SkillCraftI

Report per l'Esame di Fondameti di Machine Learning

Francesco Moretti

130301
Ingegneria informatica
260877@studenti.unimore.it

Luglio 2021

Abstract

Questo progetto è incentrato sulla classificazione dei giocatori di videogiochi sulla base delle loro abilità. L'obbiettivo è capire se è possibile creare un'implementazione che usi il machine learning per svolgere questo compito, e che vantaggi ha questa soluzione rispetto alle alternative attualmente usate correntemente. Per farlo costruiremo diversi modelli di machine learning e li confronteremo per capire quale si adatta meglio alle necessità di questo problema.

Indice

1 Introduzione

Il mondo dei videogiochi ha subito una rapida evoluzione negli ultimi anni, arrivando al punto che alcuni di questi possono essere già definiti come dei veri e propri Esports. Questo sviluppo ha contribuito a rendere questo settore sempre piu‘ competitivo e dinamico.

I giochi multiplayer spesso dividono i giocatori in leghe o gradi in modo tale da rendere le partite bilanciate, ovvero affinché ogni giocatore trovi un avversario che abbia un livello simile al suo. Per implementare questa soluzione sono solitamente usati dei sistemi a punti che premiano le vittorie e puniscono le sconfitte. Tutto ciò però non è spesso sufficiente a garantire il bilanciamento dei giochi, la prova di questo penso si possa trovare nel fenomeno detto smurfing, che sta spopolando in molti giochi, forse l'esempio più lampante si può trovare in Leagues of legends, dove perfino la casa produttrice si è trovata costretta a prendere misure a riguardo, come si può leggere nell'articolo scritto da Outplayed sul tema[2], ma comunque è pratica che sta diventando comune in diversi titoli videoludici.

Lo smurfing consiste nel creare un nuovo account con lo scopo di nascondere la reale identità del giocatore e usarlo al posto dell'account reale.
Le ragioni dietro questa pratica possono essere molteplici: da giocatori professionisti che vogliono usare un altro account per fare pratica e nascondere le proprie strategie, al comportamento ben più nocivo del "newb bashing", ovvero giocatori di livello medio-alto che si fanno un nuovo account per giocare contro i principianti e sembrare quindi dei fuoriclasse durante le partite.

Attraverso questo report cercherò di capire se esiste un metodo per dividere i giocatori in leghe migliore rispetto ai sistemi a punti o quantomeno che possa integrarsi ad essi per migliorarli.

Il videogioco su cui baserò la mia ricerca è StarCraft II, un gioco di strategia in tempo reale a tema fantascientifico nonché uno dei principali Esports. Starcraft II usa un sistema a punti per dividere i giocatori in 7 diversi rank dal piu‘ basso al piu‘ alto: Bronzo, Argento, Oro, Platino, Diamante, Master e Grand Master. Oltre a questi è stato aggiunto al dataset anche un ottavo rank, quello dei giocatori professionisti: questa modifica è stata effettuata perché non tutti i giocatori a Grand Master sono abbastanza forti per competere nei tornei più importanti.

Figure 1: Leghe di Starcraft

2 Analisi dei dati

In questa sezione saranno analizzati i dati di SkillCraftI [3] per cercare di capire la struttura dei dati e correggere eventuali problemi che il dataset potrebbe avere. Durante questa analisi saranno usate le librerie Python Matplotlib e Seaborn per graficare i dati e renderli più interpretabili.

2.1 Bilanciamento del dataset

Le leghe di Starcraft sono per loro natura sbilanciate. Dal sito Liquipedia [1], un'enciclopedia online a tema videogiochi, si può leggere come queste siano strutturate e gli obbiettivi che si sono posti gli sviluppatori quando le hanno create.

La distribuzione a cui puntavano gli sviluppatori è come segue:

- Grand Master 1000 giocatori
- Master 2%
- Diamante 18%
- Platino 20%
- Oro 20%
- Argento 20%
- Bronzo 20%

Nella realtà le cifre sono destinate ad essere leggermente diverse, ma la proporzione è a grandi line corretta. In più, oltre alle leghe del gioco, nel data set possiamo trovare anche i giocatori professionisti. Questa ulteriore divisione è stata aggiunta perché anche se i Grand master possono essere considerati un élite tra i giocatori, il divario tra le loro capacità e quelle dei professionisti è considerevole.
Le proporzioni del database Skillcraft invece sono indicate nel grafico a seguire 2.

Come previsto il dataset è sbilanciato, e ciò comporta una maggiore attenzione alle modalità di raggruppamento dei campioni durante l'addestramento dei modelli e nello splitting. Oltre a questo, l'unica altra anomalia che è possibile individuare è che le ultime tre leghe sono più numerose di quello che dovrebbero essere. Tale anomalia non è rispecchiata nella realtà ed è dovuto al modo in cui sono stati campionati i dati, ma ai fini della classificazione non dovrebbe creare grossi problemi.

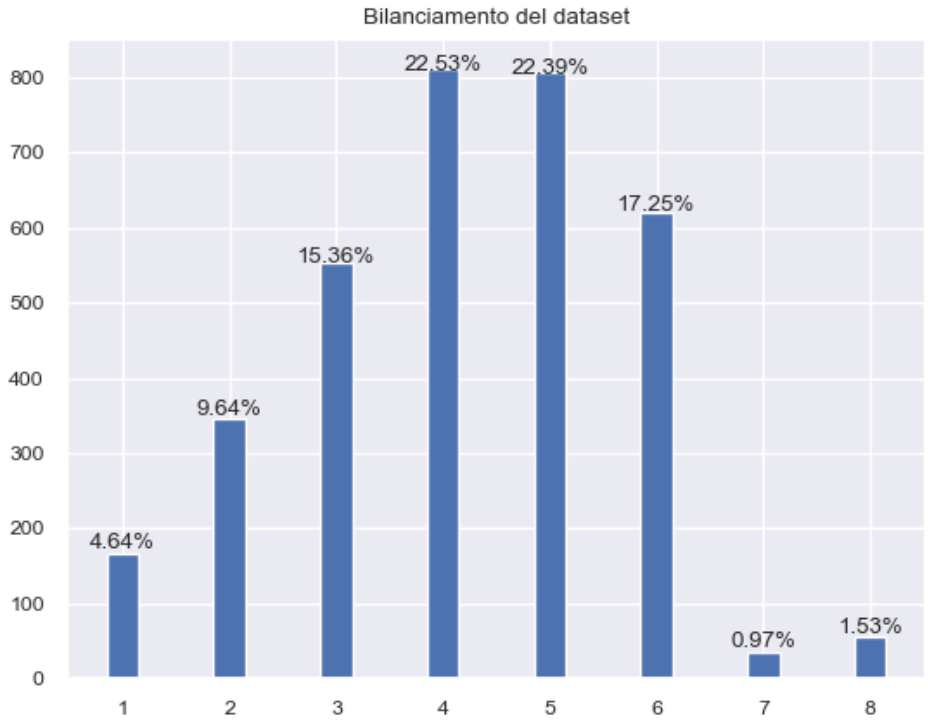

Figure 2: Distribuzione sample nelle leghe

2.2 Elementi nulli

Per la buona riuscita dell'addestramento è necessario rimuovere tutti gli elementi nulli, e il nostro dataset ne contiene alcuni. Ora cercheremo di capire quanti sono e come sono distribuiti per trovare una soluzione al problema.

