Università degli Studi di Bergamo(Dalmine)
Corso di Ingegneria Informatica, laurea magistrale

Elaborato: On Making Stochastic Classifiers Deterministic

Sudati Simone 1045936

Corso: Intelligenza Artificiale

Docente: Francesco Trovò

Sessione: Estiva 2019-2020, 24 Luglio

Analisi critica del paper scientifico di Intelligenza Artificiale: "On Making Stochastic Classifiers Deterministic"

21 luglio 2020

Indice

Capitolo 1

Contesto

Prima di addentrarci nel problema, ci serviremo di questo capitolo introduttivo per inquadrare il contesto di applicazione del problema. Inizialmente, nel primo paragrafo, descriveremo la materia e il campo in cui il paper si colloca: il machine learning. Nel secondo e terzo paragrafo specificheremo la branchia del machine learning, ovvero il supervised learning e in particolare la classificazione. Poi passeremo ad inquadrare il paper: descriveremo il problema analizzato (paragrafo 1.2.2) e gli scenari applicativi (paragrafo 1.2.1) in cui si colloca il paper e infine elenchiamo una serie di nozioni utili per comprendere il proseguo degli argomenti.

1.1 Machine Learning

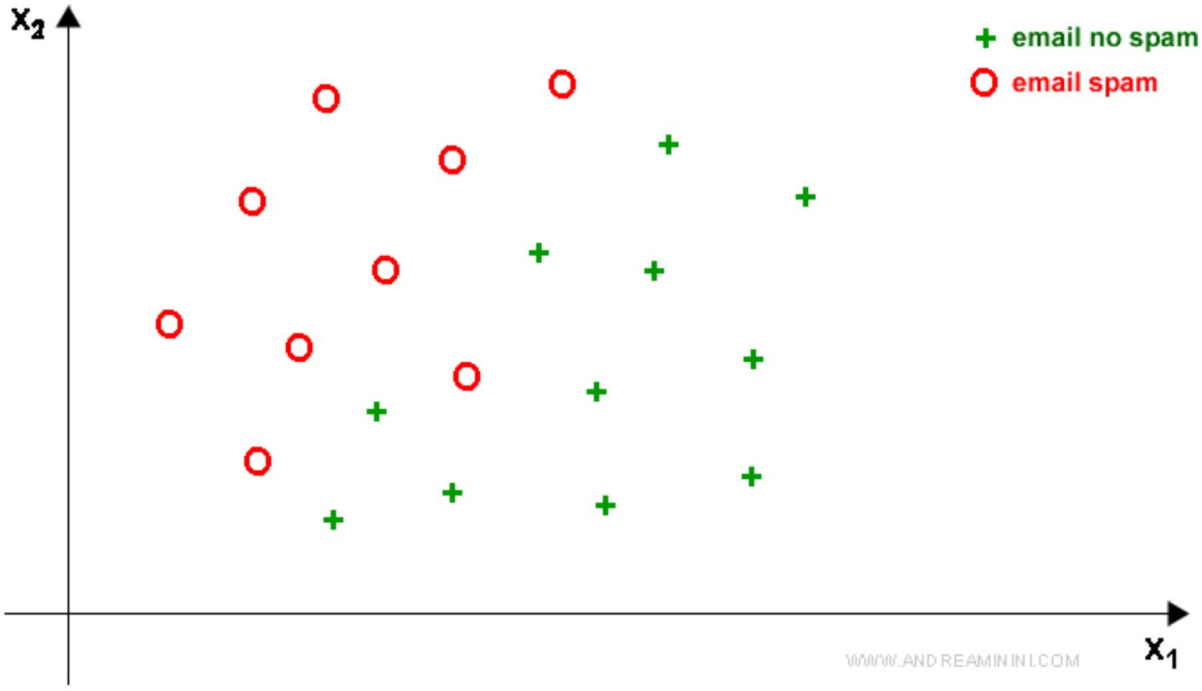
Quando si parla di machine learning [1] (in italiano apprendimento automatico), si parla di una particolare branca dell'informatica che può essere considerata una parente stretta dell'intelligenza artificiale. Definire in maniera semplice le caratteristiche e le applicazioni del machine learning non è sempre possibile, visto che questo ramo è molto vasto e prevede differenti modalità, tecniche e strumenti per essere realizzato. Inoltre, le differenti tecniche di apprendimento e sviluppo degli algoritmi danno vita ad altrettante possibilità di utilizzo che allargano il campo di applicazione dell'apprendimento automatico rendendone difficile una definizione specifica. Si può tuttavia dire che quando si parla di machine learning si parla di differenti meccanismi che permettono a una macchina intelligente di migliorare le proprie capacità e prestazioni nel tempo. La macchina, quindi, sarà in grado di imparare a svolgere determinati compiti migliorando, tramite l'esperienza, le proprie capacità, le proprie risposte e funzioni. Alla base dell'apprendimento automatico ci sono una serie di differenti algoritmi che, partendo da nozioni primitive, sapranno prendere una specifica decisione piuttosto che un'altra o effettuare azioni apprese nel tempo. La strada per realizzare macchine intelligenti è stata lunga, ma ha portato, oggi, ad avere differenti modalità di apprendimento, tutte efficaci, che differiscono non solo per gli algoritmi utilizzati, ma soprattutto per lo scopo per cui sono realizzate le macchine stesse. A seconda del tipo di algoritmo utilizzato per permettere l'apprendimento alla macchina, ossia a seconda delle modalità con cui la macchina impara ed accumula dati e informazioni, si possono suddividere tre differenti sistemi di apprendimento automatico: supervisionato, non supervisionato e per rinforzo. I tre modelli di apprendimento sono utilizzati in maniera differente a seconda della macchina su cui si deve operare, garantendo così sempre la massima performance e il migliore risultato possibile per la risposta agli stimoli esterni.

1.1.1 Supervised Learning

In particolare il paper si occupa di un sottocampo del machine learning chiamato supervised learning. Con il termine supervised learning (o apprendimento supervisionato) [1], [2] si intende la metodologia di apprendimento automatico in cui alla macchina vengono passati degli esempi composti da una coppia di dati contenenti il dato originale e il risultato atteso. Compito della macchina è quello di trovare la regola (funzione o modello) con cui creare una relazione tra i due in modo tale che, al presentarsi di un esempio sconosciuto in precedenza, possa ottenere il risultato corretto. I dati sono precedentemente etichettati, ovvero assegnati a una certa categoria. Il supervised learning ha come training set delle coppie $D = (x, t)$ estratti da una qualche funzione $f(x) = t$. L'obiettivo è trovare un'approssimazione della funzione $f()$ che sia in grado di generalizzare su input non visti nel training set. I vettori x sono detti input o predittori e i valori t sono detti target o risposte o etichette.

1.1.2 Classificazione

Il target può essere, a seconda del problema da trattare, ordinale o categorico o una distribuzione a cui corrispondono rispettivamente un problema di regressione, di classificazione o di stima della probabilità. Nel paper che verrà analizzato il problema trattato è di classificazione. Quindi parliamo di classificazione [2], [3] quando è necessario decidere a quale categoria appartiene un determinato dato. Per esempio, il determinare a quale categoria appartiene la figura con quattro lati, è un tipico esempio di classificazione. [1]



Nell'immagine un primo esempio di classificazione che in seguito tratteremo ampiamente. L'esempio illustra una semplice classificazione di email in "spam" e "no spam" di email ricevute.

[1] Diritti di autore di Andrea Minini http://www.andreaminini.com/ai/machine-learning/classificazione-supervisionata-nel-machine-learning

1.1.3 Classificatori deterministici/stocastici binari

I classificatori binari hanno solamente due target $t \subseteq \{0, 1\}$ una positiva ed una negativa.

- t può essere intepretato come la probabilità della classe positiva
- l'output del modello potrebbe quindi rappresentare la probabilità che esso attribuisce alla classe positiva

Ci sono differenti metriche per valutare le prestazioni di un metodo di classificazione binaria:
[2]

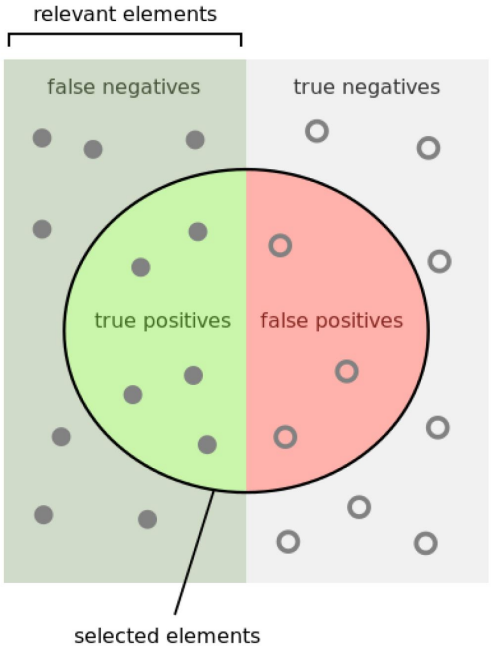
	Actual Class: 1	Actual Class: 0
Predicted Class: 1	tp	fp
Predicted Class: 0	fn	tn

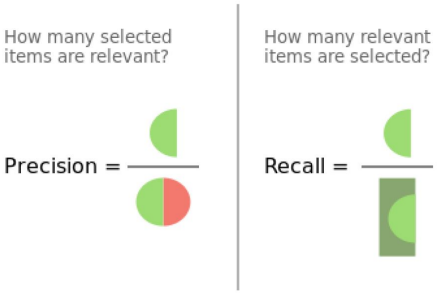
Accuracy	$Acc = \frac{tp+tn}{N}$;
Precision	$Pre = \frac{tp}{tp+fp}$;
Recall	$Rec = \frac{tp}{tp+fn}$;
F1 score	$F1 = \frac{2 \cdot Pre \cdot Rec}{Pre+Rec}$;

I classificatori si possono inoltre suddividere in stocastici e deterministici. Come indica il termine sta ad evidenziare la natura randomica dei primi rispetto agli altri. Infatti un classificatore stocastico binario è un classificatore che ha una probabilità di dare target positivo e la restante target negativo (per esempio $t = 1$ con probabilità 55% e 45% il restante per $t = 0$). Anche ripetendo lo stesso esempio il classificatore stocastico potrebbe dare due risultati differenti. Viceversa il classificatore deterministico binario ha una propria "legge" o funzione secondo la quale definisce come classificare un input e ripetendo lo stesso input x darà sempre la stessa classificazione. Un modello di classificazione stocastica è un processo che determina una regola che associa ad ogni valore della caratteristica x una distribuzione di probabilità $p(\Delta; x)$ sull'insieme di classi. Ancora una volta, il modello di classificazione deve essere innescato o addestrato su un campione di unità con caratteristiche misurate e classi note. (Nel linguaggio statistico, il classificatore è un modello statistico con parametri sconosciuti da stimare dall'allenamento dati).

[2] https://trovo.faculty.polimi.it/02source/iabg_2019/11_classification.pdf

Ora per concludere questo paragrafo saranno mostrate due metriche molto utilizzate quali la precision e la recall. [3]





[3] https://trovo.faculty.polimi.it/02source/iabg_2019/11_classification.pdf

1.2 Paper scelto

Il paper analizzato nel seguente elaborato è "On Making Stochastic Classifiers Deterministic" sviluppato e pubblicato nell'anno 2019 da Andrew Cotter, Maya Gupta e Harikrishna Narasimhan. Esso rientra nei lavori presentati alla conferenza NeurIPS 2019. La conferenza e il seminario sui sistemi di elaborazione delle informazioni neurali è una conferenza sull'apprendimento automatico e sulle neuroscienze computazionali che si tiene ogni dicembre.
http://papers.nips.cc/paper/9273-on-making-stochastic-classifiers-deterministic.pdf

1.2.1 Scenari applicativi

I classificatori stocastici sorgono in una grande varietà di problemi di machine learning. Ad esempio, sono prodotti da problemi di allenamento vincolati [4], [5], in cui si cerca di ottimizzare un soggetto con obiettivo di classificazione puntando all'equità, al recall e al churn. L'uso di classificatori stocastici risulta essere cruciale per rendere trattabili tali problemi di ottimizzazione vincolati, a causa della loro natura potenzialmente non convessa dei vincoli [6]. Per motivi simili, i classificatori stocastici sono importanti per l'ottimizzazione personalizzata delle metriche di valutazione come l'ottimizzazione robusta [7] o la G-media o la metrica H-media popolare in compiti di classificazione squilibrata per classe [8], [9]. I classificatori stocastici inoltre subentrano nella letteratura del PAC-Bayes[10], [11], nell'ensemble learning[12].

1.2.2 Problema analizzato

I classificatori stocastici risultano essere molto utili ma la loro natura randomica li rende problematici da usare nel concreto per una serie di ragioni pratiche. Infatti spesso nella pratica non ha senso usare classificatori stocastici. Per esempio un classificatore per filtrare le email se sono o meno spam, che a singola applicazione rigetta lo spam con il 99% delle volte. Per riuscire ad oltrepassare un classificatore stocastico così costruito basterebbe inviare centinaia di copie, fiduciosi che qualcuna passerà il controllo. Tra gli altri problemi aggiungiamo che molti ricercatori che si occupano degli aspetti pratici sostengono che sono difficili da debuggare, farne il test e la visualizzazione; nonchè aggiungono complessità al modello. Infine i classificatori stocastici possono essere visti come "ingiusti" perchè non rispettano il principio di equità secondo cui "Simili individui devono ricevere lo stesso trattamento"[23]. Invece con i classificatori stocastici perfino lo stesso esempio può ricevere due risultati diversi se classificato in due iterazioni differenti.
Ora brevemente descriverò la soluzione proposta dal paper.
Il paper si prende carico di rispondere alla domanda: "Quanto bene un classificatore stocastico può essere approssimato da un classificatore deterministico?" e allo stesso tempo attraverso diversi tipi di approcci vuole riuscire a fornire dei bound (sia lower che upper) all'approssimazione stessa che può essere compiuta. Inoltre consapevole

che, seppur sconsigliati da utilizzare, è spesso difficile evitare l'utilizzo dei classificatori stocastici (per esempio nei problemi di ottimizzazione soggetti a vincoli di non convessità che forniscono garanzie teoriche) il paper vuole investigare la questione di come rendere un classificatore stocastico dato un classificatore deterministico. E da esso quali problemi sorgono e di quale criteri si possono usare per giudicarne la bontà. L'idea chiave dell'approccio utilizzato è di non cercare una funzione deterministica che approssimi il classificatore stocastico puntualmente ma invece di ricercarne una che performi bene in termini di metriche aggregate su gruppi di dati.

1.2.3 Nozioni preliminari

Volendo approssimare la funzione f appartente ad uno spazio F utilizzando i dati appartenenti al training set. Per specificare un metodo di machine learning dobbiamo definire 1) Una loss $l(f)$ 2) uno spazio di ipotesi H 3) un metodo di ottimizzazione . La loss function [13] è una funzione che calcola la bontà di una certa funzione che voglio utilizzare per approssimare la funzione reale. Un esempio di loss è l'errore quadratico medio nella regressione. Una funzione loss è per un singolo esempio di allenamento. Talvolta viene anche chiamata funzione d' errore. Una funzione di costo, d'altra parte, è la perdita media sull'intero dataset di allenamento. Lo spazio di ipotesi è lo spazio all'interno del quale ricerco le funzioni per approssimare la mia funzione reale. Il metodo di ottimizzazione serve per discriminare e scegliere la funzione che meglio approssima la funzione reale; di solito il metodo di ottimizzazione è quindi un metodo di minimizzazione della loss function. Per esempio un algoritmo di ottimizzazione è il gradient descent. Nel caso di due sole classi parliamo di classificazione binaria e il target t può essere solamente o 0 o 1. Per valutare un metodo di classificazione avremo diverse metriche come ad esempio l'accuracy, precision, recall e f1score; tutte date da diverse combinazioni di tp (true positive), fp (false positive), fn (false negative) e tn (true negative). Dove true positive significa che la classe predetta e la classe reale sono entrambe 1, stesso discorso per true negative dove classe predetta e reale sono 0. Quindi vi è stata una corretta classificazione. Invece nel caso di false positive e false negative si ha che nel primo la classe predetta è 1 e classe reale è 1, invece vicervesa nel secondo la classe predetta è 0 e la classe reale 1. Quindi vi è stata un'errata classificazione. La bontà della classificazione sarà tanto maggiore quanto più sarà elevato il valore delle metriche.

