente le procedure in *03 Building The Prediction Model* tenendo conto delle nuove variabili aggiunte.

---

07 Revalidating The Prediction Models

Ho ritestato l'accuracy delle previsioni utilizzando tre modelli: ***decision tree classifier***; ***AdaBoost classifier*** e ***random forest classifier***. Le considerazioni sulla divisione del dataset in **train** e **validation** e sul metodo di tuning degli iperparametri è stato lo stesso del punto *04 Validating The Prediction Model* con l'aggiunta di una breve **feature relevance analysis** prima della vera e propria **validation**. È inoltre risultata più precisa l'accuracy considerando i dati di tutti gli anni:
`Best Max csvID: 0 - Accuracy: 0.6944347217360868`

Feature Relevance Analysis
Ho verificato l'accuratezza delle predizioni di un albero con applicato il tuning alla profondità sia per il dataset con tutte le feature che per quello senza le feature con le caratteristiche specifiche del giocatore A e il giocatore B contenente solo quelle di confronto tra le caratteristiche dei due giocatori (`caratteristicaA>caratteristicaB`). Ho ritenuto opportuno applicare questo tipo di analisi in quanto, a mio avviso, i modelli che sfruttano alberi di decisione raggiungono risultati di maggior precisione considerando solo confronti binari.
Ho applicato il dataset con l'eliminazione delle feature in quanto risulta avere un'accuracy migliore (Vedere *Allegato 6*).
Dataset con tutte le feature: `Best Depth: 5 - Accuracy: 0.6922762804806907`
Dataset considerando solo le feature di confronto: `Best Depth: 3 - Accuracy: 0.6944347217360868`

Decision Tree Classifier
Gli iperparametri considerati per il tuning dell'***albero classificatore*** sono stati:
- `max_depth; max_features` (Vedere *Allegato 7*).

Dall'analisi della **decomposizione della varianza e distorsione quadratica** delle previsioni del modello si evince come l'albero abbia un comportamento pari a quello osservato al punto *04 Validating The Prediction Model*
(Vedere *Allegato 8*).
L'aggiunta delle nuove feature non ha migliorato sensibilmente l'accuratezza rispetto ai modelli costruiti al passo *04 Validating The Prediction Model.* Questo significa che le feature aggiunte sono poco rilevanti per la predizione.
`Best Max Depth: 3 - Accuracy: 0.6944347217360868`
`Best Max Features: 90 - Accuracy: 0.6944347217360868`

AdaBoost Classifier

Ho deciso di non procedere al Bagging dell'albero in quanto, dall'analisi della decomposizione della varianza e distorsione quadratica delle sue previsioni, la distorsione risulta essere la maggiore fonte di errore. Per provare a migliorarla ho utilizzato la tecnica di ***AdaBoost***.
L'iperparametro considerato per il tuning dell'***AdaBoost Classifier*** è stato `n_estimators` e ho deciso di dare in pasto al modello l'albero con il tuning degli iperparametri descritto al punto precedente, in quanto AdaBoost preferisce **small tree** con piccola varianza per diminuire la distorsione (Vedere *Allegato 9*).
L'analisi della **decomposizione della varianza e distorsione quadratica** mostra come all'aumentare del numero di stimatori, la distorsione si abbassi di poco, ma la varianza cresca aumentando l'errore totale (Vedere *Allegato 10*).
L'accuratezza del modello risulta essere pari rispetto a quella dell'albero con il tuning degli iperparametri.
`Best Number of Estimators: 2 - Accuracy: 0.6944347217360868`

Random Forest Classifier

Gli **iperparametri** considerati per il tuning della ***random forest di classificazione*** sono stati:

- `n_estimators; max_depth` (Vedere *Allegato 11*).

L'analisi della **decomposizione della varianza e distorsione quadratica** delle previsioni del modello evidenzia che il modello ha un comportamento pari a quello della random forest al punto *04 Validating The Prediction Model* (Vedere *Allegato 12* e *Allegato 13*).
Una random forest con 225 stimatori e profondità 8 raggiunge un'accuratezza minimamente inferiore rispetto ad un albero con applicato il tuning ed è possibile di conseguenza considerare le previsioni similari tra i due modelli.
`Best Estimators Number 225 - Accuracy: 0.6629914829074787`
`Best Depth 8 - Accuracy: 0.6942597129856493`

.