Il grafico sotto rappresenta il numero totale di elementi nulli per ogni lega, divisi per la feature a cui appartengono.

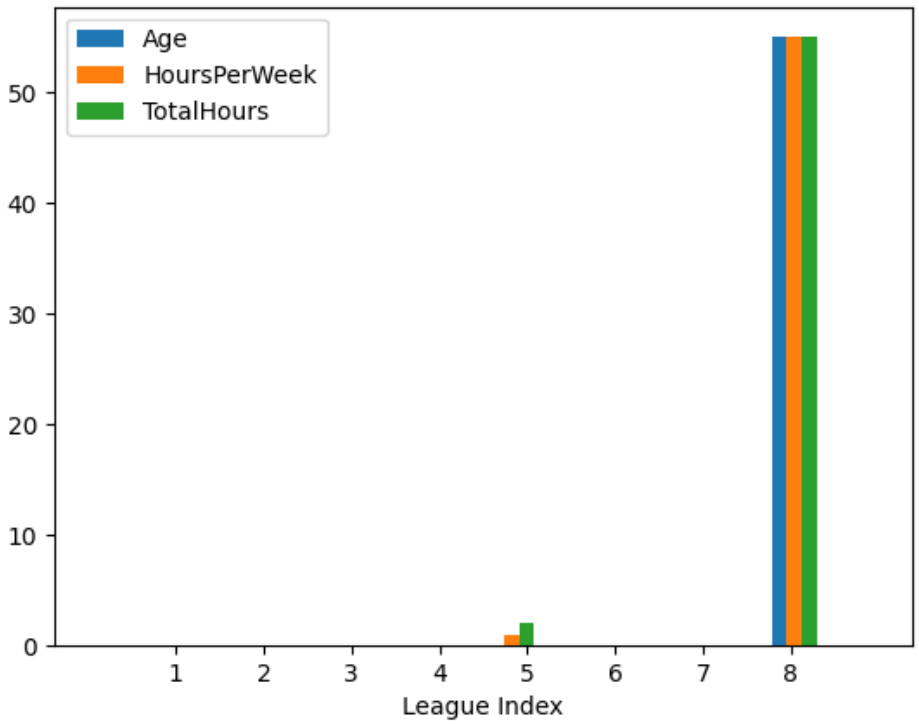

Figure 3: Distribuzione elementi nulli

Dal grafico si può vedere che gli elementi nulli sono concentrati in 3 features che rappresentano: Età, ore di gioco settimanali e ore di gioco totali. Inoltre, la maggior parte è inserito nella lega 8, ovvero i professionisti.

Esistono diverse soluzioni per gestire gli elementi nulli di un dataset, ad esempio si possono eliminare le righe che ne contengono uno, ma nel nostro caso, controllando i dati dei professionisti si può notare che le tre colonne identificate prima sono tutte nulle in questa lega, di conseguenza eliminare le righe con elementi nulli vorrebbe dire perdere completamente i dati sui professionisti.

Un'altra alternativa è quella di riempire i dati nulli con un valore che potrebbe stimarli, come la media degli altri valori di quella feature in quella lega, ma come visto prima non

ci sono altri valori su cui costruire la stima e di conseguenza questa strada è impraticabile.

L'unica alternativa rimasta è eliminare le colonne con i valori nulli, e per misurare l'effetto che questa decisione avrà sulla classificazione possiamo sfruttare una mappa di correlazione

La mappa di correlazione rappresenta le relazioni tra gli elementi su cui è costruita, da essa è possibile capire quanto due feature sono legate tra di loro e con la classe che vogliamo predire.