Capitolo 2

Studi correlati

In questa sezione ci occuperemo di riassumere e esporre gli argomenti trattati dagli studi correlati al paper. Per ogni studio correlato si farà una descrizione degli aspetti più rilevanti. Il lavoro è "inedito" nel senso che non viene portato avanti un lavoro e/o problema precedentemente elaborato bensì si lavora su un argomento nuovo in seguito al quale potranno esserci ulteriori sviluppi in futuro. Inoltre è un paper molto recente, pubblicato nel 2019, quindi vi saranno probabilmente lavori futuri ad ora non ancora pubblicati (ricerche su motori specializzati come ad esempio `https://scholar.google.com/` non presentano lavori citati o correlati per questo paper ancora). Vi sono diversi elaborati tuttavia dal quale gli autori del paper hanno attinto per svolgere questo elaborato. In questa sezione si descriveranno le caratteristiche principali dei paper.

2.1 Related work & studi citati in bibliografia

[1] Gabriel Goh, Andrew Cotter, Maya Gupta, and Michael P Friedlander. Satisfying real-world goals with dataset constraints. In NIPS, pages 2415–2423. 2016.

Questo documento propone la gestione di obiettivi multipli, su più set di dati allenandosi con i vincoli del set di dati, utilizzando la ramp penality. Per quantificare accuratamente i costi e presentare un algoritmo efficiente, a ottimizzare (approssimativamente) il risultante problema di ottimizzazione vincolata non convessa. Esperimenti su set di dati del settore di riferimento e del mondo reale dimostrano l' efficacia del loro approccio. Quindi questo paper ci aiuta a capire come riuscire a gestire molti obbiettivi su differenti datasets sfruttando dei training set con vincoli sul dataset.

[2] Harikrishna Narasimhan. Learning with complex loss functions and constraints. In AIStats, 2018.

In questo paper si sviluppa un approccio generale per risolvere i problemi di classificazione vincolata, in cui la perdita e i vincoli sono definiti in termini di una funzione generale della matrice di confusione. Gestendo funzioni loss complesse e non lineari come la F-measure, la G-mean o la H-mean e vincoli che vanno dai limiti di budget, ai vincoli di equità, ai limiti di metriche di valutazione complesse. Il loro approccio si basa sul framework di Narasimhan et al. (2015) per la classificazione non vincolata con perdite complesse e riduce il problema dell'apprendimento vincolato a una sequenza di compiti di apprendimento sensibili ai costi. Esperimenti su una varietà di compiti dimostrano l'efficacia dei loro metodi.

[3] Alekh Agarwal, Alina Beygelzimer, Miroslav Dudík, John Langford, and Hanna M. Wallach. A reductions approach to fair classification. In ICML, pages 60–69, 2018.

Viene presentato un approccio sistematico per raggiungere l'equità in un contesto di classificazione binaria. L'idea chiave è quella di ridurre la classificazione equa a una sequenza di problemi di classificazione sensibili ai costi, le cui soluzioni producono un classificatore randomizzato con l'errore più basso (empirico) soggetto ai vincoli desiderati. Questo paper ci mostra come, in contesto di classificazione binaria, riuscire a raggiungere l'equità.

[4] Andrew Cotter, Heinrich Jiang, and Karthik Sridharan. Two-player games for efficient nonconvex constrained optimization. In Algorithmic Learning Theory, pages 300–332, 2019.

Si mostra che: Un approccio naturale all'ottimizzazione vincolata è l'ottimizzazione

del Langragiana, ma non è garantito che funzioni in un ambiente non convesso e, se si utilizza un metodo del primo ordine, non è in grado di far fronte a vincoli non differenziabili (ad esempio vincoli su rates o proporzioni). La Langragiana può essere interpretata come una partita a due giocatori giocata tra un giocatore che cerca di ottimizzare i parametri del modello e un giocatore che desidera massimizzare i moltiplicatori di Lagrange. Viene proposto una variante non a somma zero della formulazione lagrangiana che può far fronte a vincoli non differenziabili, anche discontinui, che chiamiamo "proxy-lagrangiana". Il primo giocatore minimizza il rimpianto esterno in termini di "vincoli proxy" facili da ottimizzare, mentre il secondo giocatore applica i vincoli originali minimizzando il rimpianto di swap. Per questa nuova formulazione, come per la Lagrangiana in ambiente non convesso, il risultato è un classificatore stocastico. Per entrambe le formulazioni proxy-lagrangiana e lagrangiana, tuttavia, viene dimsotrato che questo classificatore, anziché avere dimensioni illimitate, può essere considerato una distribuzione su non più di modelli m + 1 (dove m è il numero di vincoli). Questo è un miglioramento significativo in termini pratici.

[5] Andrew Cotter, Heinrich Jiang, Serena Wang, Taman Narayan, Maya Gupta, Seungil You, and Karthik Sridharan. Optimization with non-differentiable constraints with applications to fairness, recall, churn, and other goals. JMLR, 2019. [To appear: https://arxiv.org/ abs/1809.04198].

Una nuova formulazione porta a un algoritmo che produce un classificatore stocastico giocando un gioco a somma non zero a due giocatori risolvendo quello che chiamiamo un equilibrio correlato semi-ampio, che a sua volta corrisponde a una soluzione approssimativamente ottimale e fattibile al vincolo problema di ottimizzazione. Diamo quindi una procedura che riduce la soluzione randomizzata a una che è un'unione di al massimo $m+1$ soluzioni deterministiche, dati i vincoli m. Questo culmina in algoritmi in grado di risolvere problemi di ottimizzazione vincolata non convessa con possibili vincoli non differenziabili e non convessi con garanzie teoriche. Forniamo ampi risultati sperimentali che impongono una vasta gamma di obiettivi politici tra cui metriche di equità diverse e altri obiettivi su accuratezza, copertura, richiamo e churn.

[6] Robert S. Chen, Brendan Lucier, Yaron Singer, and Vasilis Syrgkanis. Robust optimization for non-convex objectives. In NIPS, 2017.

L'obiettivo è ottimizzare nel peggiore dei casi su una classe di funzioni oggettive. Sviluppiamo una riduzione dalla solida ottimizzazione impropria all'ottimizzazione stocastica: dato un oracolo che restituisce soluzioni α-approssimative per le distribuzioni sugli obiettivi, calcoliamo una distribuzione sulle soluzioni che è α-approssimativa nel peggiore dei casi. Mostrano che togliere il random da questa soluzione è un problema NP-difficile in generale, ma può essere fatto per un'ampia classe di compiti di apprendimento statistico. Valutano sperimentalmente il loro approccio sulla classificazione dei caratteri corrotti e sulla massimizzazione della forte influenza nelle reti.

[7] D.D. Lewis. Evaluating text categorization. In HLT Workshop on Speech and Natural Language, pages 312–318, 1991.

In questo documento si discutono una varietà di modi per valutare l'efficacia dei sistemi di categorizzazione del testo, attingendo sia dagli esperimenti di categorizzazione riportati sia ai metodi utilizzati nella valutazione del recupero guidato da query.

[8]J-D. Kim, Y. Wang, and Y. Yasunori. The Genia event extraction shared task, 2013 editionoverview. ACL 2013, 2013.

Il paper mostra come gestire un task setting e dei datasets, i risultati finali vengono commentati e discussi con diverse osservazioni.

[9] Y. Sun, M.S. Kamel, and Y. Wang. Boosting for learning multiple classes with imbalanced class distribution. In ICDM, 2006.

In questo documento, si svilluppa un algoritmo di boosting sensibile ai costi per migliorare le prestazioni di classificazione di dati squilibrati che coinvolgono più classi. Per risolvere il problema della matrice dei costi si applica l' Algoritmo "Genetic" per cercare la configurazione ottimale dei costi di ogni classe.

[10] S. Wang and X. Yao. Multiclass imbalance problems: Analysis and potential solutions. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 42(4):1119–1130, 2012.

Questo documento studia le sfide poste dai problemi di squilibrio multiclasse e studia la capacità di generalizzazione di alcune soluzioni di ensemble, incluso il l' algoritmo AdaBoost.NC recentemente proposto, con l'obiettivo di gestire la multiclasse e lo squilibrio in modo efficace e diretto.

[11] S. Lawrence, I. Burns, A. Back, A-C. Tsoi, and C.L. Giles. Neural network classification and prior class probabilities. In Neural Networks: Tricks of the Trade, LNCS, pages 1524:299–313. 1998.

L'indagine sul problema(la rete fatica ad imparare le classi più rare in alcuni casi) mostra che la differenza tra i risultati teorici e pratici sta nelle ipotesi formulate nella teoria (la stima accurata delle probabilità bayesiane a posteriori richiede che la rete sia sufficientemente grande, l'addestramento per convergere al minimo globale, i dati infiniti sull'allenamento e le probabilità della classe a priori del set di test per essere rappresentate correttamente nel set di addestramento).

[12] H. Narasimhan, P. Kar, and P. Jain. Optimizing non-decomposable

performance measures: a tale of two classes. In ICML, 2015.

Il contributo principale è uno schema di linearizzazione adattivo per certe famiglie, con il quale svillupare tecniche di ottimizzazione che consentono aggiornamenti stocastici veramente puntuali. Per misure di prestazione concave proponiamo SPADE, un doppio solutore primitivo stocastico; per misure pseudo-lineari proponiamo STAMP, una procedura di massimizzazione alternativa stocastica.

[13] John Langford and John Shawe-Taylor. PAC-Bayes and margins. In NIPS, 2002.

Dato un classificatore stocastico composto da una somma pesata di features con grande margine, si riesce a costruire un classificatore stocastico con errore di training rate irrilevante.

[14] David McAllester. PAC-Bayesian stochastic model selection. In Machine Learning, 2003.

Questo documento offre una garanzia delle prestazioni bayesiane PAC per la selezione di modelli stocastici superiore alle garanzie analoghe per la selezione di modelli deterministici.

[15] Xiong Li, Bin Wang, Yuncai Liu, and Tai Sing Lee. Stochastic feature mapping for PAC-Bayes classification. In Machine Learning, 2015.

In questo documento, si propone un meccanismo di accoppiamento sviluppato nell'ambito del framework PAC-Bayes in grado di mettere a punto i modelli generativi e le funzioni di mappatura delle caratteristiche in modo iterativo per migliorare le prestazioni del classificatore.

[16] Alexandre Lacasse, François Laviolette, Mario Marchand, Pascal Germain, and Nicolas Usunier. Pac-bayes bounds for the risk of the majority vote and the variance of the gibbs classifier. In Advances in Neural information processing systems, pages 769–776, 2007

Proponiamo nuovi limiti PAC-Bayes per il rischio del voto a maggioranza ponderata che dipende dalla media e dalla varianza dell'errore del suo classificatore Gibbs associato.

[17] Foster Provost and Tom Fawcett. Robust classification for imprecise environments. Machine learning, 42(3):203–231, 2001.

In questo paper si mostra che è possibile costruire un classificatore ibrido che per-

formi almeno bene quanto il miglior classificatore disponibile comunque scelte delle condizioni target. In molti casi, le prestazioni del classificatore ibrido posso superare il classificatore migliore già conosciuto.

[18] Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Leon Roth. Preserving statistical validity in adaptive data analysis. In Proceedings of the Forty- Seventh Annual ACM on Symposium on Theory of Computing, pages 117–126. ACM, 2015.

In questo lavoro si studia un principio su come garantire la validità dell'inferenza statistica nell'analisi adattativa dei dati. Viene proposto e studiata la questione della stima delle aspettative di m funzioni scelte in modo adattivo su una distribuzione sconosciuta dati n campioni casuali.

[19] Moritz Hardt, Eric Price, and Nathan Srebro. Equality of opportunity in supervised learning. In NIPS, 2016.

Si propone un criterio di discriminazione nei confronti di un attributo sensibile specificato nell'apprendimento supervisionato, in cui l'obiettivo è prevedere un obiettivo in base alle funzionalità disponibili.

[20] C. Dwork, M. Hardt, T. Pitassi, O. Reingold, and R. Zemel. Fairness through awareness. In Proc. 3rd Innovations in Theoretical Computer Science, pages 214–226. ACM, 2012.

Il grande contributo di questo paper è un quadro per una classificazione equa che comprende 1) una metrica specifica per determinare il grado in cui gli individui sono simili 2) un algoritmo per massimizzare l'utilità soggetta al vincolo di equità, secondo cui individui simili sono trattati in modo simile.

[21] T. Hastie, R. Tibshirani, and J. Friedman. Elements of Statistical Learning. Springer, 2016.

Questo libro tratta di come fare learning a partire dai dati. Un tipico scenario che spiega è il ricevere delle misurazioni sugli output quantitativi o categorici che useremo per predire (basate su un set di features).

[22] G. Sher. What makes a lottery fair? In Nous, pages 203–216, 1980.

Questo paper indaga e approfondice il concetto di fairness (equità) e analizza il perchè si possa considerare equo il gioco della lotteria nel mondo reale. Trae delle conclusioni che permettono di avere delle condizioni precise per definire l'equità nella lotteria e tratta moralemente gli aspetti relativi al gioco equo.

[23] B. Saunders. The equality of lotteries. In Philosophy, volume 83, pages 359–372, 2008.

Questo articolo sostiene che, dato che la selezione appropriata è impossibile quando le parti hanno pari diritti, una lotteria è preferibile a un'asta perché esclude influenze ingiuste.

[24] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. Machine bias: There's software used across the country to predict future criminals, and it's biased against blacks, May 2016.

In questo paper si spiega il funzionamento di un software che a partire dalla foto di una persona stima la probabilità delle persone di essere futuri potenziali criminali.

[25] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017. URL http: //archive.ics.uci.edu/ml.

E' un sito in cui al momento propongono 507 datasets come servizio per la comunità del machine learning.

[26]Michael Kearns, Seth Neel, Aaron Roth, and Zhiwei Steven Wu. Preventing fairness gerrymandering: Auditing and learning for subgroup fairness, 2017. URL https://arxiv.org/ abs/1711.05144.

Viene dimostrato che il problema computazionale di calcolo dell'equità dei sottogruppi sia per l'uguaglianza dei rate di falsi positivi sia per la parità statistica è equivalente al problema dell'apprendimento agnostico debole, il che significa che è computazionalmente difficile nel peggiore dei casi, anche per le semplici sottoclassi strutturate.

[27] L. Wightman. LSAC national longitudinal bar passage study. Law School Admission Council, 1998.

Questo studio fornisce dati longitudinali nazionali per esaminare molte delle domande sollevate sulla ricerca sul passaggio dell'esame e l'ingresso nella professione da parte di membri di gruppi etnici selezionati. Lo studio ha monitorato gli studenti che hanno iniziato la scuola di legge nell'autunno 1991 attraverso tre o più anni di giurisprudenza scuola e fino a cinque amministrazioni dell'esame forense.

[28] L. Dixon, J. Li, J. Sorensen, N. Thain, and L. Vasserman. Measuring and mitigating unintended bias in text classification. In AIES, 2018.

Si presenta e illustra un nuovo approccio alla misurazione e per mitigare la bias involon-

taria nei modelli di machine learning. Illustriamo come questo possa essere utilizzato per valutare i classificatori di testo utilizzando un set di test sintetico e un corpus pubblico di commenti annotati per tossicità da Pagine di discussione di Wikipedia.

[29]M. Gupta H. Narasimhan, A. Cotter. Optimizing generalized rate metrics with three players. In NeurIPS, 2019.

Estendono i precedenti approcci di gioco a due giocatori per l'ottimizzazione vincolata a un approccio con tre giocatori per disaccoppiare i rates di classificazione dall'obiettivo non lineare e cercano di trovare un equilibrio del gioco

[30] A. Beutel, J. Chen, T. Doshi, H. Qian, L. Wei, Y. Wu, L. Heldt, Z. Zhao, L. Hong, E. H. Chi, and C. Goodrow. Fairness in recommendation through pairwise experiments. KDD Applied Data Science Track, 2019. URL arxiv.org/abs/1903.00780.pdf.