Recursive Feature Elimination

Applicando la tecnica del **recursive feature elimination** alla random forest con il tuning degli iperparametri descritta al paragrafo precedente ho ottenuto la metà delle feature più significative per la predizione.
Le predizioni di una Random Forest con il tuning nuovamente applicato a `n_estimators` e `max_depth` ha portato ad una leggera riduzione dell'accuratezza del modello, ma non troppo significativa (Vedere *Allegato 14*).
`Best Estimators Number 275 - Accuracy: 0.6602496791506242`
`Best Depth 5 - Accuracy: 0.6938513592346284`

Ho deciso di applicare quest'ultimo modello per prevedere i risultati del torneo in quanto utilizza un numero di parametri inferiore agli altri, garantendo una complessità computazionale più bassa ed un'accuratezza non troppo ridotta rispetto ai modelli che utilizzavano tutte le feature.

Inspection of Prediction Results

I risultati dell'ispezione fatta sulle predizioni del modello ottenuto tramite la recursive feature elimination evidenziano come l'accuratezza aumenti prevedendo partite dove entrambi i giocatori sono tra i migliori 25 e riesce ad essere ancora più efficace nella previsione dei match in cui almeno uno dei due giocatori è uno tra i migliori 25.

- `Accuracy on matches played by the most important players: 0.7263888888888889`
- `Accuracy on matches played by the least important players: 0.6627697841726619`
- `Accuracy on matches played by at least one important player: 0.7712830957230142`

---

08 Prediction Simulation

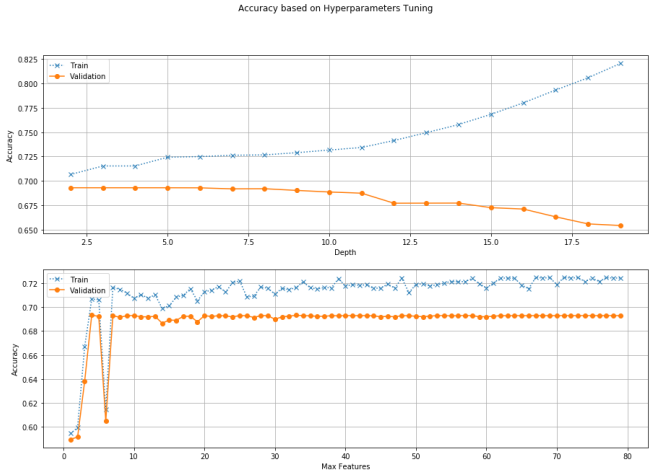
A questo punto ho selezionato i giocatori ammessi al torneo **Australian Open 2020** ed ho simulato i singoli match di ogni round considerando coppie casuali di tennisti. Dopo il risultato simulato di ogni match ho eliminato dalla lista di giocatori il perdente, prevedendo così la vittoria di **Murray A.** ed il secondo posto di **Federer R.**

Le feature di ogni match sono state inserite utilizzando lo storico delle partite precedenti.
Il modello di previsione andrebbe utilizzato durante lo svolgimento del torneo inserendo di volta in volta i dati relativi alla stanchezza in base al numero di game e set giocati e la media dei game e set vinti delle ultime cinque partite. Non ho utilizzato le feature relative al confronto della stanchezza dei due giocatori non avendo a disposizione dati reali.
Ho inserito comunque i dati relativi al confronto della media dei set e game vinti considerando gli ultimi cinque match precedenti l'inizio del torneo.
Ho inserito la comparazione tra gli odd medi e quella tra gli odd massimi dei bookmaker applicando la media dei precedenti per ogni giocatore, non potendo accedere ad i dati reali.
Il modello prevede di esplicitare il confronto tra i rank e quello tra i punti che ho inserito utilizzando il valore dell'ultimo Rank e quello dell'ultimo punteggio dei due contendenti.