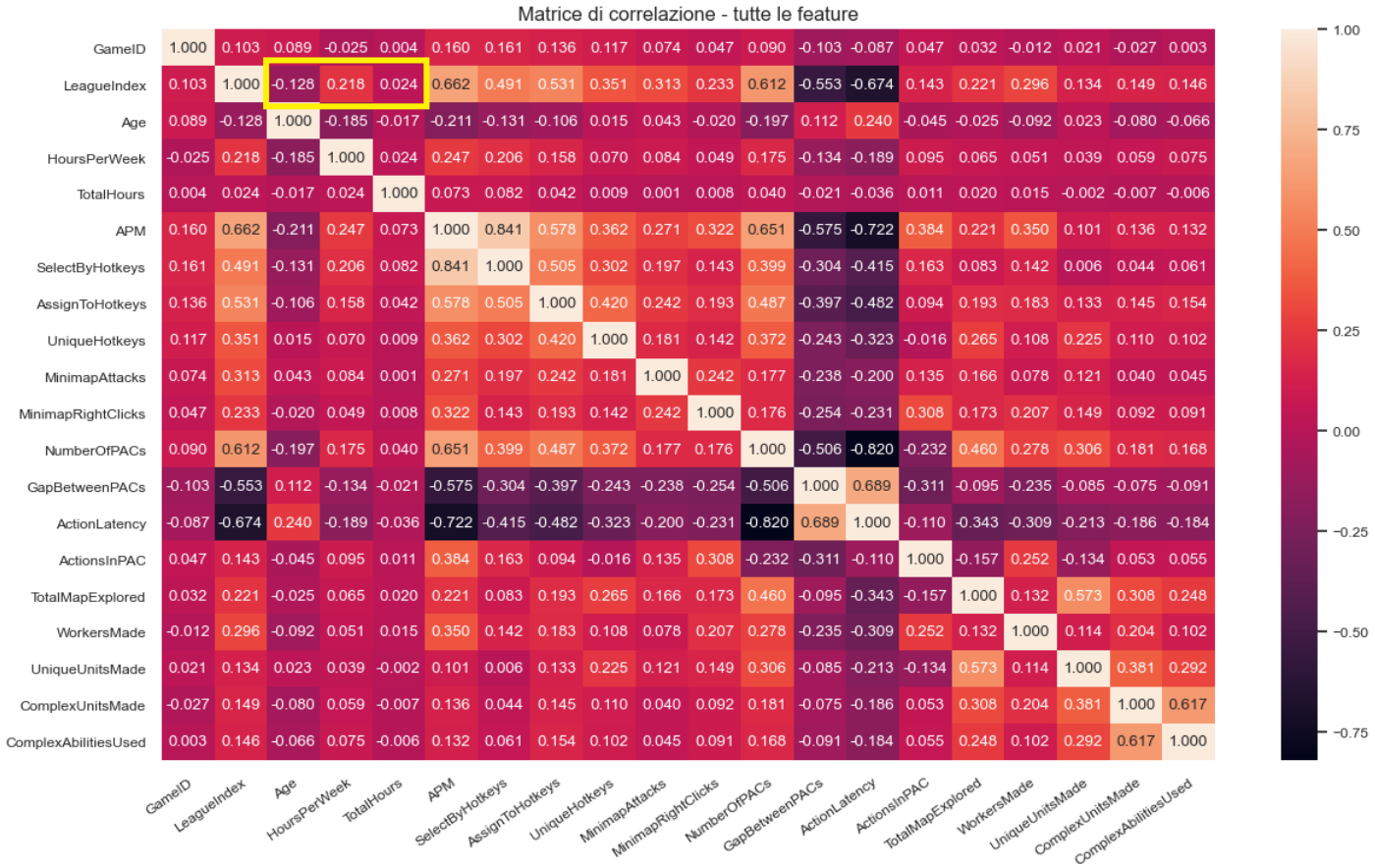

Figure 4: Mappa di correlazione

I valori evidenziati in giallo sono gli indici di correlazione delle feature che contengono i valori nulli, fortunatamente sono tendenzialmente bassi, quindi togliendoli non andremo ad influire sulla classificazione, ma potremmo addirittura migliorare le prestazioni di certi algoritmi di classificazione.

2.3 Relazioni tra le feature

A questo punto bisogna controllare come sono fatte le feature, che relazioni hanno tra di loro e in particolare con il target da classificare.

Per farlo useremo principalmente due grafici: il pairplot e la matrice di correlazione vista prima. Il primo ci serve a definire il tipo di relazione, mentre la seconda da un'indicazione più quantitativa della relazione.

Il pairplot può essere usato per guardare le relazioni tra tutte le feature, ma nel nostro caso è sconveniente vista l'elevata quantità di esse, perché i grafici verrebbero troppo piccoli e disordinati per essere comprensibili. Quindi ho deciso di fare un pairplot con solo le relazioni delle feature con il target, ovvero l'indice della lega.

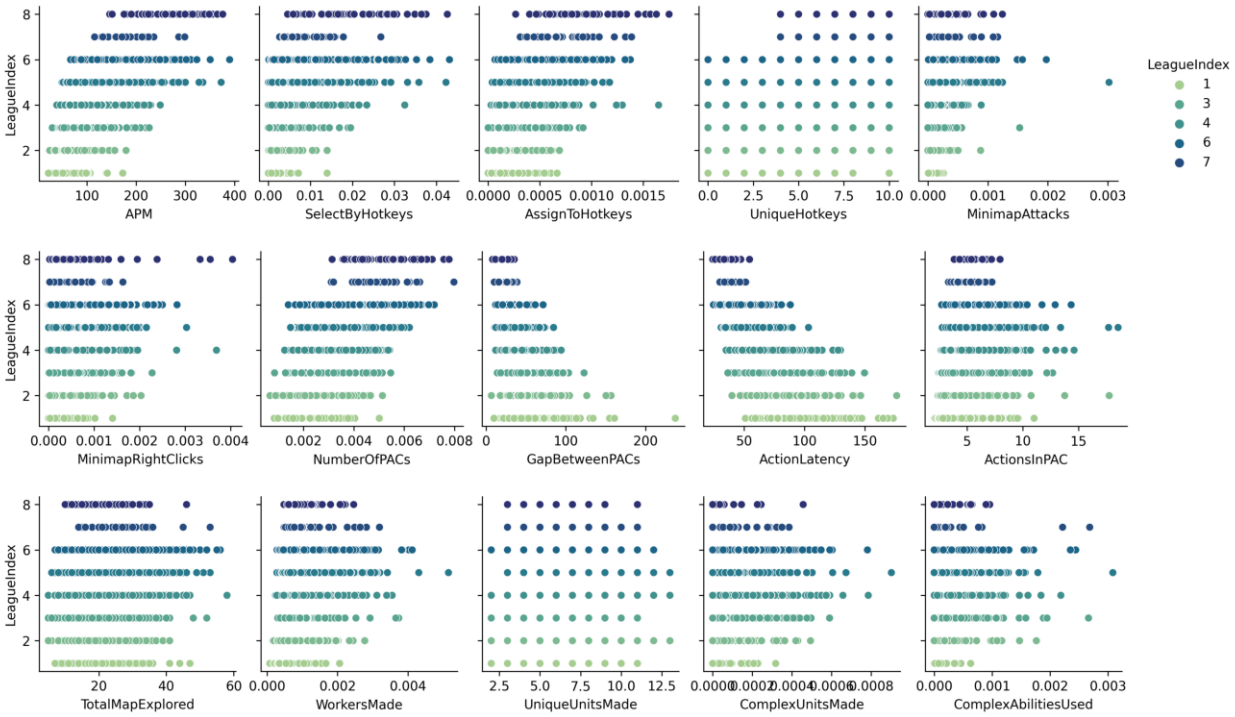

Figure 5: Pairplot delle feature con il target

Le cose che possiamo notare dai grafici sono due:

- La prima è che guardando il pairplot possiamo notare che alcune features hanno dei pattern nella loro relazione con il target, in particolare l'APM e l'ActionLatency, che sono anche le due feature con la correlazione maggiore, sembrano seguire un andamento vagamente lineare.
- La seconda riguarda la mappa di correlazione, dove possiamo notare che alcune features hanno una correlazione forte tra di loro, che è un indice di collinearità e multicollinearità. Questi fenomeni sono abbastanza pericolosi perché rendono i modelli suscettibili al rumore delle feature.

Analizzando questo il problema della multicollinearità a fondo però è possibile rendersi conto che queste feature, sebbene collegate, non sono l'una la diretta conseguenza dell'altra. Prendendo ad esempio le APM e il numero di selezioni fatte con tasti personalizzati, l'usare dei tasti personalizzati, se fatto male può portare anche al calo degli APM dovuto al tempo necessario per correggere gli errori, quindi l'abilità di fare queste operazioni correttamente è un buon indice della qualità di un giocatore.