In particolare, mostriamo come misurare l'equità basata su confronti a coppie di esperimenti randomizzati fornisce un mezzo trattabile per ragionare sull'equità nelle classifiche dei sistemi. Basandoci su questa metrica, offriamo un nuovo regolarizzatore per incoraggiare a migliorare questa metrica durante l'addestramento del modello e quindi migliorare l'equità nelle classifiche risultanti.

[31] N. Kallus and A. Zhou. The fairness of risk scores beyond classification: Bipartite ranking and the xAUC metric. arXiv preprint arXiv:1902.05826, 2019.

In questo documento, esaminiamo l'equità di punteggi di rischio predittivi dal punto di vista di un'attività di classificazione bipartita, in cui si cerca di classificare esempi positivi superiori a quelli negativi. Usando la disparità xAUC come metrica di valutazione dell'impatto dei punteggi di rischio.

[32] M. Gupta S. Wang H. Narasimhan, A. Cotter. Pairwise fairness for ranking and regression, 2019. URL https://arxiv.org/abs/1906.05330.

Si presentano metriche di equità a coppie per modelli di classificazione e modelli di regressione che formano analoghe nozioni di equità statistiche come pari opportunità, pari accuratezza e parità statistica. La formulazione a coppie supporta sia gruppi protetti discreti, sia attributi protetti continui. Si mostra che i problemi di formazione che ne risultano possono essere risolti in modo efficiente ed efficace utilizzando l'ottimizzazione vincolata esistente e solide tecniche di ottimizzazione sviluppate per un'equa classificazione.

[33] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glo-

ve: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pages 1532–1543, 2014.

In questo paper si analizzano e rendono esplicite le proprietà del modello necessarie affinché tali regolarità emergano nei word vectors. Il risultato è un nuovo modello di regressione log-bilineare globale che combina i vantaggi dei due modelli principali di famiglie in letteratura: matrice globale di fattorizzazione e metodi locali a finestra.

[34] Ronitt Rubinfeld. Notes for lecture 5 of MIT 6.842: Randomness and computation, February 2012. URL https://people.csail.mit.edu/ronitt/ COURSE/S12/handouts/lec5.pdf.

In questo paper si studia il potere e le fonti di casualità nel calcolo, concentrandoci su connessioni e applicazioni alla complessità computazionale, teoria dell'apprendimento computazionale, crittografia e combinatoria. Gli argomenti includono: (1) Strumenti di base: probabilistico, prove, lemma locale di Lovasz, generazione uniforme e conteggio approssimativo, analisi di Fourier, influenza delle variabili sulle funzioni, espansione dei grafici, lemma della regolarità di Szemeredi. (2) Casualità vs prevedibilità: generatori pseudocasuali, fonti casuali deboli, casualità di derandomizzazione e riciclo, teoria dell'apprendimento computazionale, algoritmi di apprendimento basati su Fourier, apprendimento debole contro forte, potenziamento, durezza media vs durezza peggiore, lemma XOR.

Capitolo 3

Descrizione

Nel primo paragrafo di questa sezione ci occuperemo di definire la la notazione che useremo per trattare i classificatori stocastici e deterministici. Nel secondo paragrafo riusciremo a calcolare un lower bound su quanto bene si può approssimare con un classificatore deterministico un classificatore stocastico. Nel terzo paragrafo analizziamo il metodo standard del thresholding e le sue performance, probabilmente il più immediato e semplice metodo deterministico che si può pensare. Nel quarto paragrafo tramite l'approccio hashing che è famoso per essere conosciuto come metodo per rendere un classificatore stocastico deterministico, riusciremo a dimostrare che fornisce valori garantiti molto buoni confrontato al lower bound precedentemente calcolato. Nel quinto paragrafo analizzeremo l'ordine nei classificatori stocastici analizzando l'effetto dell'uso ripetuto e dell'uguaglianza; infine vedendo le caratteristiche di clustering unito ad hashing. Nel sesto paragrafo analizzeremo gli stochastic ensembles. Nell'ultimo paragrafo passeremo all'implementazione di una tecnica per passare da un classificatore stocastico ad uno deterministico in particolare "Hashing function to make a stochastic classifier deterministic.".

3.1 Notazione classificatori

Spazio ipotesi: $\mathcal{X}$

Spazio ipotesi in cui valutare loss function: $\mathcal{X}_\ell \subseteq \mathcal{X}$

Relativa distribuzione associata allo spazio di ipotesi: $\mathcal{D}_x$

I due possibili target (classificazione binaria): $\mathcal{Y} = \{0, 1\}$

Distribuzione condizionata: $\mathcal{D}_{y|x}$

Distribuzione congiunta: $\mathcal{D}_{xy}$

Il classificatore stocastico binario: $f : \mathcal{X} \to [0, 1]$

Il classificatore deterministico binario: $\hat{f} : \mathcal{X} \to \{0, 1\}$

Rate metric: $(\ell, \mathcal{X}_\ell)$

Metrica (è una funzione loss binaria): $\ell : \{0, 1\} \times \{0, 1\} \to \{0, 1\}$

Valore di una metrica stocastica: $E_\ell(f) := E_{x,y}\left[f(x)\ell(1, y) + (1 - f(x))\ell(0, y) \mid x \in \mathcal{X}_\ell\right]$

Valore di una metrica deterministica: $E_\ell(\hat{f}) := E_{x,y}\left[\ell(\hat{f}(x), y) \mid x \in \mathcal{X}_\ell\right]$

3.2 Stochastic Classifiers

Sia $\mathcal{X}$ lo spazio di ipotesi con distribuzione di probabilità associata $\mathcal{D}_x$ e $\mathcal{Y} = \{0, 1\}$ i 2 possibili target (visto che siamo in ambito di classificazione binaria), con $\mathcal{D}_{y|x}$ la distribuzione condizionata del target con corrispondente distribuzione congiunta $\mathcal{D}_{xy}$. I classificatori deterministici saranno sempre scritti con la seguente notazione $\hat{f}$ e i classificatori stocastici f. Un classificatore stocastico binario è una funzione $f : \mathcal{X} \to [0, 1]$ che mappa le istanze x alla probabilità di compiere una predizione positiva. Il nostro obbiettivo sarà di trovare un classificatore deterministico $\hat{f} : \mathcal{X} \to \{0, 1\}$ che approssima f , ma prima dobbiamo chiarire che cosa si intende per "buona approssimazione". Per questo fine definiamo la rate metric come una coppia $(\ell, \mathcal{X}_\ell)$ dove $\ell : \{0, 1\} \times \{0, 1\} \to \{0, 1\}$ è una funzione loss binaria e $\mathcal{X}_\ell \subseteq \mathcal{X}$ è il sottoinsieme dello spazio di ipotesi in cui la loss function dev'essere valutata. Queste rate metric sono sorprendentemente flessibili , e coprono una serie di compiti che servono a chi si occupa della pratica [4]. Per esempio , su un problema di equità basato sul vincolo di parità demografica [15] potremmo essere interessati nel rate di predizioni positive l sui membri di una certa classe protetta $\mathcal{X}_\ell$. Denoteremo il valore di una metrica come $E_\ell(f) := E_{x,y}\left[f(x)\ell(1, y) + (1 - f(x))\ell(0, y) \mid x \in \mathcal{X}_\ell\right]$ per un classificatore stocastico f e come $E_\ell(\hat{f}) := E_{x,y}\left[\ell(\hat{f}(x), y) \mid x \in \mathcal{X}_\ell\right]$ per un classificatore deterministico $\hat{f}$. Noi lavoreremo su diverse metriche $\ell_1, \ldots, \ell_m$ ognuna delle quali cattura alcune proprietà di f che dev'essere preservata (per esempio potremmo volere $E_{\ell_1}(f) \approx E_{\ell_i}(\hat{f}) forall i \in [m]$. Tipicamente , il set di metriche dipende dal problema originale di apprendimento. Per esempio se trovassimo f minimizzando il rate di falsi positivi FPR soggetto a FNR e vincoli di churn, allora la metrica rilevante includerà presumibilmente FPR, FNR e churn. La chiave al nostro approccio è che non cerchiamo di trovare un classificatore deterministico che approssimi un classificatore stocastico puntualmente ma che performi bene rispettando le metriche aggregate su raggruppamenti di dati. entre potrebbe essere allettante formulare la ricerca di $\hat{f}$ come un problema di ottimizzazione esplicita, l'unica tecnica appropriata di cui siamo a conoscenza sono risolutori vincolati che a loro volta producono classificatori stocastici [6], [16] . Invece, ci concentriamo su strategie agnostiche problematiche che sono facili da implementare, ma che nonostante la loro semplicità, spesso godono di buone garanzie teoriche e si comportano bene nella pratica.

3.3 Lower bound

Questo teorema in seguito enunciato definisce un limite all'approssimazione di qualsiasi classificatore deterministico

Teorema 1 (Lower bound)**.** *Dato uno spazio di ipotesi $\mathcal{X}$, una distribuzione dei dati $\mathcal{D}_x$, un sottogruppo di metriche valutato nello spazio di ipotesi $\mathcal{X}_\ell \subseteq \mathcal{X}$ e un classificatore stocastico f , allora esiste una metrica (funzione loss ℓ) e una distribuzione condizionata $\mathcal{D}_{y|x}$ tali che:*

$$\left|E_\ell(f) - E_\ell(\hat{f})\right| \geq \max_{x \in \mathcal{X}_\ell} \left\{ Pr_{x' \sim \mathcal{D}_x|x_\ell} \{x' = x\} \cdot \min\{f(x), 1 - f(x)\} \right\}$$

per tutti i classificatori deterministici $\hat{f}$, dove $\mathcal{D}_{x|\mathcal{X}_\ell}$ ristretta a X_ℓ.

Questo è il primo risultato concreto che abbiamo ottenuto. Esso ci illustra quali sono i 2 più grandi ostacoli che si incontrano nell'approssimazione e nella ricerca di un buon $\hat{f}$ deterministico: A) punti materiali$Pr_{x' \sim \mathcal{D}_x|x_\ell} \{x' = x\}$ B) la stocasticità $\min\{f(x), 1 - f(x)\}$. Se la funzione f che voglio approssimare contiene troppa stocasticità su un grande punto materiale, allora è impossibile riuscire ad approssimarla bene con una deterministica $\hat{f}$. Quanto affermato ci dice che a priori (indipendentemente dalla funzione deterministica che vorremo utilizzare) a seconda della funzione stocastica che voglio approssimare so già dei limiti a quanto bene posso approssimarla. Quindi il problema "risiede" nel classificatore stocastico che voglio approssimare che "può non essere fatto" per essere approssimato da un classificatore deterministico. Per saperlo mi basta valutare i valori dei punti materiali (condizione A)) e della stocasticità (condizione B)).

3.4 Thresholding

Tramite il metodo thresholding riusciremo a provare un upper bound all'approssimazione che si può ottenere. Thresholding è un metodo standard, nonchè il più immediato da pensare per riuscire a approssimare un classificatore stocastico binario dato in uno deterministico. L'approccio si basa sul seguente ragionamento: se $f(x) > 1/2$ allora scegliamo la predizione positiva (per esempio valore=1) e nel caso opposto la predizione negativa. Se il target è scelto in maniera casuale seguendo la distribuzione di probabilità $f(x)$ allora il metodo thresholding viene a coincidere con il classificatore di Bayes e quindi minimizza le classificazioni errate predette [17]. Per qualsiasi scelta di funzione loss l si può ricavare un upper bound delle performance del metodo di thresholding:

Teorema 2 (Thresholding)**.** *Sia $f : \mathcal{X} \to [0, 1]$ un classificatore stocastico, e $\mathcal{D}_x$ la distribuzione dei dati su spazio di ipotesi $\mathcal{X}$. Definito il classificatore stocastisco threshold come $\hat{f}(x) := 1\{f(x) > 1/2\}$. Comunque presa una qualsiasi rate metric $(\ell, \mathcal{X}_\ell)$ e la relativa distribuzione di probabilità condizionata $\mathcal{D}_{y|x}$:*

$$\left|E_\ell(f) - E_\ell(\hat{f})\right| \leq E_{x\sim\mathcal{D}_x|x_\ell}[\min\{f(x), 1 - f(x)\}]$$

dove $\mathcal{D}_{x|\mathcal{X}_\ell}$ è la distribuzione di $\mathcal{D}_x$ ristretta a $\mathcal{X}_\ell$.

Questo upper bound conferma che quanto più il classificatore stocastico f si avvicina ad essere deterministico quanto meglio il classificatore deterministico threshold lo imiterà. Tuttavia analizzato in termini di punti materiali si ha che a differenza del teorema 1 (per il calcolo del lower bound), l'approccio threshold non migliora quando i punti materiali si riducono. Anzi, persino con una distribuzione dei dati $\mathcal{D}_x$ continua (cioè nessun punto materiale), le prestazioni del classificatore threshold $\hat{f}$ potrebbero essere molto scarse. Per esempio, se $f(x) = 0.51$ per ogni x, allora $\hat{f}$ farà sempre predizioni positive (per esempio valore=1), diversamente dal classificatore stocastico di partenza, che fa predizioni negative il 49% delle volte.

3.5 Hashing

Per migliorare le performance del classificatore threshold, scegliamo $\hat{f}$ in modo che le prestazioni siano migliori non solo riguardo una minore stocasticità di f ma anche con punti materiali ridotti in $\mathcal{D}_x$. Con questo fine, proponiamo di "simulare" la casualità di un classificatore stocastico eseguendo l'hashing dei classificatori di input per generare in modo deterministico un numero che sembra casuale. L'idea che sta alla base è che se anche un classificatore prende una decisione deterministica su una data istanza x , producendo predizioni non del tutto simili per istanze che sono vicini a x, il classificatore darà l'illusione di essere stocastico se visto dalla prospettiva analizzata di aggregazione di rate metric. Riusciremo a dimostrare che scegliendo accuratamente il tipo di funzione hash possiamo delimitare fortemente in termini di bounds le performance del risultante classificatore deterministico.

Definizione 1 (Indipendenza a coppie)**.** *Una famiglia $\mathcal{H}$ di funzioni hash $h : \mathcal{C} \to [k]$ su un set finito chiamato C è indipendente a coppie se, per tutti $c, c' \in \mathcal{C}$ e $i, i' \in [k]$, abbiamo che*

$$Pr_{h\sim Unif(\mathcal{H})}\{(h(c) = i) \wedge (h(c') = i')\} = 1/k^2$$

ogni volta che $c \neq c'$

A prima vista, potrebbe sembrare una proprietà abbastanza forte, ma in realtà è abbastanza semplice da costruire una funzione hash indipendente a coppie da un numero logaritmico di bit casuali. Da notare inoltre che abbiamo definito una funzione hash su un set di clusters C invece che sullo spazio di ipotesi X.
Questo gestisce il caso in cui X è un insieme infinito e ci permette di definire una C finita e la cui mappatura associata $\pi : \mathcal{X} \to \mathcal{C}$, il cui risultato, $\Pi(x)$, è ciò sui cui facciamo hash. In sostanza, X sarà comunque finito (ad es. vettori d-dimensionali di numeri in virgola mobile) e uno è quindi libero di scegliere $C = X$ e prendere Π per essere la funzione identità. Tuttavia persino nel caso finito potrebbe esserci beneficio nel pre-assegnare le instanze a dei clusters prima di fare hashing.