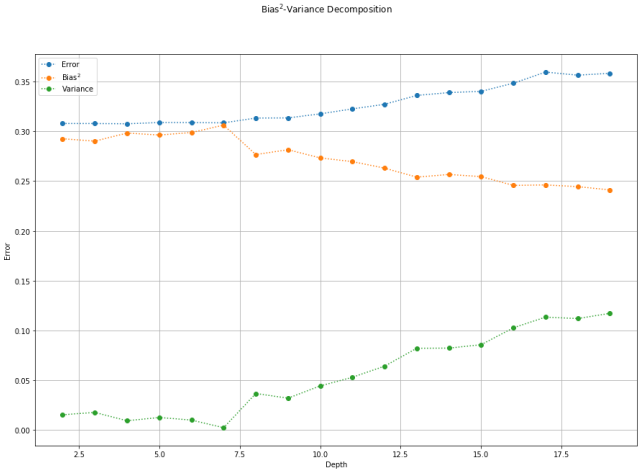
Ritengo che il modello costruito raggiunga una buona accuratezza pur nel limite dei dati utilizzati e nella semplicità della sua costruzione. Rimangono comunque numerose variabili imponderabili quali la stanchezza effettiva, lo stato fisico e la condizione psicologica dei giocatori, che possono vanificare la previsione.

Allegati

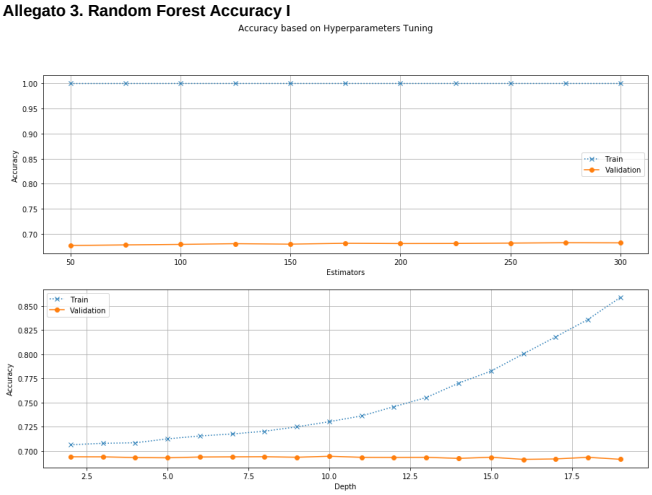
Allegato 1. Decision Tree Accuracy I



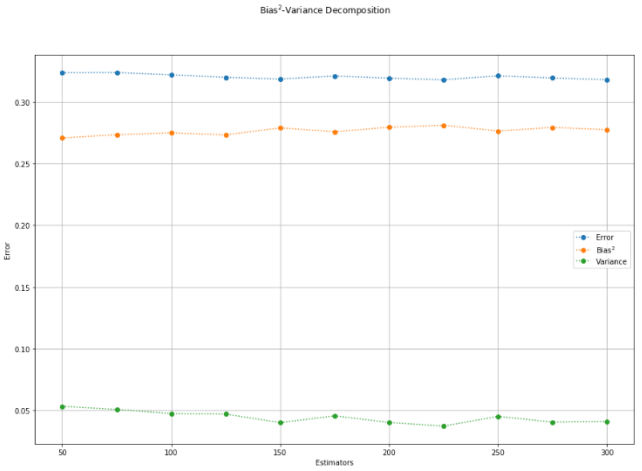
Allegato 2. Decision Tree Bias^2-Variance Decomposition I