Visto che anche gli altri casi sono simili a questo, ovvero anche se c'è correlazione il modo in cui sono correlati può essere significativo alla classificazione, ho deciso di lasciare inizialmente tutte le feature. Se poi durante l'analisi dei risultati avessi trovato la conferma che questo peggiora i risultati, allora sarei tornato indietro a avrei fatto una maggiore feature selection, ma non è stato così.

3 Discussione dei modelli

In questa sezione parleremo dei modelli che ho selezionato e del motivo di queste scelte, evidenziando i loro punti di forza e le eventuali debolezze.

Il nostro caso è un problema di classificazione con le seguenti caratteristiche:

- Multiclasse, significa che la variabile da predire può assumere più di due valori possibili.
- Lineare, anche se non tutte le feature hanno questo andamento, quelle principali a grandi linee lo seguono.
- Diverse feature hanno un indice di correlazione basso, per questo potrebbe essere utile scegliere dei modelli che non hanno bisogno di molta feature selection e ridurre di conseguenza il lavoro da fare in preprocessing.

I modelli scelti devono poter funzionare in queste condizioni, in più mi piacerebbe che almeno una parte dei modelli sia leggera in modo da trovare un'effettiva applicazione in caso di una possibile integrazione in un videogioco.

3.1 Modelli scelti

Nella scelta dei modello ho cercato di selezionarne diversi facendo attenzione a dare un diverso scopo ad ognuno di loro. I modelli scelti sono:

- Softmax regression, con lo scopo di fare da base di paragone per altri modelli.
- Decision tree classifier, un modello robusto semplice che dovrebbe funzionare bene con questo tipo di dati.

- Ridge regression arrotondando la previsione alla classe più vicina, questa è una scelta abbastanza originale, ma dettata da alcune motivazioni che saranno trattate in seguito.

Ho deciso di usare sempre lo stesso pre-processing, che consiste in uno scaling con una feature selection molto semplice, questo perché tutti i modelli scelti fanno una certa quantità di feature selection in autonomia. In particolare, ho deciso di togliere solo Game ID, perché che lo ritengo completamente scollegato dalla classificazione che stiamo facendo. Mentre lo scaling applicato è uno scaling MinMax.

Per gli ensemble invece, ho usato un bagging del Decision tree chiamato Random Forest Classificator, che è un modello robusto e molto usato, anche se per realizzarlo si sacrifica l'interpretabilità del Decision tree. Mentre per gli altri due modelli, visto che hanno una varianza bassa, ho optato per un ensemble di Boosting.

Softmax regression

È una versione della logistic regression che usa la curva softmax invece che la sigmoide. Questo le permette di dare in output un vettore distribuito secondo una legge di probabilità i cui elementi hanno somma uguale a uno. In questo modo è possibile estendere la logistic regression anche a casi di classificazione multiclasse, che normalmente non potrebbero essere risolti con questo modello.

La softmax regression segue un comportamento simile alla linear regression con gradient descent, il che comporta che i pesi devono essere calcolati in modo iterativo. Quindi se fosse necessario calcolare tanti modelli, per la cross validation o per l'ensemble, potrebbe diventare pesante da processare.

Decision tree classifier

Questo modello è un modello di classificazione semplice da capire e per questo anche da interpretare. Consiste in uno schema costituito da nodi, nodi terminali o foglie e rami, da ogni nodo partono due rami e per questo il grafico prende la forma di un albero, da cui ne deriva il nome. Dentro ad ogni nodo è contenuto un test che viene usato per decidere quale ramo percorrere, questo viene fatto per diversi nodi fino a che non si arriva a un nodo terminale o foglia che contiene la classe selezionata per quel campione.

Una volta addestrato l'albero, è facile classificare un nuovo elemento perché basta percorrere l'albero in base alle caratteristiche dell'elemento in questione.
Ho scelto questo modello perché è un classificatore molto efficiente e in più penso che la sua versione in ensemble, il Random forest classifier, sia il modello migliore per questo tipo di studio vista la sua semplicità e resistenza vari tipi di anomalie sui dati.

Ridge regression con arrotondamento all'intero più vicino

La ridge regression è un modello di regressione abbastanza usato spesso per la sua capacità

naturale fare feature selection, inoltre ha il pregio di poter essere usata in ciclo chiuso, il che vuol dire che può arrivare a una soluzione in modo non iterativo e quindi molto veloce.

Per quanto questa sembri una scelta fuori luogo, ci sono alcune motivazioni per cui penso che questo modello possa essere una valida aggiunta a questa analisi.

In primo luogo, SkillCraft è classificato dai creatori come un dataset per svolgere task di regressione e quindi volevo provare a rispettare questo limite almeno con uno dei modelli.

L'altro aspetto di questa scelta, secondo me il più interessante, è che questa analisi non è proprio il classico esempio di classificazione multinomiale, dove un errore nel classificare è solo fine a se stesso. In questo caso sono convinto che, se calato in certi contesti, anche una classificazione sbagliata possa comunque trovare delle applicazioni pratiche e questo è dovuto al fatto che le classi di quest'analisi hanno un aspetto gerarchico.
Ad esempio, se classificando il giocatore A della lega Bronzo risulta che la sua classe di abilità è 7, ovvero la lega Grand Master, allora è possibile fare l'assunzione che, anche se la predizione fosse sbagliata, non lo sia di tanto, e quindi questo giocatore è troppo forte per la categoria in cui si trova.

Questa assunzione poteva essere fatta anche con la softmax regression, ma in quel caso non avremmo il pregio di poter risolvere il problema in forma chiusa, che è un grosso vantaggio se si pensa alle applicazioni pratiche che potrebbe avere questa classificazione. In più, il fatto che questo modello sia una regressione, permette di avere facile accesso alle metriche di questa categoria, usate spesso per verificare la dimensione dell'errore commesso.

4 Dettagli implementativi

In questa sezione saranno trattati i dettagli implementativi e le scelte che ho fatto durante la costruzione di questo progetto.

Il progetto è scritto in Phyton e ho fatto largo uso della libreria sklearm, una libreria per il machine learning molto usata in questo ambito. All'interno di essa si possono trovare la maggior parte delle funzioni necessarie per un progetto di learning.

In questo capitolo analizzeremo, prima quali tecniche sono state usate per dividere i dati, poi l'implementazione di tutti i modelli in tutte le loro versioni e infine quali metriche sono state usate per valutare i risultati.

Tutti i modelli sono stati implementati in tre versioni:

- Versione 1, che è una versione base costruita sui dati originali
- Versione 2, che è una versione base costruita sui dati pre-processati. In tutti e tre i modelli sono state usate le stesse tecniche di pre-processing in modo da poter fare un confronto più attendibile.

- Versione 3, che è una versione costruita sull'ensemble della versione migliore tra le due precedenti.

In tutte le versioni è stata usata la cross validation per scegliere i migliori iper-parametri.