Teorema 3. *Sia $f : \mathcal{X} \to [0, 1]$ un classificatore stocastico, e $\mathcal{D}_x$ una distribuzione di dati su X. Supponendo che sia date m metriche $(\ell_i, \mathcal{X}_{\ell_i})\, for\, i \in [m]$, ognuno dei quali è potenzialmente associata a una differente distribuzione di target condizionata $\mathcal{D}_{y_i|x}$. Si prenda $\mathcal{H}$ come set indipendente a coppie di funzioni hash $h : \mathcal{C} \to [k]$ e, $\pi : \mathcal{X} \to \mathcal{C}$ come funzione pre-assegnata di istanze a clusters prima di fare hashing.*
Un campione preso dalla distribuzione $h \sim Unif(\mathcal{H})e$ si definisce un classificatore deterministico $\hat{f}_h : \mathcal{X} \to \{0, 1\}$ come:

$$\hat{f}_h(x) = 1\left\{f(x) \geq \frac{2h(\pi(x)) - 1}{2k}\right\}$$

dove l'espressione $(2h(\Pi(x)) - 1)/2k$ *mappa* $[k]$ *(il range di* h*) tra* $[0, 1]$*.*
Poi con probabilità 1-δ *sul campionamento di* $h \sim Unif(\mathcal{H})$*, per tutte le* $i \in [m]$*:*

$$\left|E_f(\ell_i) - E_{\hat{f}_h}(\ell_i)\right| < \frac{1}{2k} + \left(\frac{m}{\delta}\sum_{c\in\mathcal{C}}\left(\left(Pr_{x\sim\mathcal{D}_x|x_{\ell_i}}\{\pi(x) = c\}\right)^2 \times E_{x\sim\mathcal{D}_x|x_{\ell_i}}\left[\frac{1}{2k} + f(x)(1 - f(x)) \mid \pi(x) = c\right]\right)\right)^{\frac{1}{2}}$$

dove dove $\mathcal{D}_{x|\mathcal{X}_\ell}$ *è la distribuzione di* $\mathcal{D}_x$ *ristretta a* $\mathcal{X}_\ell$*.*

Si noti che $1/2k$ si avvicina a zero all'aumentare del numero di bucket hash k. Il limite superiore del Teorema 3 ha forti somiglianze con il limite inferiore del Teorema 1, in particolare alla luce del fatto che il pre-clustering è facoltativo. Le principali differenze sono che : 1) i punti materiali $Pr_{x\sim\mathcal{D}_x|x_{\ell_i}}\{\pi(x) = c\}$ sono misurati sugli interi clusters $\rfloor \subseteq \mathcal{C}$ anzichè su istanze $\S \subseteq \mathcal{X}$ 2) si prendono le l^2 norme al quadrato sopra i punti materiali, invece di massimizare sopra di essi 3) la stocasticità si misura con la varianza attesa $\left(Pr_{x\sim\mathcal{D}_x|x_{\ell_i}}\{\pi(x) = c\}\right)^2 \leq Pr_{x\sim\mathcal{D}_x|x_{\ell_i}}\{\pi(x) = c\}$ su un cluster, invece di $min\{f(x), 1 - f(x)\}$.

Molto importante è il fatto che, diversamente dall'approccio di thresholding, le proprietà del lower bound si riescono a calcolare quando si usa un approccio con hashing. Sarà più facile se allentiamo/togliamo i limiti introdotti con il teorema 3: 1) la stocasticità come $f(x) \times (1 - f(x)) <= 1/4$ oppure 2) i punti materiali come $\left(Pr_{x\sim\mathcal{D}_x|x_{\ell_i}}\{\pi(x) = c\}\right)^2 \leq Pr_{x\sim\mathcal{D}_x|x_{\ell_i}}\{\pi(x) = c\}$:

$$\left|E_f(\ell_i) - E_{\hat{f}_h}(\ell_i)\right| < \frac{1}{2k} + \sqrt{\frac{m}{2k\delta}} + \sqrt{\frac{m}{\delta}}\min\left\{\frac{1}{2}\sqrt{\sum_{c\in C}\left(Pr_{x\sim\mathcal{D}_x|x_{\ell_i}}\{\pi(x) = c\}\right)^2} \cdot \sqrt{E_{x\sim\mathcal{D}_x|x_{\ell_i}}[f(x)(1 - f(x))]}\right\}$$

Ignorando i primi due termini additivi (ricorda che possiamo scegliere k), se la distribuzione sui cluster $\rfloor \subseteq \mathcal{C}$ è approssimativamente uniforme, quindi il limite si azzera all'aumentare del numero di cluster, all'incirca un al rate di $1/\sqrt{\|C\|}$. Allo stesso modo come la varianza $E_{x\sim\mathcal{D}_x|x_{\ell_i}}[f(x)(1 - f(x))]$ va a zero, l'errore del classificatore deterministico si avvicina a zero per tutte le metriche m con alta probabilità.

Abbiamo mostrato che, con riferimento al paragrafo 2, l'affermazione fatta precedentemente era approssimativa. In questo caso anche se uno dei due: o A) punti materiali o B) stocasticità si avvicinano a zero, è lo stesso possibile trovare un classificatore deterministico su cui gli errori delle nostre metriche si avvicinano allo stesso modo a zero.

3.6 Orderliness

Fino ad adesso siamo riusciti a mostrare come l'approccio con metodo hash fornisca un migliore bound rispetto a quello standard ottenuto tramite un approccio threshold. Ora sposteremo l'attenzione su altri criteri per giudicare la qualità di approssimazione di classificatori deterministici di classificatori stocastici.
Gli approcci che abbiamo considerato finora possono essere ordinati in base a quanto siano "ordinabili". Dove il termine "ordine" è una nozione libera che misura quanto un classificatore sia "liscio" o "autoconsistente". Ma i classificatori analizzati fino ad ora quanto sono ordinabili? Procediamo per ordine: Il classificatore stocastico originale è il meno ordinato: potrebbe classificare lo stesso esempio in modo diverso,quando viene rilevato più volte. Il classificatore hashing è più ordinato perché è deterministico,e quindi fornirà sempre la stessa classificazione sullo stesso esempio. Tuttavia potrebbe comportarsi in modo diverso su esempi estremamente simili (se sono sottoposti a hash diversi). Il classificatore threshold è il più ordinato, poiché classificherà ogni esempio esattamente nello stesso modo, quindi simili esempi saranno molto probabilmente classificati in modo identico.

3.6.1 Repeated Use & Fairness

Come abbiamo precedentemente notoato, un classificatore stocastico può essere una cattiva scelta quando un utente può forzare il classificatore a fare più previsioni. Ad esempio, se un filtro antispam è stocastico, allora uno spammer potrebbe riuscire ad oltrepassare il controllo con un'email, semplicemente inviandola ripetutamente. Sostituendo un classificatore stocastico con uno deterministico potrebbe non bastare: un filtro antispam disordinato (anche deterministico) potrebbe essere superato inviando molte varianti dello stesso messaggio di spam (cambiando anche solo gli spazi bianchi).

Visto che misuriamo la qualità di un'approssimazione del classificatore stocastico in termini di metriche aggregate, ciò implica che stiamo osservando l'equità da un punto di vista statistico: anche se singolarmente i risultati sono casuali (o deterministici ma arbitrari), il classificatore potrebbe ancora essere considerato "equo" se si potesse dimostrare essere privo di pregiudizi sistematici (imposti tramite vincoli su gruppi usando metriche aggregate per l'equità). Come abbiamo mostrato nel Teorema 3, anche il limite prestazionale di un classificatore hashing migliora quanto più diventa disordinato (cioè come il numero di cluster in C e/o il numero di hash k aumenta), misurato in questi termini. Diversamente da questa prospettiva Dwork et al. [15] propone l'idea secondo cui "individui simili ricevono simili esiti ", che considera l'equità dal punto di vista di un individuo singolo. Questo principio è meglio seguito da classificatori più ordinati: lo sono le regioni decisionali di un classificatore threshold che è più equo se misurato con questo principio rispetto ad esempio di un classificatore hashing con regioni ristrette. Questa contrapposizione tra estremi: classificatori meno ordinati (rate metric precise) e classificatori più ordinati (individui simili, risultati

simili). Ci si chiede se esista una via di mezzo. Nella prossimo paragrafo "Clustering & Hashing" presentiamo un approccio che consente di ottenere un trade-off tra questi estremi. La realtà, ovviamente, è più complicata: ad esempio, le lotterie sono spesso considerate "giuste" dai partecipanti se ciascuno ritiene che il meccanismo sottostante sia equo, indipendentemente dai risultati individuali. In tali casi, il disordine, o anche la stocasticità, potrebbero essere desiderabili da un punto di vista di equità, e questa contrapposizione tra estremi svanisce.

3.6.2 Clustering + Hashing

La tecnica di hashing mostrata precedentemente ha un meccanismo "incorporato" per affrontare (almeno parzialmente) la mancanza di ordine caratteristica del suo metodo: il pre-clustering. Se $\pi : \mathcal{X} \to \mathcal{C}$ assegna elementi "simili" $x, x' \in \mathcal{X}$ allo stesso cluster $c \subseteq C$, quindi tali elementi avranno hash identici e i valori del classificatore stocastico $f(x), f(x')$ pertanto sottoposti alla soglia con lo stesso valore. Quindi, supponendo che il classificatore stocastico f sia "fluido" e con un'opportuna scelta di Π, il risultato deterministico $\hat{f}$ può essere considerato "localmente ordinato" e pertanto soddisferà una forma di input simili, simili target e fornire una certa protezione contro l'uso ripetuto. Sfortunatamente, questo approccio presenta alcuni svantaggi. In primo luogo, l'onere è su chi effettua la pratica di progettare la funzione di clustering Π in modo tale da catturare la nozione appropriata di somiglianza. Ad esempio, se si desidera codificare una nozione intuitiva di equità, allora si collocano le istanze in diversi cluster (e sono quindi trattati in modo incoerente da $\hat{f}$) dovrebbe essere abbastanza distintoda rendere questo incarico giustificabile. In secondo luogo, si dovrebbe osservare che il limite del Teorema 3 è migliore quando ci sono più cluster, e peggiore quando ce ne sono pochi. Quindi, c'è un compromesso tra ordine e prestazioni: se si deve raggiungere un livello richiesto di accuratezza di rate metric, allora fare così potrebbe costringere uno a usare così tanti cluster tali per cui non vi sia sufficiente ordine locale.

3.7 Stochastic Ensembles

Ora ci focalizziamo su un particolare tipo di classificatore stocastico che seleziona casualmente da un numero finito di classificatori deterministici di base. Questo tipo di classificatore viene utilizzato in diversi algoritmi di ottimizzazione vincolata. Sia un *stochastic ensemble* $f : \mathcal{X} \to [0, 1]$ definito in termini di n classificatori deterministici $\hat{g}_1, \ldots, \hat{g}_n : \mathcal{X} \to \{0, 1\}$ e una distribuzione di probabilità associata $p \in \Delta^{n-1} \subseteq R^n$, per il quale $f(x) := \sum_{j=1}^{n} p_j \hat{g}_j(x)$. In modo da poter valutare il classificatore su un esempio x, facendo come primo esperimento un indice $j \in [n]$ seguendo la distribuzione p e le predizioni $\hat{g}_j(x)$.
L'approccio hash mostrato precedentemente può essere applicato agli stochastic ensembles, ma a causa della speciale struttura di questi modelli, è possibile fare ancora meglio. Qui, proponiamo una strategia alternativa che innanzitutto applica clustering, e poi sudddivide ogni cluster in n raggruppamenti, per i quali l'i-esimo raggruppamento contiene all'incirca una porzione p_i delle istanze del cluster, e assegna tutte le istanze all'i-esimo raggruppamento del classificatore $\hat{g}_j(x)$. Facciamo ciò usando una predefinita funzione score q e un parametro casuale di shift r_c per ogni cluster c. Il beneficio di questo approccio è di riuscire a bilanciare la dimensione dei raggruppamenti a seconda della distribuzione di probabilità p, permettendoci di ridurre il numero di raggruppamenti e quindi una maggiore ordine locale, in confronto al classificatore hash (che si basa su un numero elevato di raggruppamenti di dimensioni approssimativamente uguali). Questo approccio si chiama *approccio variable binning* .

Teorema 4 (Variable binning)**.** *Sia $f : \mathcal{X} \to [0, 1]$ un classificatore stocastico e D_x la distribuzione di $\mathcal{X}$. Essendo date m metriche $(\ell_i, \mathcal{X}_{\ell_i})$ per $i \in [m]$, ognuna di esse potenzialmente associata ad una differente distribuzione target condizionale $\mathcal{D}_{y_i|x}$. Sia scelto $\pi : \mathcal{X} \to \mathcal{C}$ per essere una funzione che pre-assegna le instanze al cluster e $q : \mathcal{X} \to [0, 1]$ come funzione score predefinita. Si scelga $p_{i.0} = 0$ e si denoti $p_{:j} = p_1 + \ldots + p_j, \forall j \in [n]$. Definendo $clip(z) = z - \lfloor z \rfloor$. Campionando $\|C\|$ numeri casuali $r_1, \ldots, r_{|C}$ indipendenti e uniformemente distribuiti da $[0, 1)$ e definito il classificatore deterministico $\hat{f}(x) = \sum_{j=1}^{n} s_j(x)\hat{g}_j(x)$ dove $s : \mathcal{X} \to \{0, 1\}^n$ seleziona uno dei n classificatori ed è dato da:*

$$s_j(x) = \sum_{c \in C} 1 \left\{ \pi(x) = c, clip\left(q(x) + r_c\right) \in [p_{:j-1}, p_{:j}) \right\}$$

Poi, con probabilità $1 - \delta$ sui campionamenti $r_1, \ldots, r_{|C}$:

$$\left| E_f\left(\ell_i\right) - E_{\hat{f}}\left(\ell_i\right) \right| < \left(\frac{m}{\delta} \sum_{c \in \mathcal{C}} \left(\left(Pr_{x \sim \mathcal{D}_x | x_{\ell_i}} \{ \pi(x) = c \} \right)^2 \right. \right.$$

$$\left. \left. \times E_{x \sim \mathcal{D}_x | x_{\ell_i}} [f(x)(1 - f(x)) \mid \pi(x) = c] \right) \right)^{\frac{1}{2}}$$

dove $\mathcal{D}_{x|\mathcal{X}_{\ell_i}}$ è la distribuzione $\mathcal{D}_x$ ristretta a $\mathcal{X}_{\ell_i}$.

Il limite soprastante è simile al limite per l'hash del Teorema 3, tranne per il fatto che non contiene termini che dipendono dal numero di bucket di hash k ed è quindi un leggero miglioramento. Nei nostri esperimenti, lo troviamo per abbinare le prestazioni dell'hash con un maggior ordinamento locale.

3.8 Implementazione algoritmo

3.8.1 Codice

Di seguito verrà riportato il codice di un esempio concreto di approssimazione di classificatore stocastico con un classificatore deterministico. L'approccio utilizzato sarà quello hash: "Hashing function to make a stochastic classifier deterministic."