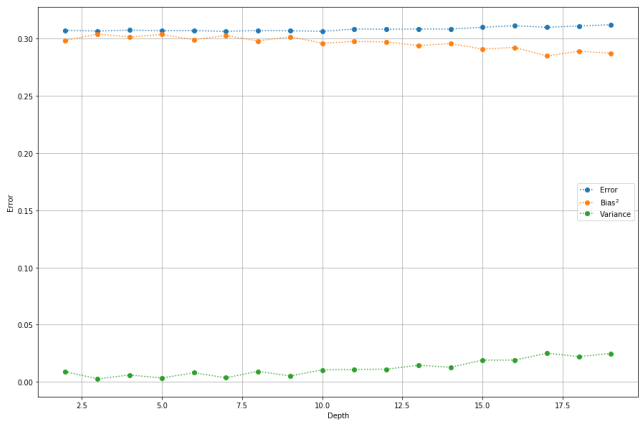
Allegato 3. Random Forest Accuracy I



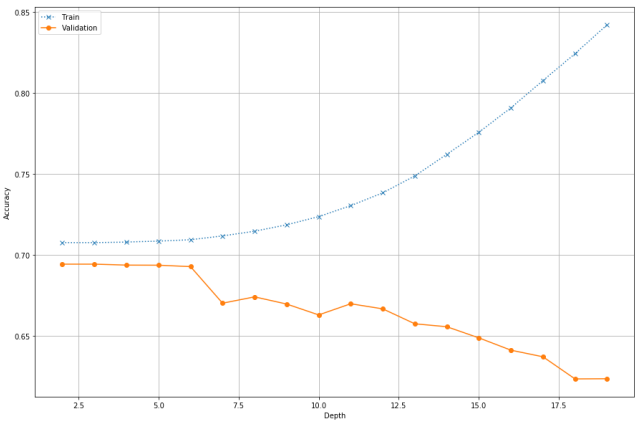
Allegato 4. Random Forest Bias^2-Variance Decomposition I-a



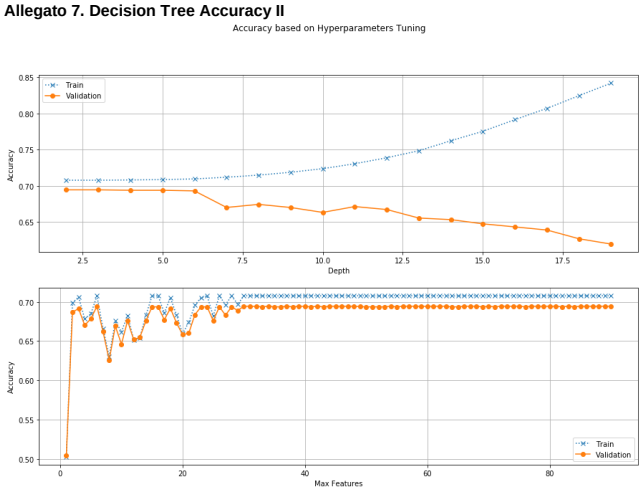
Allegato 5. Random Forest Bias^2-Variance Decomposition I-b



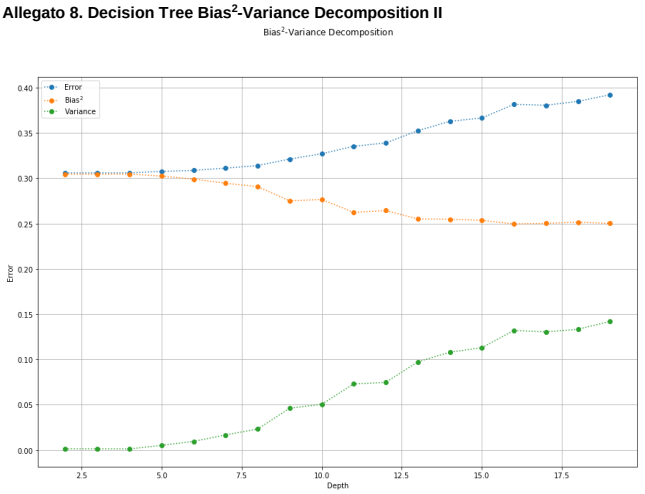
Allegato 6. Decision Tree Accuracy after Features Relevance Analysis