4.1 Suddivisione dati e Cross validation

Come visto nell'analisi dei dati, il data set è sbilanciato, quindi ho dovuto prestare attenzione ad alcuni dettagli in tutte le operazioni che prevedevano la suddivisione dei dati in sottogruppi.

In particolare, ho dovuto usare una tecnica chiamata stratificazione che serve ad assicurarsi, ogni volta che si divide il dataset in sottoinsiemi, che il numero di elementi di ogni classe all'interno del sottoinsieme sia proporzionale a quello del dataset iniziale.

Inoltre, ho deciso di suddividere il dataset nelle due parti di train e di test prima di iniziare l'implementazione dei modelli. Questa operazione è stata necessaria perché certe classi hanno una numerosità molto bassa. Se non avessi preso questa precauzione, il risultato dell'addestramento sarebbe variato a seconda di quali elementi venivano selezionati e questo avrebbe reso difficile capire quale effetto era causato del cambiamento dei parametri delle funzioni e quale dalla differente selezione dei campioni.

Durante l'addestramento dei modelli ho invece usato una tecnica chiamata k-fold per scegliere gli iper-parametri dei modelli. Il suo funzionamento è il seguente:

- Divide i dati di train in k parti.
- Fa l'addestramento su k-1 parti .
- Usa la parte esclusa dall'addestramento per la validazione.
- Ripete k volte, ovvero fino a che tutte le parti sono state usate per la validazione una volta.

Questo procedimento permette di calcolare le prestazioni di un modello in modo migliore e meno dipendente dai dati che vengono selezionati. Per trovare gli iper-parametri migliori basta ripetere questo procedimento per ogni parametro che si vuole trovare e alla fine si selezionano gli iper-parametri con le prestazioni migliori.

Per implementare sia la suddivisione nelle parti di test e train, che la cross validazione per la selezione del modello, ci sono delle funzioni già fatte in sklearn.

Per la suddivisione del dataset ho usato Train test split. Nel nostro caso è importante aggiungere l'opzione stratify con la variabile target, in questo modo si avrà una distribuzione delle classi circa proporzionale a quella dell'intero dataset. Un'altra scelta importante da prendere quando si usa questa funzione è la dimensione di test e train set. In questo caso ho scelto di impostare il test set al 20 % del dataset totale, questo perché certe classi hanno una numerosità molto bassa e, se avessi scelto una dimensione troppo grossa del test set, non ci

sarebbero stati abbastanza elementi di quelle classi per un buon addestramento dei modelli.

Per implementare la cross validazione invece, ho usato principalmente due funzioni, la prima per la Ridge regression chiamata RidgeCV , che è una funzione che costruisce il modello usando la cross validazione per la ricerca degli iper-parametri.

Per tutti gli altri modelli ho usato una funzione chiamata GridSearchCV, anch'essa usata per la selezione degli iper-parametri dei modelli. In entrambe le funzioni ho impostato come tipo di cross validation la StratifiedKFold, che è la versione stratificata di k-fold. Dopo diverse prove ho visto che il numero di fold ideale è 5, avrei preferito un numero più alto, ma il numero di campioni delle classi 7 e 8 sarebbe stato troppo basso per poter ottenere una classificazione efficace.

In più, nella funzione GridSearchCV ho usato come metrica per la scelta dei parametri migliori l'F1 score weighted, che è una metrica abbastanza completa, in particolare tiene conto anche dello sbilanciamento dei dataset, che risulta ideale per il nostro caso.

4.2 Softmax regression

Le prestazioni della Softmax regression dipendono principalmente dai parametri che vengono usati per la regolarizzazione, penalty e costo. Questa opzione permette aggiunge di un vincolo di costo totale che può essere basato sulla Ridge o sulla Lasso regression, questo permette di ridurre il peso delle variabili marginali che altrimenti porterebbero a una varianza del modello troppo alta.

Per calibrare la regolarizzazione si usano due parametri:

- Penalty, che sceglie il tipo di regolarizzazione da applicare. Può essere l1 (Lasso) o l2(Ridge), la differenza tra le due è che la l1 può azzerare il peso di alcune feature, di fatto eliminandole, mentre la Ridge può soltanto renderle molto piccole.
- C, che è il vincolo sul costo totale, ovvero il costo totale che si può raggiungere.

Per la C ho messo tra le alternative 1 e alcuni sui sottomultipli. Per il tipo di penalty invece il discorso è un po' più complesso, inizialmente ho messo tra le possibili alternative sia l1 che l2, ma dopo diverse prove ho deciso di lasciare solo l2 e togliere l1. I motivi che mi hanno portato a questa scelta sono:

- Entrambe le penalty l1 e l2 avevano prestazioni molto simili per le versioni 1 e 2.
- Non ci sono così tante features da giustificare l'eventuale eliminazione di alcune di esse.
- Infine, la versione 3 di ensemble aveva prestazioni maggiori se il modello su cui era costruita usava la penalty l2.

Dopo diverse prove per verificare che queste osservazioni erano corrette, ho deciso di togliere la penalty l1 dai modelli.

Approfondiamo ora come sono state costruite le versioni 2 e 3. Queste due versioni aggiungono diverse tecniche per cercare di migliorare il risultato della predizione.

La versione 2 usa dei dati pre-processati, come anticipato prima il preprocessing usato è il Min Max Scaling, che consiste nel trasformare i dati in modo che stiano dentro l'intervallo [0,1], ma mantenendo le proporzioni generali dei valori nell'insieme. Oltre a questo, ho anche tolto la feature GameID perché la ritengo inutile ai fini di questa classificazione.

La versione 3 è composta dall'ensemble della migliore tra i due casi precedenti e nella nostra situazione la versione 2 è risultata essere migliore, quindi il modello di ensamble è stato costruito con essa.
Il tipo di ensemble scelto è stato il Boosting, questo perché per si adatta meglio ai modelli come che hanno poca varianza, come questo caso. Il Boosting consiste nell'addestrare diversi modelli in serie e aggiornare i pesi dopo ogni addestramento in base a come si comporta il modello nuovo rispetto a quello vecchio. Lo svantaggio di questo modello è che addestrando i modelli in serie è più lento rispetto al Bagging dove i modelli possono essere addestrati in parallelo.
Per realizzarlo ho usato AdaBoostClassifier, una funzione di sklearn che permette di costruire ensemble di boosting a partire da altri modelli.

I parametri iniziali che ho fissato sono:

- Il modello, il modello dalla versione 2 con i parametri migliori
- Il numero di stimatori. Dove possibile ho cercato di inserire questo parametro nella cross validazione del modello, ma in questo caso rendeva l'addestramento troppo lento. Il valore che ho scelto è 50, ho visto che questo numero mi permette di avere una via di mezzo tra velocità di training e performance.
- Il random state, che serve a rendere la selezione randomica dei dati riproducibile tra un'esecuzione e l'altra.