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import hashlib
from absl import app
import numpy as np


def compute_hash(features, hash_matrix, hash_vector):
  """Compute hash values for features using the hash function (A * x + c) mod 2.
  Args:
    features: NumPy float array of shape (n, d), the features to hash.
    hash_matrix: NumPy float array of shape (num_feature_bits, num_hash_bits),
      a random matrix A to construct the hash function.
    hash_vector: NumPy float array of shape (1, num_hash_bits),
      a random vector c to construct the hash function.
  Returns:
    NumPy float array of shape (n, 1) containing the hashed values in [0, 1].
  """
  # Helper function to convert an int array to a bit string array.
  def convert_int_to_bin(x, dimension):
    # Converts x to an array of bit strings of size dimension.
    return '{:b}'.format(x).zfill(dimension)[-dimension:]
  convert_int_to_bin = np.vectorize(convert_int_to_bin)

  # Helper function to convert a bit string array to an into array.
  convert_bin_to_int = np.vectorize(lambda x: int(x, 2))

  # Number of features and hash bits.
  num_features = features.shape[0]
  num_feature_bits, num_hash_bits = hash_matrix.shape

  # Concatenate features and apply MD5 hash to get a fixed length encoding.
  feature_sum_str = [''.join(x) for x in features.astype('str')]
```

```
  feature_sum_hex = [hashlib.md5(s.encode('utf-8')).hexdigest()
                     for s in feature_sum_str]
  feature_sum_int = [int(h, 16) for h in feature_sum_hex]

  # Binarize features
  feature_sum_bin = convert_int_to_bin(
      feature_sum_int, dimension=num_feature_bits)
  feature_sum_bin_matrix = np.array(
      [[int(c) for c in s] for s in feature_sum_bin])

  # Compute hash (Ax + c) mod 2.
  feature_hashed = (
      np.dot(feature_sum_bin_matrix, hash_matrix) +
      np.repeat(hash_vector, repeats=num_features, axis=0))
  feature_hashed_bits = np.mod(feature_hashed, 2)

  # Convert hash to bit string.
  feature_hashed_bit_char = convert_int_to_bin(feature_hashed_bits, 1)
  feature_hashed_bit_str = [''.join(s) for s in feature_hashed_bit_char]
  feature_hashed_int = convert_bin_to_int(feature_hashed_bit_str)
  hashed_val = feature_hashed_int * 1. / 2 ** num_hash_bits

  # Return normalized hashed values in [0, 1].
  return hashed_val.reshape(-1, 1)


def main(argv):
  """Example usage of hash function."""
  del argv

  num_feature_bits = 128
  num_hash_bits = 32

  # Random hash matrix and vector to construct hash function.
  hash_matrix = (np.random.rand(
      num_feature_bits, num_hash_bits) > 0.5).astype('int')
  hash_vector = (np.random.rand(1, num_hash_bits) > 0.5).astype('int')

  # Generate random features.
  num_examples = 10
  dimension = 4
  features = np.random.normal(size=(num_examples, dimension)).astype(np.float32)
```

```
  # Compute hash.
  hash_val = compute_hash(features, hash_matrix, hash_vector)

  print('Feature matrix:')
  print(features)
  print('\nHashed values:')
  print(hash_val)


if __name__ == '__main__':
  app.run(main)
```

3.8.2 Analisi

In questo paragrafo ci occuperemo di analizzare e descrivere il codice. Il codice ci mostra come rendere deterministico un classificatore stocastico sfruttando la funzione hash. Il procedimento può essere suddiviso nelle seguenti fasi: **1)** settare il numero di bit delle features e il numeri di bit dell'hash **2)** costruire la matrice hash e il vettore di hash che ci permetteranno di costruire la funzione hash **3)** il classificatore stocastico (le random features) viene generato in maniera randomica **4)** il cuore dell'algoritmo in cui viene calcolato l'hash a partire dal classificatore stocastico generato al punto precedente. Nel calcolo degli hash values dalle features viene usata la funzione hash $(A * x + c) mod 2$.

Vengono definite due funzioni: *compute_hash* e *convert_int_to_bin* . La seconda è una semplice funzione di sostegno che permette di convertire un array di interi in un array di bit di stringhe scegliendo opportunamente la dimensione. Essa riceve in input x e *dimension* che sono il vettore da convertire e appunto la dimensione che si vuole ottenere. E restituisce come output l'array convertito $result = convert\_int\_to\_bin(x, dimension)$. La funzione *compute_hash* è la funzione cardine dello script e si occupa di calcolare i valori hash partendo dalle features e usando la funzione hash $(A*x+c) mod 2$. Essa restituisce un vettore contenente gli hash values nell'intervallo $[0, 1]$ dando in input le features che riceveranno l'hash, una matrice hash per costruire la funzione hash e un vettore hash costruito in maniera random per costruire la funzione hash. $hash\_val = compute\_hash(features, hash\_matrix, hash\_vector)$

L'implementazione concreta nel nostro script è quindi così strutturata:

Fase 1) viene settato nel nostro esempio a 128 e 32 il numero di bit rispettivamente delle features e dell'hash.

Fase 2) avviene l'estrazione dei valori random (presi da distribuzione normale) per definire la matrice hash $[num\_feature\_bits, num\_hash\_bits]$ e il vettore hash $[1, num\_hash\_bits]$ che permetteranno di costruire la funzione hash.

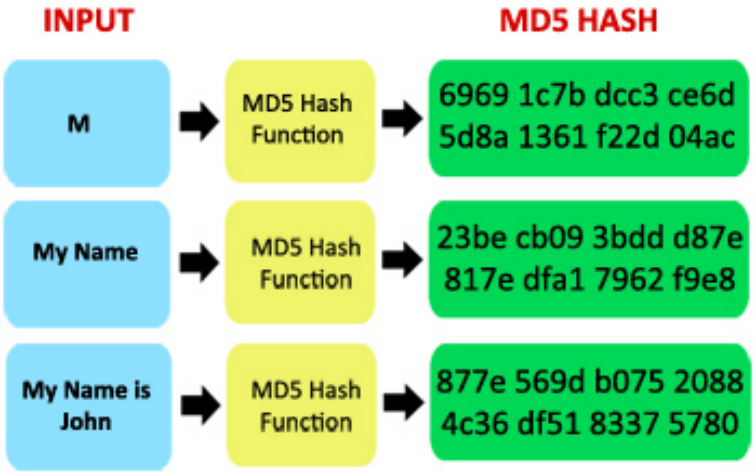
Fase 3) vengono generate in maniera casuale le features dopo aver settato le dimensioni a 10 (numero di esempi) e 4 (dimensioni).

Fase 4) si calcola l'hash. Inizialmente si usa la funzione di sostegno
$result = convert\_int\_to\_bin(x, dimension)$ per la conversione di un array di interi in un array di bit di stringhe, poi si settano il numero di features e di bit hash ai valori richiesti/voluti. Si concatenano le features e si applica MD5 hash per ottenere una codifica di lunghezze fissa. In seguito dopo aver reso binarie le features si calcola l' hash utilizzando la funzione $(Ax + c) mod 2$. Il comando $np.mod(feature_hashed, 2)$permettere di fare li modulo 2 e il comando$np.dot(feature_sum_bin_matrix, hash_matrix)$ +

$np.repeat(hash\_vector, repeats = num\_features, axis = 0)$ permette di calcolare $(Ax+c)$. Si converte l'hash ad una stringa di bit e si restituiscono i valori hash normalizzati nell'intervallo $[0, 1]$. Dal punto di vista matriciale partendo da una matrice delle features $[10]\times[4]$ si ottiene un vettore colonna contenente i valori hash di dimensione $[10]\times[1]$.

Breve excursus su funzione MD5 hash

L'MD5 [14] è una funzione hash crittografica realizzata da Ronald Rivest nel 1991 e standardizzata con la RFC 1321. È una funzione unidirezionale diversa dalla codifica e dalla cifratura perché irreversibile. Questa funzione prende in input una stringa di lunghezza arbitraria e ne produce in output un'altra a 128 bit.
[1]



[1] https://www.gohacking.com/what-is-md5-hash/ by GHadmin

3.8.3 Output

Di seguito riporto due esempi di output.

Esempio 1

```
C:\Users\susim\anaconda3\envs\PROBUS\python.exe C:/Users/susim/Desktop/PROBUS/hashing.py
Feature matrix:
[[ 1.5316728   0.98732394 -0.54588324  1.2429447 ]
 [ 0.03132028 -1.7138846  -0.68816185  0.42396426]
 [-1.0276923  -0.6888094   0.849453    1.1258888 ]
 [ 0.3748992   0.36284897 -1.015902    2.002314  ]
 [-0.5599782  -0.85117877  0.13552833 -0.8046104 ]
 [ 0.32519066  0.47499204 -0.8668793  -1.0938522 ]
 [ 0.5978385  -0.79568577  0.23241848 -0.48803192]
 [-0.4091363   1.3950428  -1.0487597   1.350318  ]
 [-0.5476029  -1.6994717   0.37357715  1.996468  ]
 [-0.38142103 -1.0314503   0.31825745 -1.4260862 ]]

Hashed values:
[[0.86431425]
 [0.20551744]
 [0.79040122]
 [0.71041611]
 [0.51997374]
 [0.99208487]
 [0.64448005]
 [0.75244336]
 [0.90780729]
 [0.32854648]]

Process finished with exit code 0
```

Le features vengono scelte in maniera random estraendo valori dalla distribuzione normale. Per esempio il comado np.random.normal si utilizza nel seguente modo per estrarre un valore da una distribuzione di probabilità normale gaussiana di media 0 e varianza 0,1

```
mu, sigma = 0, 0.1 # mean and standard deviation
s = np.random.normal(mu, sigma, 1000)
```

E gli hashed values vengono calcolati seguendo la funzione hash $(A * x + c) mod 2$. E saranno valori nell'intervallo $[0, 1]$.

Esempio 2

```
C:\Users\susim\anaconda3\envs\PROBUS\python.exe C:/Users/susim/Desktop/PROBUS/hashing.py
Feature matrix:
[[ 1.1010432  -0.3490139   1.3369502  -0.47640717]
 [ 0.5260095   1.1258053  -0.3806545  -0.20056555]
 [-0.9336634  -0.9722095  -0.27131858  0.72952074]
 [ 0.34521544 -0.364367   -0.9277071   0.7954069 ]
 [ 0.6766685   0.7043259   0.20502117  1.1920146 ]
 [-0.8905546   0.66671443  1.6314458   1.0025772 ]
 [-2.231158   -0.03358049 -1.0390402  -0.25778934]
 [ 1.6776731  -1.3090858  -1.65893    -0.08726081]
 [ 1.6485438  -1.7100762   0.92251956 -0.24562421]
 [ 2.3360853  -1.4318976  -1.0093176  -1.4485496 ]]

Hashed values:
[[0.87557007]
 [0.70384175]
 [0.04275874]
 [0.27131199]
 [0.79285429]
 [0.42554853]
 [0.18557719]
 [0.9828957 ]
 [0.83503139]
 [0.10391375]]