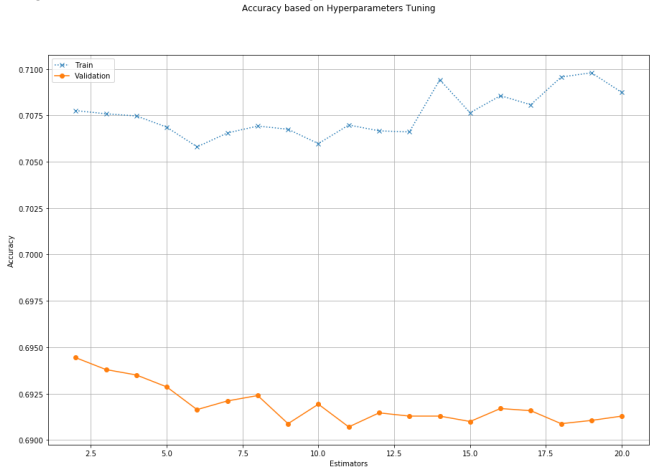
Allegato 7. Decision Tree Accuracy II



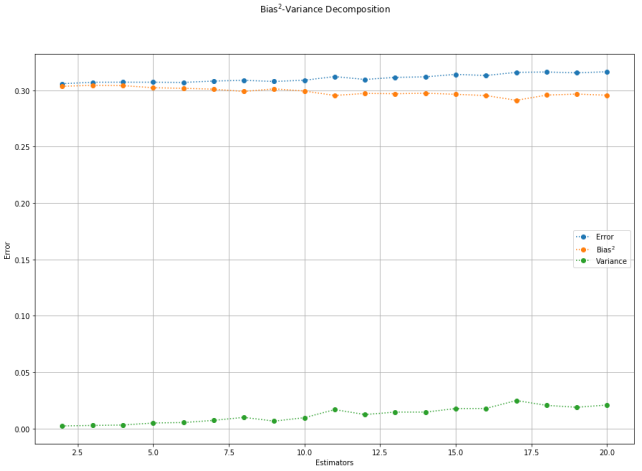
Allegato 8. Decision Tree Bias^2-Variance Decomposition II



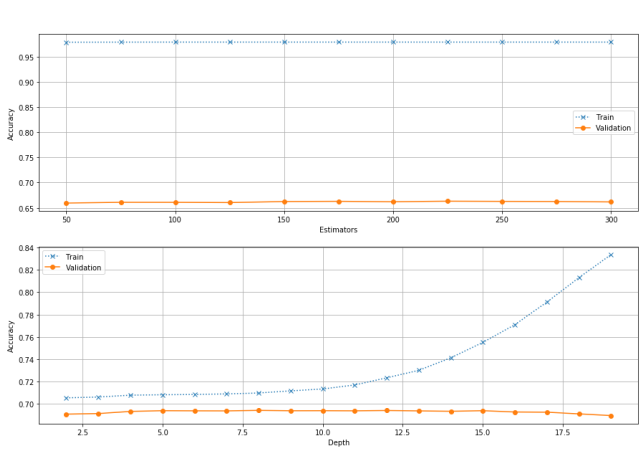
Allegato 9. AdaBoost Classifier Accuracy



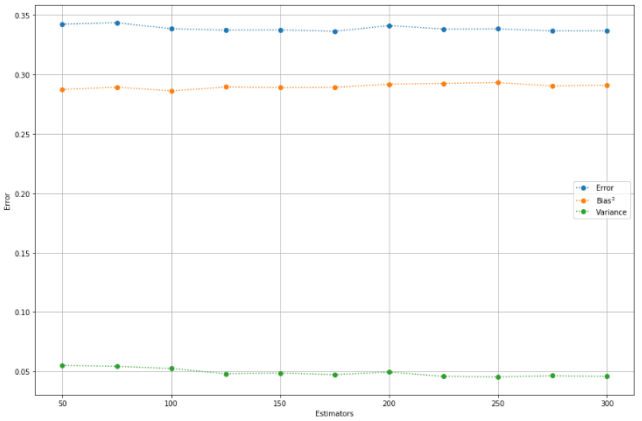
Allegato 10. AdaBoost Classifier Bias^2-Variance Decomposition



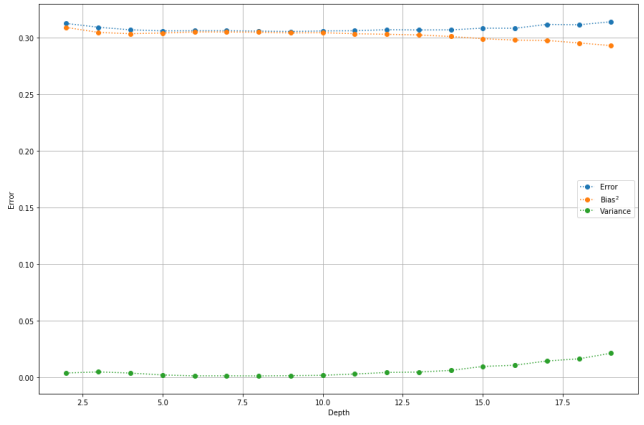
Allegato 11. Random Forest Accuracy II



Allegato 12. Random Forest Bias^2-Variance Decomposition II-a



Allegato 13. Random Forest Bias^2-Variance Decomposition II-b



Allegato 14. Random Forest Accuracy after Recursive Feature Elimination