Invece l'unico parametro che ho variato durante la cross validation è il learning rate, che indica il peso di quanto i parametri di un nuovo modello possono influire nella modifica dei parametri del modello precedentemente trovato.

4.3 Decision tree classification

Per quanto riguarda il decision tree ho deciso di impostare di base il parametro di class weight come balanced, una scelta necessaria visto che il dataset è sbilanciato.
Mentre durante la cross validation del modello ho deciso di provare diverse profondità massime dell'albero e diversi criteri di valutazione.

Per quanto riguarda la profondità massima ho deciso di provare tutte le decine da 10 fino a 100, anche se possono sembrare tanti parametri da testare, il Decision tree ha un

addestramento abbastanza veloce, quindi non ci sono grossi problemi di lentezza.
Per i criteri di valutazione invece ho deciso di provarne due:

- Il coefficiente di Gini, che è misura la disuguaglianza della distribuzione. Questo viene usato all'interno delle foglie dell'albero per capire quanti elementi vengono messi dentro una classe anche se non le appartengono.
- L'entropia, anche questo indice è usato per calcolare l'omogeneità dei campioni classificati all'interno di una foglia.

Nella versione 2 ho usato lo stesso tipo di pre-processing usato anche per la Softmax regression. Questo perché gli alberi decisionali non hanno grosse difficoltà ad usare dati non pre-processati, soprattutto in casi come questo dove le feature sono poche, quindi come preprocessing penso sia sufficiente uno scaling dei dati.
Anche in questa versione ho provato gli stessi iper-parametri della versione 1 tramite cross validation.

Il tipo di ensemble del decison tree che ho deciso di usare è la Random Forest Classification, che usa diversi alberi decisionali generati in modo randomico e non legati tra loro per produrre una lista di risultati, il cui risultato finale sarà quello più numeroso. Visto che gli alberi possono essere addestrati in parallelo, qui ho potuto mettere un numero maggiore di stimatori.
Il modello che ho usato in questo caso è il primo, ovvero la versione con i dati originali, questo perché è risultata essere la migliore, ma dei suoi parametri ho deciso di tenere solo il criterio di valutazione e ricalcolare il resto.

I parametri usati nella cross validation sono:

- Numero di stimatori, ho messo con 100, 200 e 300 perché ho osservato che quasi sempre viene messo a 200 e volevo dargli un po' di spazio per variare in caso i dati lo richiedano.
- La profondità degli alberi, in questo caso invece che prendere quella del modello migliore ho deciso di lasciarla variare per dare un po' più di elasticità al modello.

4.4 Ridge regression

Per realizzare questo modello ho creato prima un modello di regressione con la funzione RidgeCV, che è una funzione di sklearn che permette di realizzare modelli di Ridge regression con la cross validation integrata. Poi ho arrotondato i risultati della regressione all'intero più vicino con la funzione rint di numpy, una libreria usata principalmente per gestire matrici e array multidimensionali. Infine ho usato la funzione clip di numpy, ovvero una funzione che scelto un limite massimo e uno minimo riporta tutti i valori dell'insieme bersaglio all'interno dell'intervallo compreso tra il massimo e il minimo, nel nostro caso [1,8].

Per realizzare la cross validation in questo modello viene usato come criterio di valutazione

R2 score, visto che non è un modello di classificazione non è possibile usare F1 score per valutare le prestazioni.
Come parametri durante la cross validation del modello ho deciso di usare solo Alpha, che è uguale 1/2C, dove C è il vincolo di costo, questo permette al modello di tarare i parametri dando importanza solo a quelli significativi.
Come preprocessing ho usato sempre lo stesso, visto che questo modello ha caratteristiche simili a quelli precedenti, ovvero di non aver bisogno di molta features selection.
Per il modello in ensemble ho deciso di usare un modello di Boosting, come nella Softmax Regression, perché anche questo modello è caratterizzato da una bassa varianza.
In questo caso ho usato una funzione chiamata AdaBoostRegressor e come parametro ho inserito soltanto il learning rate. Anche questo modello ha il problemadi dover calcolare tutti gli stimatori in serie, e se si aggiungono tanti parametri alla cross validazione il training diventa lento.

4.5 Metriche di valutazione

Per valutare le prestazioni dei modelli è necessario usare le metriche corrette in base alla situazione in cui ci si trova.

Le metriche che ho scelto sono:

- Precision, rappresenta la percentuale di predizioni positive corrette rispetto al numero totale di predizioni per quella classe.
- Racall, rappresenta la percentuale di predizioni positive corrette rispetto alla numerosità totale di quella classe.
- F1 score, è una metrica più completa che viene costruita sulla base di precision e recall.

Per ognuna di queste ho preso sia la versione weighted che quella macro. Quella weighted considera il peso della numerosità di quel tipo di campione, quindi dà un'indicazione delle capacità generali del modello.
La macro invece dà lo stesso peso a ogni classe, anche se la numerosità è molto bassa, questo permette di riflettere la capacità del modello di classificare tutte le classi

Tutte queste metriche vengono fornite da una funzione di sklearn chiamata classification report, che fornisce anche i valori di tutte queste metriche per ogni classe, un dettaglio che fa sempre comodo avere. Per poterla usare meglio ho creato una funzione che usa il classification report e toglie le metriche che non volevo avere. In più, in questa funzione i risultati vengono memorizzati in due modi diversi, uno definitivo che salva il report in formato xlsx e l'altro salva i dati in un dizionario che saranno usati per la successiva analisi dei risultati.

Oltre a queste metriche ho deciso di raccogliere i tempi di training per verificare la velocità dei modelli e poterli paragonare anche sotto questo punto di vista.

Per paragonare meglio le prestazioni dei modelli ho deciso di graficare i risultati in modo

che siano più facili da consultare, in particolare ho fatto due grafici: un grafico contenente le metriche e un grafico dei tempi di addestramento.

5 Risultati e discussione finale

Per trarre le conclusioni di questo studio useremo i risultati delle metriche e dei tempi di addestramento. Questi ci permetteranno di capire quale modello è migliore in quali circostanze, certi scenari traggono maggior vantaggio dalla leggerezza del modello, mentre altri prediligono la precisione nelle predizioni.

Il grafico seguente contiene i risultati di tutti i modelli e lo loro versioni, divisi secondo le metriche scelte.