Process finished with exit code 0
```

Qui un secondo esempio di come l'hash abbia valori completamente diversi dai precedenti. Il tutto a conferma del fatto che seppur in maniera deterministica riusciamo a riprodurre un comportamento stocastico sfruttando la funzione hash per costruire un classificatore deterministico a partire da un classificatore stocastico ricalcolato in maniera randomica ogni volta con valori differenti.

Capitolo 4

Esperimenti

4.1 Problem Setting

Ora ci occupiamo di valutare le diverse strategie descritte precedentemente con lo scopo di individuare le migliori approssimazioni del classificatore stocastico a seconda del classificatore deterministico scelto. Considereremo training tasks vincolati con due diversi obbiettivi di equità:

- ROC Curve Matching.
- Matching regression histograms tra gruppi protetti

Questi obiettivi impongono un gran numero di vincoli al modello e le soluzioni stocastiche diventano cruciali per riuscire a soddisfarle. Abbiamo usato il proxy-lagrangiano ottimizzatore di Cotter et al. [5], [6] per risolvere il problema di ottimizzazione vincolata. Questo solutore genera stochastic ensemble, nonché il miglior classificatore deterministico, scelto euristicamente trai i suoi iterati.

Datasets: Useremo vari datasets di correttezza con attributi binari protetti. Abbiamo usato modelli lineari per tutti gli esperimenti: (1) $COMPAS$[25] dove l'obiettivo è prevedere la recidiva con il genere come attributo protetto; (2) *Communities & Crime* dove l'obiettivo è prevedere se una comunità negli Stati Uniti ha un tasso di criminalità superiore al 70 percentile; (3) *LawSchool* dove il compito è prevedere se lo studente scolastico supererà l'esame di laurea; (4) $UCIAdult$ dove il compito è prevedere se il reddito di una persona supera i 50.000 / anno, con candidati femminili come gruppo protetto; (5) $WikiToxicity$ [24]dove l'obiettivo è prevedere se un commento pubblicato su una pagina di discussione di Wikipedia contiene contenuti spam/tossici/ accettabili, con i commenti che contengono il termine "Gay"cosiderati come gruppo protetto; (6) $BusinessEntityResolution$ un dataset proprietario di una grande società di servizi Internet, dove il compito è prevedere se una coppia di descrizioni aziendali si riferiscono alla stessa attività reale, con attività non a catena trattate come protette.

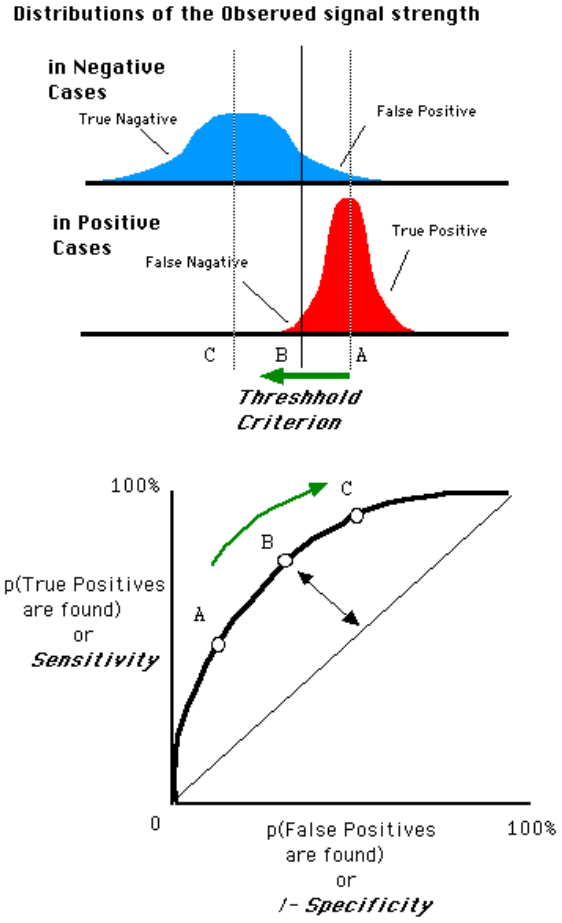
Dataset	No. of instances	No. of features	No. of classes
COMPAS	4,073	31	2
Communities & Crime	1,495	135	2
Law School	15,388	36	2
Adult	32,561	122	2
Wiki Toxicity	95,692	100	2
Business	11,560	36	2
Abalone	4,177	12	10

Metodi: Useremo le tecniche di thresholding, hashing e variable binning (VarBin) per convertire un stochastic ensemble in un classificatore deterministico. Per fare l'hashing faremo prima il mapping degli input features ai 2^{18} clusters (usando a 128 bit funzione hash crittografica) e applicheremo la funzione hash indipendente a coppie per mappare 2^{32} gruppi. Per VarBin noi scegliamo una direzione β uniformemente a caso dalla sfere di lato $\updownarrow^2$,proietta istanze in questa direzione, e ha il mapping π dei cluster che divide in $k = 25$ continui bins. Ecco come, $\pi(x) = c$ quando $u_{c-1} \leq \langle \beta, x \rangle \leq u_c$ con $u_0 = \min_x \langle \beta, x \rangle < u_1 < \ldots < u_{25} = \max_x \langle \beta, x \rangle$ sono soglie equamente distribuite. La funzione score $q(x)$ per un'istanza x è scelta per essere il valore previsto $\langle \beta, x \rangle$ normalizzato per essere compreso tra il minimo e il massimo valore all'interno del cluster $q(x) = \frac{\langle \beta, x \rangle - u_\pi(x) - 1}{u_\pi(x) - u_\pi(x) - 1}$. Inoltre, aggiungendo i numeri random $r_1, \ldots, r_{|\mathcal{C}|}$ è non necessario e prende r_c per tutti i c, che semplifica di molto l'implementazione di VarBin.

4.2 ROC Curve Matching

4.2.1 Analisi

Nella teoria delle decisioni, le curve ROC (Receiver Operating Characteristic) sono degli schemi grafici per un classificatore binario. Lungo i due assi si possono rappresentare la sensibilità e (1-specificità), rispettivamente rappresentati da True Positive Rate (TPR, frazione di veri positivi) e False Positive Rate (FPR, frazione di falsi positivi). In altre parole, si studiano i rapporti fra allarmi veri (hit rate) e falsi allarmi. La curva ROC viene creata tracciando il valore del True Positive Rate (TPR, frazione di veri positivi) rispetto al False Positive Rate (FPR, frazione di falsi positivi) a varie impostazioni di soglia. Il tasso di veri positivi è anche noto come sensibilità, richiamo o probabilità di rilevazione. Il tasso di falsi positivi è anche noto come fall-out o probabilità di falsi allarmi e può essere calcolato come (1 - specificità).



1

[1]NekoJaNekoJa commonswiki autore secondo quanto affermano i diritti di autore. https://commons.wikimedia.org/w/index.php?curid=407628

Può anche essere pensato come un diagramma della potenza in funzione dell'errore di tipo I :quando la prestazione viene calcolata da un solo campione della popolazione, può essere considerata come una stima di queste quantità. La curva ROC è quindi il tasso dei veri positivi in funzione del tasso dei falsi positivi. In generale, se sono note le distribuzioni di sensibilità e 1-specificità, la curva ROC può essere generata tracciando la funzione di distribuzione cumulativa della probabilità di rilevamento nell'asse y rispetto alla funzione di distribuzione cumulativa della probabilità di falso allarme sull'asse x.
Il nostro primo compito è quello di formare un modello a punteggio che produca curve ROC simili sia per i gruppi protetti che per la popolazione complessiva. Sia TPR_t il true positive rate nella curva ROC del modello se la soglia arriva a un rate t di falsi positivi e sia TPR_t^{ptr} il true positive rate raggiunto sui gruppi protetti quando la soglia è lo stesso valore per raggiungere gli stessi false positive rate t sui gruppi protetti. Siamo interessati a un set selezionato di $FPRs$ nella porzione iniziale della curva: $\tau = \{0.1, 0.2, 0.3, 0.4\}$. Il nostro obiettivo è di massimizzare la somma di $TPRs$ a questi $FPRs$, soggetti a valori TPR simili sia per i gruppi prottetti sia per la popolazione complessiva:

$$\max \sum_{t \in \mathcal{T}} \mathrm{TPR}_t s.t. \left| \mathrm{TPR}_t - \mathrm{TPR}_t^{ptr} \right| \leq 0.01, \forall t \in \mathcal{T}$$

Questo risultato in sottostante a 24 vincoli di true e false positive rates. Per questo problema, gli output dell'ottimizzatori vincolati ensembles con 3-5 classificatori deterministici.

Tabella 1: violazioni oggettive e vincolate per i modelli stocastici addestrati

	Crime (4)		COMPAS (5)		Law School (5)	
	Train	Test	Train	Test	Train	Test
Unconstrained	0.917 (0.14)	0.893 (0.13)	0.585 (0.06)	0.586 (0.07)	0.805 (0.22)	0.784 (0.26)
Stochastic	0.863 (0.01)	0.834 (0.10)	0.584 (0.01)	0.583 (0.06)	0.684 (0.02)	0.675 (0.06)
Majority	0.870 (0.02)	0.846 (0.12)	0.585 (0.02)	0.586 (0.05)	0.802 (0.14)	0.773 (0.17)
Hashing	0.862 (0.01)	0.830 (0.09)	0.585 (0.01)	0.588 (0.09)	0.680 (0.02)	0.673 (0.03)
VarBin	0.861 (0.01)	0.833 (0.11)	0.583 (0.02)	0.582 (0.07)	0.684 (0.02)	0.675 (0.05)
Best Iterate	0.871 (0.02)	0.843 (0.12)	0.591 (0.03)	0.585 (0.07)	0.761 (0.11)	0.730 (0.10)
	Adult (3)		Wiki Toxicity (4)		Business (3)	
	Train	Test	Train	Test	Train	Test
Unconstrained	0.866 (0.11)	0.858 (0.08)	0.842 (0.07)	0.840 (0.06)	0.863 (0.03)	0.865 (0.06)
Stochastic	0.829 (0.01)	0.826 (0.03)	0.867 (0.01)	0.860 (0.03)	0.846 (0.01)	0.843 (0.04)
Majority	0.831 (0.04)	0.831 (0.06)	0.892 (0.04)	0.883 (0.06)	0.861 (0.02)	0.857 (0.05)
Hashing	0.825 (0.02)	0.824 (0.04)	0.867 (0.01)	0.863 (0.04)	0.846 (0.01)	0.844 (0.04)
VarBin	0.829 (0.02)	0.828 (0.05)	0.881 (0.02)	0.873 (0.03)	0.846 (0.02)	0.842 (0.05)
Best Iterate	0.831 (0.04)	0.831 (0.06)	0.898 (0.04)	0.891 (0.05)	0.861 (0.02)	0.857 (0.05)

Nella precedente tabella confrontiamo le strategie di de-randomizzazione su compiti di abbinamento ROC. Per ogni metodo riportiamo la somma TPR : $\sum_{t\in\mathcal{T}} \mathrm{TPR}_t$ (più alta è meglio è) e la massima violazione del vincolo $\max_{t\in\mathcal{T}} \left|\mathrm{TPR}_t - \mathrm{TPR}_t^{ptr}\right|$. Non vincolato si riferisce a un classificatore deterministico addestrato per ottimizzare il rate di errore senza vincoli. Il miglior iterato è il classificatore deterministico scelto euristicamente dal risolutore di ottimizzazione vincolata dai suoi iterati. Il numero di modelli di base in supporto del modello stocastico è mostrato tra parentesi. La soluzione stocastica produce una violazione del vincolo molto più bassa rispetto a un classificatore non vincolato addestrato per ottimizzare il rate di errore e il classificatore deterministico "best iterate". Nella seguente tabella 2 svolgiamo un confronto tra le diverse strategie per de-randomizzare il modello stocastico addestrato.

Tabella 2: De-randomizzare il modello stocastico addestrato

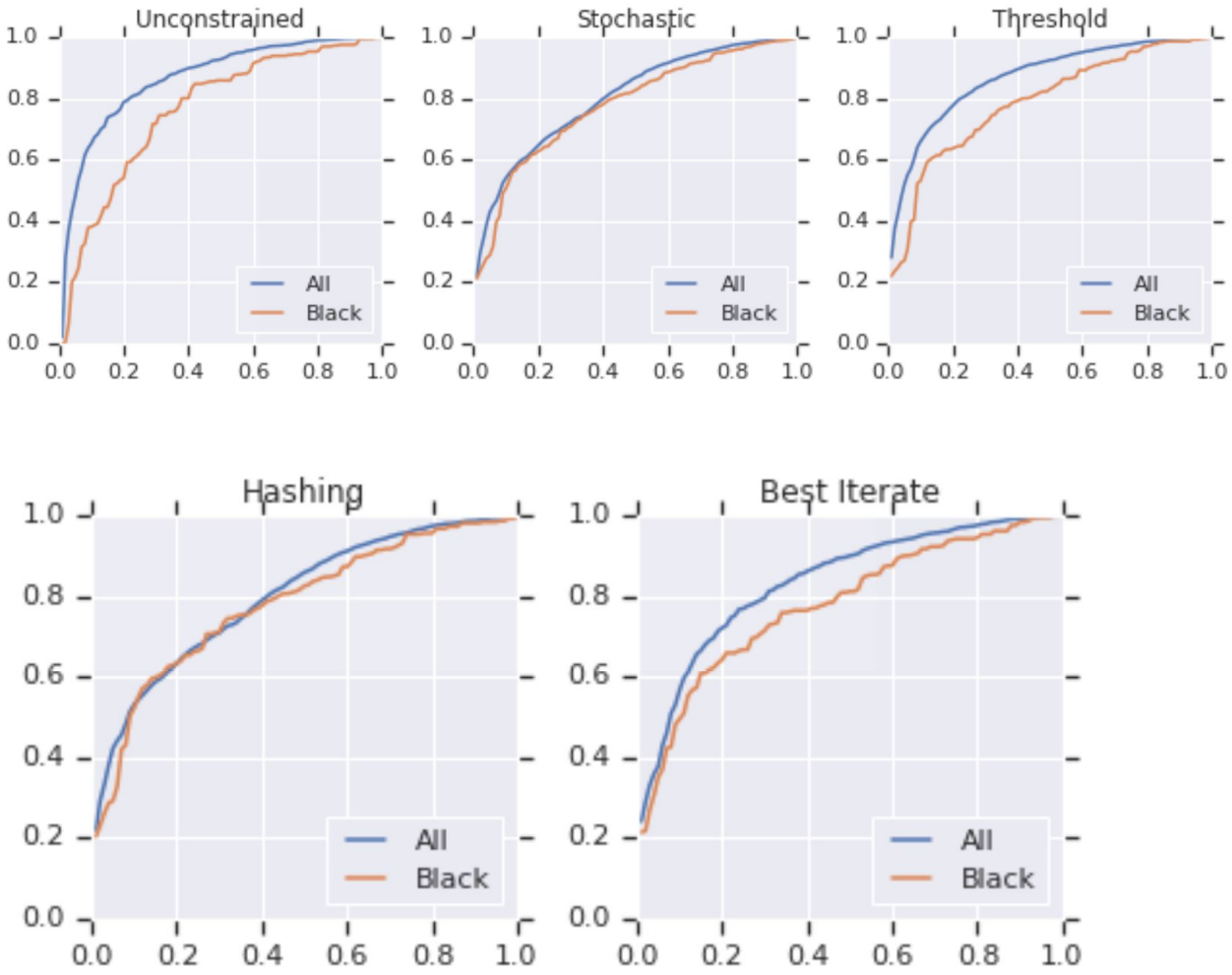
	Crime (4)		COMPAS (5)		Law School (5)	
	Train	Test	Train	Test	Train	Test
Threshold	0.007 (0.01)	0.012 (0.03)	0.002 (0.01)	0.002 (0.00)	0.118 (0.12)	0.099 (0.11)
Hashing	0.001 (0.00)	0.004 (0.01)	0.001 (0.01)	0.005 (0.03)	0.004 (0.01)	0.001 (0.03)
VarBin	0.002 (0.00)	0.000 (0.02)	0.001 (0.01)	0.002 (0.02)	0.000 (0.01)	0.000 (0.02)
	Adult (3)		Wiki Toxicity (4)		Business (3)	
	Train	Test	Train	Test	Train	Test
Threshold	0.002 (0.04)	0.005 (0.03)	0.025 (0.04)	0.024 (0.03)	0.015 (0.02)	0.014 (0.01)
Hashing	0.005 (0.01)	0.002 (0.01)	0.000 (0.01)	0.004 (0.01)	0.000 (0.01)	0.001 (0.02)
VarBin	0.000 (0.01)	0.002 (0.01)	0.014 (0.01)	0.013 (0.01)	0.000 (0.01)	0.001 (0.01)

L'hashing e VarBin sono capaci di avvicinarsi alle performance del classificatore stocastico. Il thresholding si dimostra non buono per 3 dei 6 datasets a disposizione.

4.2.2 Studio grafico

Nella figura sottostante forniamo una rappresentazione visiva degli incroci delle ROC curve.

Figura 1: ROC curve

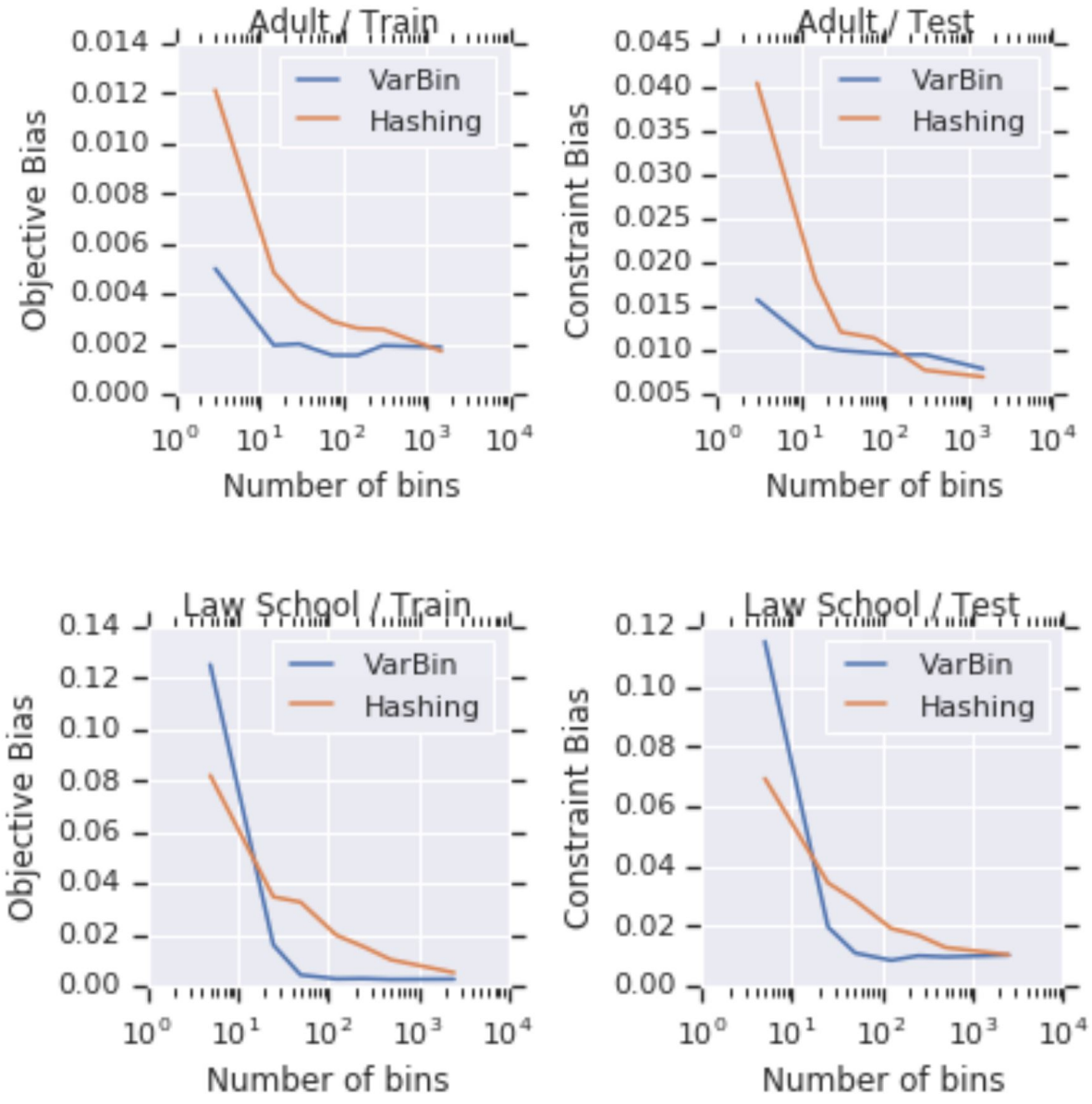


Test impostato su curve ROC per il gruppo di pelle di colore e della popolazione complessiva presso la Law School (dataset). Si noti che il classificatore stocastico corrisponde correttamente alle due curve ROC e l'approssimazione hash è molto più fedele del miglior iterato deterministico fornito dal risolutore.

Poi si passa allo studio dei trade-off tra orderliness e precisione. Per poter valutare l'hashing con molti bins, costruiamo gli input su direzioni random, formando bins equidistanti e poter fare gli hash sugli indici dei bins. Ora analizziamo graficamente la differenza tra i modelli stocastici e hash-deterministico per numero diverso di bins (vedi figura 2). Con media di oltre 50 estrazioni casuali in direzioni casuali e funzione

hash. Confronto tra Hashing pre-cluster e VarBin che mostra il compromesso tra ordine (utilizzando il proxy di un numero minore di bin) e l'accuratezza delle metriche dei rate (più bin).

Figura 2: Hashing pre-cluster e VarBin che mostra il compromesso tra ordine e accuratezza



Confrontiamo l'hash con las trategia VarBin che utilizza lo stesso numero di bin (totali). VarBin è generalmente migliore nell'approssimazione del classificatore stocastico con un piccolo numero di bin perché VarBin dimensiona i bin per rispettare la distribuzione di probabilità p, ed è quindi in grado di fornire una migliore precisione con più ordine.

4.3 Matching regression histograms

4.3.1 Analisi