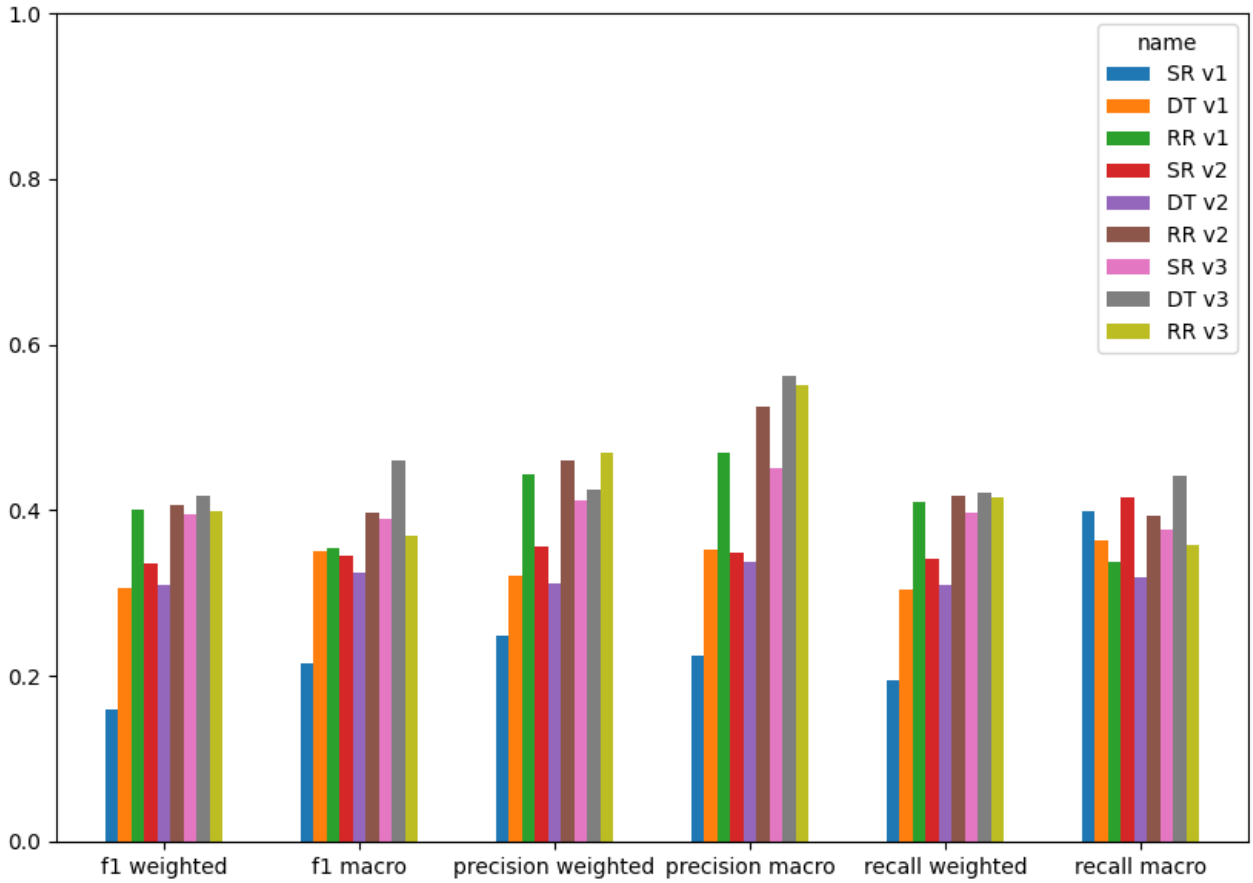

Figure 6: Grafico risultati

Come si può vedere non tutti i modelli sono migliorati con il pre-processing dei dati e l'implementazione delle versioni in ensemble, ma forse la sorpresa più grossa sono le prestazioni del modello di Ridge regression, che anche nella sua versione più semplice ha ottenuto risultati paragonabili alle versioni in ensemble degli altri modelli.

Oltre alle metriche valuteremo anche i tempi di addestramento di ogni modello, che per comodità sono stati riportati nel grafico sotto.

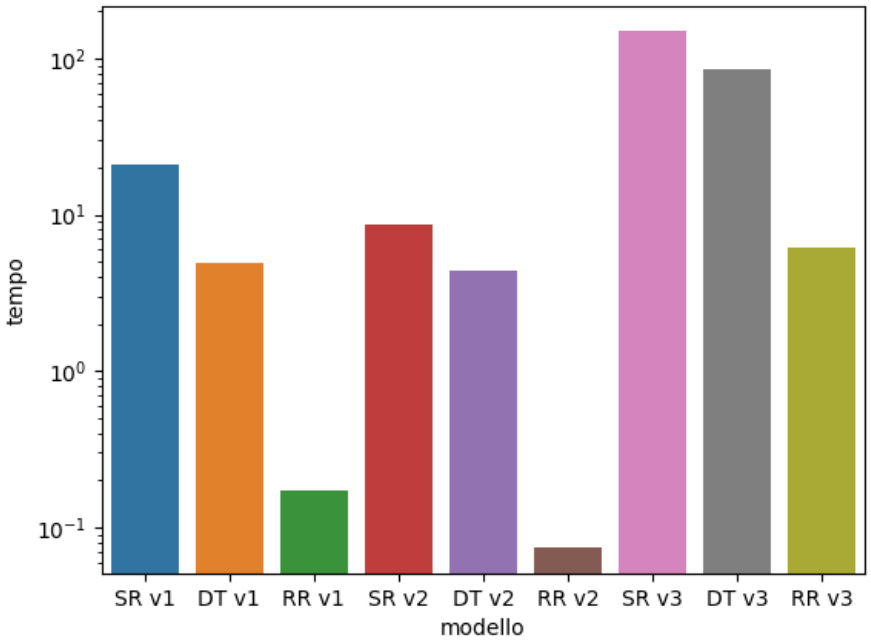

Figure 7: Grafico tempi di addestramento

5.1 Risultati della Softmax regression

Questo modello è stato selezionato per essere una base line, ovvero un termine di paragone, e si è comportato esattamente come ci aspettavamo.

Nella versione 1, quella basilare con i dati originali, ha avuto risultati abbastanza scadenti e incostanti, spesso ha mostrato anche segni di Overfitting, il suo f1 score weighted oscilla solitamente tra lo 0.1 e lo 0.2, ma penso che il dato più critico sia che spesso non riesce a predire nessun elemento di alcune classi. Questa è una grossa criticità che non è possibile ignorare.

Nella versione 2, che usa lo stesso modello della prima, ma con i dati pre-processati, i risultati sono tendenzialmente migliori, l'f1 score weighted in questo caso si aggira spesso attorno allo 0.3 e raramente scende sotto lo o.25, in più non sembra più soffrire di overfitting. Questo è un progresso non da poco, ma rimane comunque il problema che a volte non

riesce a classificare certe classi, anche se è diventato un evento più raro rispetto a prima.