Ora consideriamo il compito di regressione dove l'obbiettivo di equità è di creare una corrispondenza tra l'output della distribuzione del modello per il gruppo protetto e la totalità della popolazione. Per un modello di regressione $\hat{g} : \mathcal{X} \to \mathcal{Y}$, con dei bound $\mathcal{Y} \subset R$, dividiamo il range dell'output in 10 bins equamente dimensionati $B_1, \ldots, B_{10}$ e richiede che la frazione dei membri dei gruppi protetti in un bin sia vicina alla frazione della totalità della popolazione nel bin: $|Pr_x|\, ptr\, \{\hat{g}(x) \in B_j\} - Pr_x\, \{\hat{g}(x) \in B_j\}\ | \leq 0.01$ per tutti i$j \in [10]$. Minimizziamo l'errore quadratico soggetto al raggiungimento di questo obiettivo, che si traduce in un totale di 20 vincoli sul modello. Formiamo modelli stocastici sugli stessi dataset di prima, e utilizziamo etichette di valore reale ove disponibili: per Crime prevediamo il tasso di criminalità pro capite, per la Law School, prevediamo il GPA non laureato e per WikiToxicity prevediamo il livello di tossicità (un valore nell'intervallo $[0, 1]$). In questo caso, l'ottimizzatore vincolato manda in output un stochastic ensemble di modello di regressione $\hat{g}_1, \ldots, \hat{g}_n : \mathcal{X} \to \mathcal{Y}$ con probabilità $p \in \Delta^{n-1}$. Nel posto del thresholding riportiamo la "media" che genera semplicemente il valore atteso d'insieme: $\hat{f}(x) = \sum_{j=1}^{n} p_j \hat{g}_j(x)$. Per i nostri dataset, l'addestramento dello stochastic ensembles contiene da 4 a 8 classificatori. Nella tabella 3 riportiamo le violazioni oggettive e vincolate.

Tabella 3: Confronto di strategie di de-randomizzazione su istogrammi corrispondenti a compiti di regressione

	Crime (5)		COMPAS (4)		Law School (5)	
	Train	Test	Train	Test	Train	Test
Unconstrained	0.016 (0.57)	0.020 (0.57)	0.202 (0.14)	0.203 (0.15)	0.014 (0.00)	0.014 (0.04)
Stochastic	0.041 (0.02)	0.046 (0.07)	0.300 (0.01)	0.299 (0.02)	0.325 (0.01)	0.323 (0.01)
Average	0.041 (0.03)	0.045 (0.08)	0.232 (0.02)	0.231 (0.06)	0.061 (0.01)	0.061 (0.02)
Hashing	0.041 (0.03)	0.046 (0.07)	0.298 (0.03)	0.303 (0.06)	0.323 (0.01)	0.320 (0.02)
VarBin	0.041 (0.03)	0.047 (0.07)	0.300 (0.06)	0.293 (0.05)	0.327 (0.02)	0.325 (0.05)
Best Iterate	0.037 (0.09)	0.042 (0.11)	0.337 (0.06)	0.335 (0.07)	0.517 (0.03)	0.518 (0.03)
	Adult (4)		Wiki Toxicity (5)		Business (8)	
	Train	Test	Train	Test	Train	Test
Unconstrained	0.106 (0.35)	0.107 (0.34)	0.057 (0.15)	0.058 (0.15)	0.127 (0.35)	0.129 (0.31)
Stochastic	0.125 (0.02)	0.127 (0.02)	0.091 (0.07)	0.090 (0.09)	0.327 (0.01)	0.329 (0.02)
Average	0.122 (0.02)	0.124 (0.02)	0.068 (0.16)	0.067 (0.18)	0.236 (0.06)	0.239 (0.06)
Hashing	0.126 (0.02)	0.127 (0.01)	0.090 (0.07)	0.089 (0.08)	0.337 (0.03)	0.342 (0.07)
VarBin	0.125 (0.02)	0.128 (0.02)	0.093 (0.06)	0.093 (0.09)	0.328 (0.05)	0.324 (0.05)
Best Iterate	0.120 (0.05)	0.122 (0.05)	0.072 (0.08)	0.071 (0.10)	0.407 (0.13)	0.412 (0.11)

Per ogni metodo, riportiamo l'obiettivo di errore quadratico (inferiore è meglio) e il vincolo massimo di violazione tra i bins. Non vincolato si riferisce a un classificatore deterministico addestrato per ottimizzare il rate di errore senza vincoli. Il miglior iterato

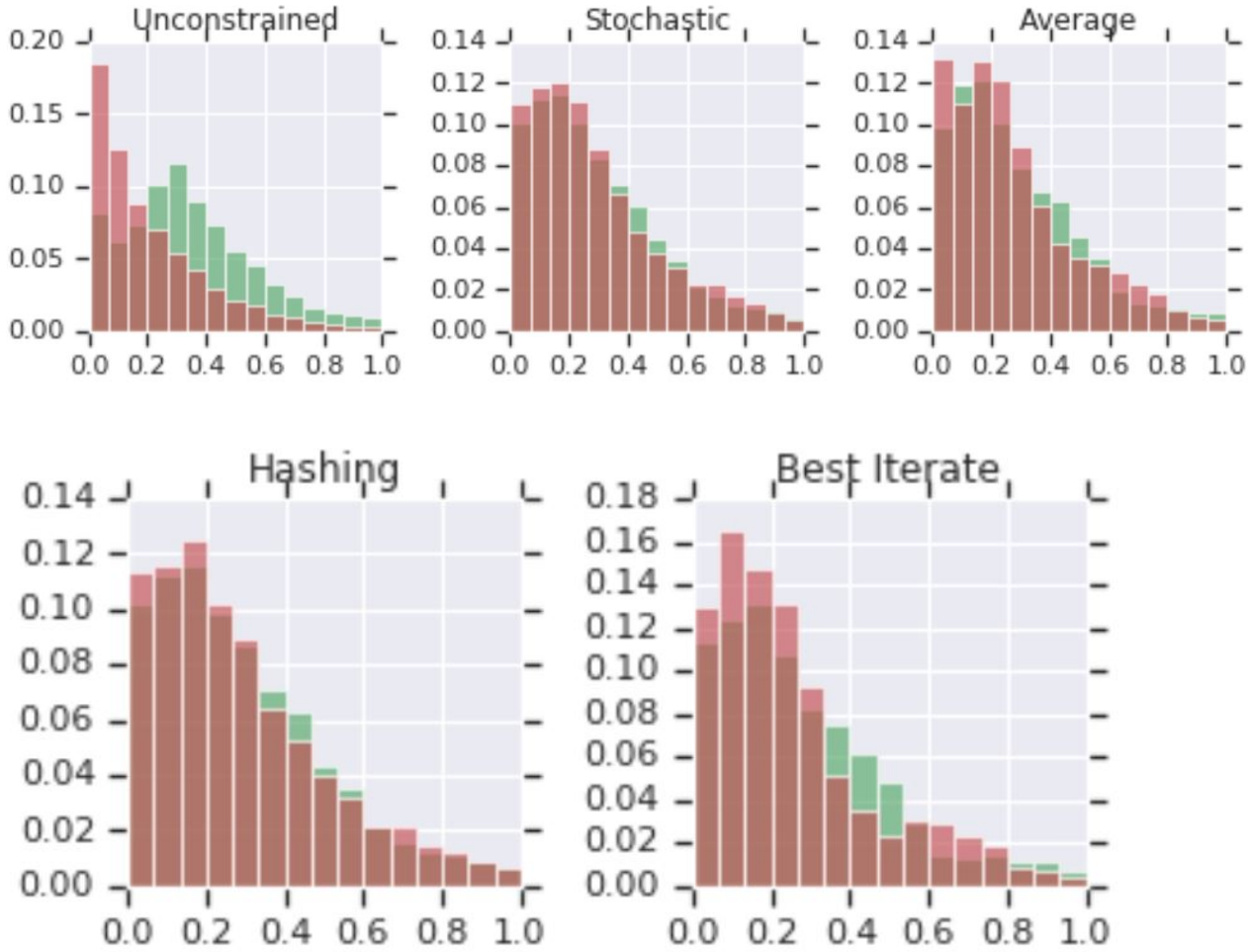
è il classificatore deterministico scelto euristicamente dal risolutore di ottimizzazione vincolata tra i suoi iterati. Una valutazione di quanto bene i classificatori deterministici approssimano il classificatore stocastico è mostrato in tabella 4. Dove Hashing e VarBin raggiungono prestazioni simili nella maggior parte dei datasets.

Tabella 4: Approssimazione classificatore deterministico del classificatore stocastico

	Crime (5)		COMPAS (4)		Law School (5)	
	Train	Test	Train	Test	Train	Test
Average	0.001 (0.02)	0.001 (0.02)	0.068 (0.03)	0.069 (0.06)	0.265 (0.01)	0.262 (0.02)
Hashing	0.000 (0.01)	0.000 (0.03)	0.002 (0.03)	0.004 (0.06)	0.002 (0.01)	0.002 (0.01)
VarBin	0.000 (0.05)	0.001 (0.14)	0.001 (0.08)	0.007 (0.07)	0.002 (0.04)	0.002 (0.06)
	Adult (4)		Wiki Toxicity (5)		Business (8)	
	Train	Test	Train	Test	Train	Test
Average	0.003 (0.01)	0.003 (0.01)	0.023 (0.09)	0.023 (0.09)	0.091 (0.07)	0.090 (0.08)
Hashing	0.000 (0.01)	0.000 (0.01)	0.000 (0.01)	0.001 (0.01)	0.010 (0.03)	0.013 (0.07)
VarBin	0.000 (0.04)	0.000 (0.04)	0.002 (0.13)	0.003 (0.18)	0.001 (0.06)	0.005 (0.08)

4.3.2 Studio grafico

La Figura 3 fornisce una visualizzazione delle distribuzioni di output corrispondenti.



In cui sono rappresentati set di istogrammi (risultati del modello) per le donne candidate (rosso) e il complesso popolazione (verde) utilizzando il dataset Adult. Possiamo notare le ottime performance del classificatore deterministico hashing che fitta bene il classificatore stocastico. Del resto come ci aspettavamo avevamo già notato in precedenza con i valori calcolati nelle tabelle che le prestazioni dei classificatori deterministici hashing e VarBin erano molto simili a quelli da dover approssimare. Di contro il classificatore deterministico non vincolato è ben lontano da riuscire ad approssimarlo. La linea di base media fallisce su 4 datasets.

4.4 Ottimizzazione della metrica G-media

In questo paragrafo presentiamo un terzo esperimento su un problema multiclasse senza vincoli in cui cerchiamo di ottimizzare la metrica di valutazione della G-media, che è la media geometrica (per ogni classe) dell'accuratezza. Applichiamo un approccio formativo basato sul metodo Frank-Wolfe [9] sull'ABI Abalone dataset e presentano il risultato della de-randomizzazione di un insieme stocastico con 100 classificatori di base.
Questo problema che consideriamo è un problema di apprendimento vincolato multiclasse da Narasimhann et al. (vedi paper [12] studi correlati) dove l'obiettivo è ottimizzare la metrica di valutazione della G-media, data dalla media geometrica dell'accuratezza di un classificatore su diverse classi: $1 - \left(\prod_{j=1}^{m} acc_j(f)\right)^{1/m}$ dove m è il numero di classi e acc_j è la precisione del classificatore sulla classe j. Per questa metrica valori più alti sono migliori, con la metrica a zero anche se l'accuratezza di una delle classi è zero. Narasimhan et al. fornisce un approccio formativo basato sull'algoritmo Frank-Wolfe (F-W) che allena un classificatore stocastico per ottimizzare questa metrica di valutazione. Per un problema multiclasse, un classificatore stocastico può essere definito come $f : \mathcal{X} \to \Delta^{m-1} \subset R^m$ che prende un'istanza x e predice la classe j con probabilità $f_j(x)$. L'approccio F-W allena uno stochastic ensemble supportando molti classificatori deterministici multiclasse.

Conduciamo esperimenti sul dataset abalone UCI (vedi paper [18] studi correlati) utilizzato nel loro articolo, che ha 4177 esempi, 10 classi e 12 funzioni. Qui l'approccio F-W genera un stochastic ensemble di 100 classificatori. La tabella 5 presenta un confronto tra diversi approcci di de-randomizzazione da sostituire questo stochastic ensemble con un classificatore deterministico multiclasse. Includiamo anche come base, un classificatore che ottimizza l'errore di classificazione standard. Questo classificatore ha una precisione pari a zero in almeno una delle classi, e quindi produce zero G-media. Così fa il miglior iterato deterministico fornito da F-W. L'approccio thresholding (che equivale a prevedere la classe che riceve la massima probabilità dal classificatore stocastico) fallisce sul set di test. Hashing e VarBin hanno prestazioni vicine al classificatore stocastico, con VarBin che è il più vicino.

Tabella 5: Confronto tra strategie di de-randomizzazione per ottimizzare la metrica G-media multiclasse

	OptError	Stochastic	Threshold	Hashing	VarBin	Best Iterate
Train	0.0	0.252	0.247	0.264	0.257	0.0
Test	0.0	0.257	0.0	0.293	0.267	0.0

4.5 Studi aggiuntivi esterni al paper

Per la compresione del paper (come indicato dagli autori nell'appendice) ho analizzato i seguenti esempi indicati. Il tema si discosta da quello trattato nel paper in questione ma bensì si utilizzano lo stesso dataset e gli stessi strumenti quindi capirne il funzionamento facilita la comprensione del codice del paper in questione. L'applicazione web utilizzata per eseguire il codice è jupyter [2]e la libreria utlizzata è TensorFlow [3].

Breve excursus sull'esecuzione con jupyter notebook: tramite il proompt di Conda creare un ambiente virtuale. Dopo aver attivato l'ambiente, scaricare la libreria tensorflow e poi jupyter. Spostarsi nella cartella con il codice e la libreria. Infine lanciare il comando "jupyter notebook" per eseguire gli esempi.

TFCO- TensorFlow Constrained Optimization: E' una libreria per l'ottimizzazione dei problemi legati alla disuguaglianza in TensorFlow 1.14 e versioni successive (incluso TensorFlow 2). Nel caso più generale, sia la funzione oggettiva che i vincoli sono rappresentati come *Tensor s*, offrendo agli utenti la massima flessibilità nello specificare i loro problemi di ottimizzazione. La costruzione di questi *Tensor s* può essere ingombrante, quindi vengono fornite anche funzioni di supporto per semplificare la costruzione di problemi di ottimizzazione vincolata basati su rate, ovvero porzioni dei dati di allenamento su cui si verifica un evento (ad es. Rate di errore, rate positivo reale, recall, ecc.).

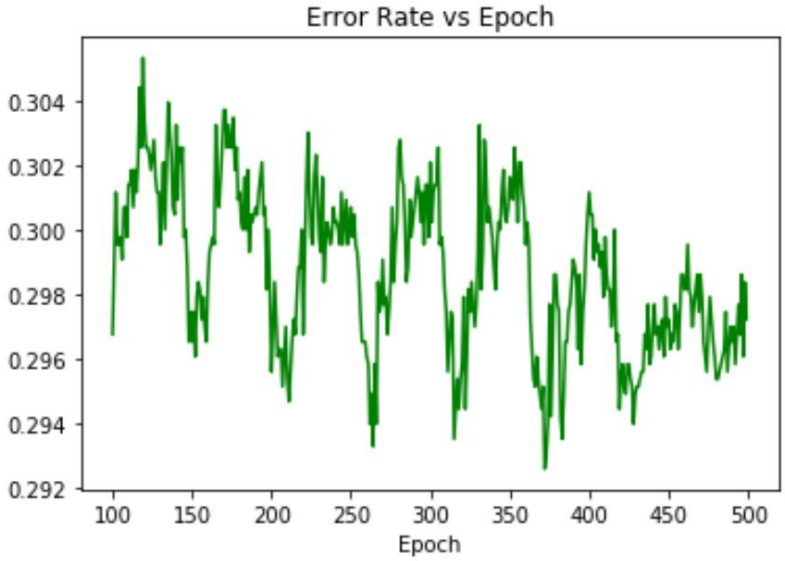
[2]Jupyter Notebook è un'applicazione web open source che consente di includere testo, video, audio e immagini e offre la possibilità di eseguire codice in diversi linguaggi di programmazione. La sua architettura è il principale vantaggio di questa applicazione. Questa ci consente di ospitare l'intera installazione di kernel, librerie e strumenti necessari in un server (può anche essere integrato con docker), impedendo così per esempio che ogni membro di un Data Science Team debba configurare tutto il proprio ambiente di lavoro localmente nel proprio computer.

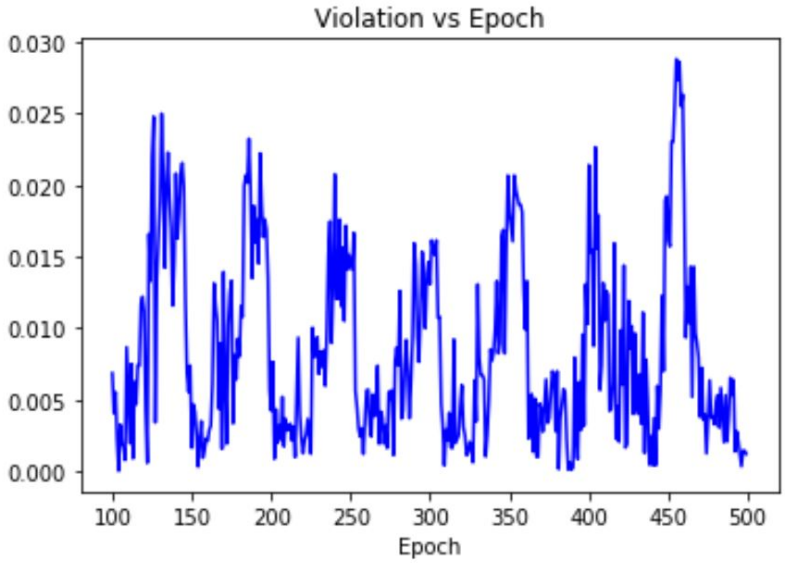
[3]TensorFlow è una Libreria software open source per l'apprendimento automatico, che fornisce moduli sperimentati e ottimizzati, utili nella realizzazione di algoritmi per diversi tipi di compiti percettivi e di comprensione del linguaggio.

4.5.1 Oscillation compas

Introduzione
Questo notebook illustra il problema di oscillazione sollevato nella sezione "shrinking": è possibile che le singole iterazioni non convergano quando si utilizza l'approccio lagrangiano all'allenamento con vincoli di equità, anche se in media convergono. Ciò motiva una selezione più attenta delle soluzioni o l'uso di un classificatore stocastico.

Questo notebook mostrerà una delle sfide che si presentano nella pratica durante l'addestramento di modelli di machine learning per soddisfare vincoli come i vincoli di equità: vale a dire la difficoltà che si presenta con i modelli non convessi. Approcci tipici per applicare tali vincoli si basano sul metodo classico dei moltiplicatori lagrangiani. È noto che nell'ambiente convesso, in condizioni abbastanza generali, un punto di sella al Lagrangiano corrisponderà a una soluzione ottimale e fattibile al problema dell'ottimizzazione vincolata originale. Tuttavia, tale non è più valido nell'impostazione non convessa: un punto di sella non corrisponde più necessariamente a una soluzione desiderata al problema di ottimizzazione vincolata. Ancora peggio, potrebbe non esserci nemmeno un punto stazionario a cui convergere nell'impostazione non convessa, facendo sì che metodi basati sul gradiente come SGD non possano eventualmente convergere e quindi oscillare. Mostriamo che questo problema si presenta con il dataset ProPublica COMPAS quando formiamo una semplice rete neurale come modello di base per prevedere la recettività a 2 anni con vincoli di pari opportunità basati su razza e genere. Questo giustifica il motivo per cui sono necessari classificatori stocastici. In particolare, ci sono quattro vincoli di pari opportunità: uno per ciascuno dei 4 gruppi protetti (bianco, nero, femmina, maschio). Come promemoria, le pari opportunità impongono che il vero tasso positivo del nostro classificatore su ciascun gruppo protetto corrisponda al vero tasso positivo del nostro classificatore dataset complessivo. Utilizziamo una rete neurale a 1 strato nascosto con attivazioni ReLU e 10 unità nascoste. Mostriamo che anche con una piccola rete neurale, vedremo già alcune oscillazioni e non convergenze che si verificano a causa della non convessità.





Ora tracciamo gli errori e le violazioni del nostro iterato alla fine di ogni periodo. Vediamo che non convergono e c'è un comportamento oscillante. Quindi suggerisce che il Lagrangiano o non ha alcun punto fermo a cui convergere o è molto difficile per i nostri metodi basati sul gradiente trovarne uno. Questo grafico ci mostra che gli iterati oscillano. Quindi dovremo usare un classificatore stocastico.

Utilizzo di un classificatore stocastico

Garanzie teoriche dell'iterato medio