Nella versione 3, che usa un ensemble con i dati pre-processati, finalmente si raggiungono dei risultati accettabili. L'f1-score weighted purtroppo resta sempre attorno a 0.4, ma è comunque un progresso rispetto a prima, l'incapacità di riconoscere certe classi è diventa un'eventualità molto rara in questa versione.

I risultati di questo modello sono buoni, ma i tempi di addestramento della Softmax regression sono i più alti se si paragonano alle stesse versioni degli altri modelli.

Comunque abbiamo ottenuto ciò che volevamo da questo modello, siamo riusciti a dimostrare l'efficacia che tecniche come pre-processing ed ensemble possono avere su certi algoritmi e ora abbiamo una buona base da paragonare agli altri algoritmi per evidenziarne gli effettivi pregi.

5.2 Risultati della Decision tree classificator

Questo modello è stato scelto principalmente per avere la sua versione in ensamble, ma anche la sua versione più semplice ha dimostrato una buona performance.

Nella versione 1 questo algoritmo ha avuto risultati paragonabili alla versione 2 della softmax regression, ma non ha avuto grandi miglioramenti una volta aggiornato alla versione 2, anzi le performance sono leggermente inferiori.

Ciò sottolinea che un classificatore basato su Decision tree non ha grossi problemi con dati non pre-processati, che è un punto a favore visto che è una cosa in meno da fare.

In più vorrei sottolineare che questo modello riesce sempre a classificare almeno una parte dei campioni per ciascuna classe anche nelle versioni 1 e 2, quindi non ha il problema principale che penalizzava la Softmax regression.

La sua versione 3 invece ha risultati paragonabili a quelli dell'ensemble della Softmax regression, ma in media il tempo impiegato per addestrare il modello dell'algoritmo è la metà.

Tirando le somme nel confronto tra Softmax regression e Decision tree, il secondo risulta superiore sotto ogni aspetto, riesce ad ottenere prestazioni paragonabili alla Softmax regression con pre-processing anche nella sua versione più semplice e lo fa impiegando meno tempo, quindi si può dire che la versione 1 del Decision tree è migliore delle prime due versioni della Softmax regression e anche la terza si comporta meglio della corrispettiva ottenendo gli stessi risultati in meno tempo.

5.3 Risultati della Ridge regression

Questo ultimo modello è quello che mi ha sorpreso di più durante questa analisi. Pur essendo uno strumento usato nella regressione, ovvero nella previsione di valori continui, è riuscito ad ottenere ottime prestazioni anche in un caso di classificazione.

In più la sua caratteristica di essere un modello di regressione ci permette di calcolare il mean square error in modo facile e capire di conseguenza di quanto si sbagliano le classificazioni errate, permettendoci di valutare l'utilità del modello in caso la sua precisione non sia alta.

I risultati della versione 1 di questo modello sono i più sorprendenti, senza pre-preprocessing o ensemble riesce a ottenere delle prestazioni paragonabili, e spesso anche superiori, rispetto ai modelli in ensemble delle altre due alternative. Anche se comunque l'f1 score weighted resta spesso vicino allo 0.4 e non supera mai lo 0.5, quindi in generale non ha dei punteggi altissimi.

Il suo aspetto migliore sono i tempi di addestramento! Se confrontato alle versioni migliori degli altri due modelli, il paragone è schiacciante: i tempi di addestramento della ridge regression sono circa 2 decimi di secondo, mentre Softmax regression e Random forest classifier impiegano rispettivamente circa 150 e 80 secondi.

La versioni 2 è leggermente più precisa della versione 1 e impiega ancora meno tempo per l'addestramento, meno di 1 decimo di secondo.

Diversamente la versione 3 non è molto conveniente, finisce con l'avere dei risultati leggermente migliori, ma scambiandoli per tempi di addestramento molto più lunghi, anche se comunque migliori delle stesse versioni degli altri modelli.

L'ultima nota è che il mean square error dell'algoritmo è sempre circa 1, che vuol dire che anche quando la predizione è errata il suo valore è solitamente nelle classi adiacenti a quella corretta. Per questo è possibile trovare alcune applicazioni pratiche anche se la precisione dell'algoritmo non è altissima.

Tirando le conclusioni, la Ridge regression risulta il modello migliore perché ottiene le prestazioni più alte con un tempo di addestramento molto inferiore. La sua versione migliore risulta essere la versione 2, dimostrando nuovamente l'utilità del pre-processing in questo ambito. Queste sue caratteristiche lo rendono sicuramente il modello migliore per provare future applicazioni in campo reale, in particolare i suoi tempi di addestramento molto veloci permettono a questo modello di essere scalato con molti più campioni e mantenere comunque dei tempi di risposta di tutto rispetto.

5.4 Considerazioni finali e futuri sviluppi

In molti videogiochi i giocatori sono divisi in leghe o gradi in base a un sistema a punti, come già accennato nell'introduzione, tale sistema funziona molto bene per la maggior parte dei giocatori, che avendo una crescita graduale riescono a migliorare nel tempo e crescere conseguentemente di livello.

Però penso che questo sistema non riesca a individuare bene le anomalie, ovvero i giocatori che hanno un livello di partenza molto alto. Alcuni giochi, Starcraft compreso, hanno un sistema di piazzamento iniziale, dove viene chiesto al giocatore di giocare alcune partite per essere piazzato al grado corretto. Questa modalità riesce a catturare alcune di queste anomalie, ma non sempre è efficacie, nei sistemi a punti spesso una volta posizionati in una lega ci vuole del tempo per poter scalare i vari gradi.

Ad esempio, un giocatore potrebbe giocare male di proposito nelle 10-15 partite di piazzamento per poi usare l'account per fare Smurfing, fenomeno sempre più diffuso nel mondo dei videogiochi che peggiora l'esperienza videoludica di molti giocatori alle prime armi.

I risultati di questo studio dimostrano che la classificazione fatta usando tecniche di machine learning potrebbe essere una buona opzione in questo ambito. Sicuramente sarà necessario risolvere alcune criticità, ma penso che queste tecnologie abbiano una prospettiva molto promettente nel mondo videoludico.

References

[1] Battle.net leagues. https://liquipedia.net/starcraft2/Battle.net_Leagues. Accessed: 2021-07-20.

[2] Outplayed. https://https://www.outplayed.it/index.php/2020/03/01/gli-smurf-rovinano-il-gioco-dice-riot-games/. Accessed: 2021-07-21.

[3] Skillcraft dataset. https://https://archive.ics.uci.edu/ml/datasets/SkillCraft1+Master+Table+Dataset. Accessed: 2021-07-17.