```
In [11]: print("Results for Average Iterate.")
         print("Train Error", results[0]["t_stochastic_error_rate"])
         print("Train Violation", results[0]["t_stochastic_max_constraint_violation"])

         Results for Average Iterate.
         Train Error 0.3005370370370371
         Train Violation 0.0021946623392345966
```

Riduzione della soluzione stocastica

La media di tutti gli iterati può essere intensiva dal punto di vista computazionale (ad esempio, richiede l'archiviazione e l'elaborazione degli iterati in ogni epoca e il numero di iterati può diventare arbitrariamente elevato). Inoltre, l'iterato medio può avere componenti non ottimali (specialmente gli iterati iniziali). Nel paper "Two player games for efficient non-convex constrained optimiziation" [6] si propone un metodo per ridurre la miscela uniforme di tutte le iterazioni a una che è una miscela di al massimo $m+1$ soluzioni in cui m è il numero di vincoli con garanzie che la nuova soluzione sarà almeno altrettanto buona dell'originale soluzione. Questo può essere fatto usando la funzione $find\_best\_candidate\_distribution$. Vediamo un miglioramento sia in termini di correttezza che accuratezza

```
In [12]: print("Results after Shrinking")
         print("Train Error", results[0]["m_stochastic_error_rate"])
         print("Train Violation", results[0]["m_stochastic_max_constraint_violation"])

         Results after Shrinking
         Train Error 0.29370749834992005
         Train Violation 5.854691731421724e-18
```

4.5.2 Codice

[Importare le librerie richieste]

```
import math
import random
import numpy as np
import pandas as pd
import warnings
from six.moves import xrange
import tensorflow.compat.v1 as tf
import tensorflow_constrained_optimization as tfco
import matplotlib.pyplot as plt

tf.disable_eager_execution()

warnings.filterwarnings('ignore')
%matplotlib inline
```

[Leggere e processare il dataset]

```
LABEL_COLUMN = 'two_year_recid'
PROTECTED_COLUMNS = ['sex_Female', 'sex_Male', 'race_Caucasian',
'race_African-American']

def get_data():
    data_path = 'https://raw.githubusercontent.com/propublica/compas-analysis/
    master/compas-scores-two-years.csv'
    df = pd.read_csv(data_path)
    FEATURES = [
      'age', 'c_charge_degree', 'race', 'age_cat', 'score_text', 'sex',
      'priors_count', 'days_b_screening_arrest', 'decile_score', 'is_recid',
      'two_year_recid']
    df = df[FEATURES]
    df = df[df.days_b_screening_arrest <= 30]
    df = df[df.days_b_screening_arrest >= -30]
    df = df[df.is_recid != -1]
    df = df[df.c_charge_degree != 'O']
    df = df[df.score_text != 'N/A']
    continuous_features = [
      'priors_count', 'days_b_screening_arrest', 'is_recid', 'two_year_recid'
    ]
    continuous_to_categorical_features = ['age', 'decile_score', 'priors_count']
    categorical_features = ['c_charge_degree', 'race', 'score_text', 'sex']
```

```
def binarize_categorical_columns(input_df, categorical_columns=[]):
    binarized_df = pd.get_dummies(input_df, columns=categorical_columns)
    return binarized_df

def bucketize_continuous_column(input_df, continuous_column_name, bins=None):
    input_df[continuous_column_name] = pd.cut(
        input_df[continuous_column_name], bins, labels=False)

for c in continuous_to_categorical_features:
    b = [0] + list(np.percentile(df[c], [20, 40, 60, 80, 90, 100]))
    if c == 'priors_count':
        b = list(np.percentile(df[c], [0, 50, 70, 80, 90, 100]))
    bucketize_continuous_column(df, c, bins=b)

df = binarize_categorical_columns(
    df,
    categorical_columns=categorical_features
    + continuous_to_categorical_features)

to_fill = [
  u'decile_score_0', u'decile_score_1', u'decile_score_2',
  u'decile_score_3', u'decile_score_4', u'decile_score_5'
]
for i in range(len(to_fill) - 1):
    df[to_fill[i]] = df[to_fill[i:]].max(axis=1)
to_fill = [
  u'priors_count_0.0', u'priors_count_1.0', u'priors_count_2.0',
  u'priors_count_3.0', u'priors_count_4.0'
]
for i in range(len(to_fill) - 1):
    df[to_fill[i]] = df[to_fill[i:]].max(axis=1)

features = [
  u'days_b_screening_arrest', u'c_charge_degree_F', u'c_charge_degree_M',
  u'race_African-American', u'race_Asian', u'race_Caucasian',
  u'race_Hispanic', u'race_Native American', u'race_Other',
  u'score_text_High', u'score_text_Low', u'score_text_Medium',
  u'sex_Female', u'sex_Male', u'age_0', u'age_1', u'age_2', u'age_3',
  u'age_4', u'age_5', u'decile_score_0', u'decile_score_1',
  u'decile_score_2', u'decile_score_3', u'decile_score_4',
  u'decile_score_5', u'priors_count_0.0', u'priors_count_1.0',
  u'priors_count_2.0', u'priors_count_3.0', u'priors_count_4.0'
```

```
    ]
    df = df[features + [LABEL_COLUMN]]
    return df, features
```

[*Carichiamo ora i dati ed effettuiamo una suddivisione casuale dell'allenamento/test*]

```
df, FEATURE_NAMES = get_data()

tf.set_random_seed(1234)

df.sample(frac=1, random_state=12345)
N = len(df)
train_df = df[:int(N * 0.7)]
test_df = df[int(N * 0.7):]
```

[*Modello*]

```
def _construct_model(input_tensor, hidden_units=10):
    hidden = tf.layers.dense(
        inputs=input_tensor,
        units=hidden_units,
        activation=tf.nn.relu)
    output = tf.layers.dense(
        inputs=hidden,
        units=1,
        activation=None)
    return output

class Model(object):
    def __init__(self,
                 feature_names,
                 hidden_units=10,
                 tpr_max_diff=0):
        tf.random.set_random_seed(123)
        self.feature_names = feature_names
        self.tpr_max_diff = tpr_max_diff
        num_features = len(self.feature_names)
        self.features_placeholder = tf.placeholder(
            tf.float32, shape=(None, num_features), name='features_placeholder')
        self.labels_placeholder = tf.placeholder(
            tf.float32, shape=(None, 1), name='labels_placeholder')
        self.protected_placeholders = [tf.placeholder(tf.float32, shape=(None, 1),
        name=attribute+"_placeholder") for attribute in PROTECTED_COLUMNS]
        self.predictions_tensor = _construct_model(
```

```
            self.features_placeholder, hidden_units=hidden_units)

    def build_train_op(self,
                       learning_rate,
                       unconstrained=False):
        ctx = tfco.rate_context(self.predictions_tensor, self.labels_placeholder)
        positive_slice = ctx.subset(self.labels_placeholder > 0)
        overall_tpr = tfco.positive_prediction_rate(positive_slice)
        constraints = []
        if not unconstrained:
            for placeholder in self.protected_placeholders:
                slice_tpr = tfco.positive_prediction_rate(ctx.subset(
                (placeholder > 0) & (self.labels_placeholder > 0)))
                constraints.append(slice_tpr <= overall_tpr + self.tpr_max_diff)
        mp = tfco.RateMinimizationProblem(tfco.error_rate(ctx), constraints)
        opt = tfco.ProxyLagrangianOptimizerV1(tf.train.AdamOptimizer(learning_rate))
        self.train_op = opt.minimize(mp)
        return self.train_op

    def feed_dict_helper(self, dataframe):
        feed_dict = {self.features_placeholder:
                  dataframe[self.feature_names],
              self.labels_placeholder:
                  dataframe[[LABEL_COLUMN]],}
        for i, protected_attribute in enumerate(PROTECTED_COLUMNS):
            feed_dict[self.protected_placeholders[i]] =
            dataframe[[protected_attribute]]
        return feed_dict
```

[*Addestramento*]

```
def training_generator(model,
                       train_df,
                       test_df,
                       minibatch_size,
                       num_iterations_per_loop=1,
                       num_loops=1):
    random.seed(31337)
    num_rows = train_df.shape[0]
    minibatch_size = min(minibatch_size, num_rows)
    permutation = list(range(train_df.shape[0]))
    random.shuffle(permutation)

    session = tf.Session()
```

```
    session.run((tf.global_variables_initializer(),
                 tf.local_variables_initializer()))

    minibatch_start_index = 0
    for n in xrange(num_loops):
        for _ in xrange(num_iterations_per_loop):
            minibatch_indices = []
            while len(minibatch_indices) < minibatch_size:
                minibatch_end_index = (
                minibatch_start_index + minibatch_size - len(minibatch_indices))
                if minibatch_end_index >= num_rows:
                    minibatch_indices += range(minibatch_start_index, num_rows)
                    minibatch_start_index = 0
                else:
                    minibatch_indices += range(minibatch_start_index,
                    minibatch_end_index)
                    minibatch_start_index = minibatch_end_index
            session.run(
                  model.train_op,
                  feed_dict=model.feed_dict_helper(
                      train_df.iloc[[permutation[ii] for ii in minibatch_indices]]))

        train_predictions = session.run(
            model.predictions_tensor,
            feed_dict=model.feed_dict_helper(train_df))
        test_predictions = session.run(
            model.predictions_tensor,
            feed_dict=model.feed_dict_helper(test_df))

        yield (train_predictions, test_predictions)
```

[Calcolo delle metriche di accuracy e fairness]

```
def error_rate(predictions, labels):
    signed_labels = (
      (labels > 0).astype(np.float32) - (labels <= 0).astype(np.float32))
    numerator = (np.multiply(signed_labels, predictions) <= 0).sum()
    denominator = predictions.shape[0]
    return float(numerator) / float(denominator)


def positive_prediction_rate(predictions, subset):
    numerator = np.multiply((predictions > 0).astype(np.float32),
```

```
                          (subset > 0).astype(np.float32)).sum()
    denominator = (subset > 0).sum()
    return float(numerator) / float(denominator)

def tpr(df):
    """Measure the true positive rate."""
    fp = sum((df['predictions'] >= 0.0) & (df[LABEL_COLUMN] > 0.5))
    ln = sum(df[LABEL_COLUMN] > 0.5)
    return float(fp) / float(ln)

def _get_error_rate_and_constraints(df, tpr_max_diff):
    """Computes the error and fairness violations."""
    error_rate_local = error_rate(df[['predictions']], df[[LABEL_COLUMN]])
    overall_tpr = tpr(df)
    return error_rate_local, overall_tpr, [tpr(df[df[protected_attribute] > 0.5]) -
    (overall_tpr + tpr_max_diff) for protected_attribute in PROTECTED_COLUMNS]

def _get_exp_error_rate_constraints(cand_dist, error_rates_vector,
                                    overall_tpr_vector, constraints_matrix):
    ""Computes the expected error and fairness violations on a randomized solution.""
    expected_error_rate = np.dot(cand_dist, error_rates_vector)
    expected_overall_tpr = np.dot(cand_dist, overall_tpr_vector)
    expected_constraints = np.matmul(cand_dist, constraints_matrix)
    return expected_error_rate, expected_overall_tpr, expected_constraints


def get_iterate_metrics(cand_dist, best_cand_index, error_rate_vector,
                        overall_tpr_vector, constraints_matrix):
    metrics = {}
    exp_error_rate, exp_overall_tpr, exp_constraints =
    _get_exp_error_rate_constraints(
      cand_dist, error_rate_vector, overall_tpr_vector, constraints_matrix)
    metrics['m_stochastic_error_rate'] = exp_error_rate
    metrics['m_stochastic_overall_tpr'] = exp_overall_tpr
    metrics['m_stochastic_max_constraint_violation'] = max(exp_constraints)
    for i, constraint in enumerate(exp_constraints):
        metrics['m_stochastic_constraint_violation_%d' % i] = constraint
    metrics['best_error_rate'] = error_rate_vector[best_cand_index]
    metrics['last_error_rate'] = error_rate_vector[-1]
    metrics['t_stochastic_error_rate'] = sum(error_rate_vector) / len(
      error_rate_vector)
    metrics['best_overall_tpr'] = overall_tpr_vector[best_cand_index]
    metrics['last_overall_tpr'] = overall_tpr_vector[-1]
```

```
    metrics['t_stochastic_overall_tpr'] = sum(overall_tpr_vector) / len(
      overall_tpr_vector)
    avg_constraints = []
    best_constraints = []
    last_constraints = []
    for constraint_iterates in np.transpose(constraints_matrix):
        avg_constraint = sum(constraint_iterates) / len(constraint_iterates)
        avg_constraints.append(avg_constraint)
        best_constraints.append(constraint_iterates[best_cand_index])
        last_constraints.append(constraint_iterates[-1])
    metrics['best_max_constraint_violation'] = max(best_constraints)
    for i, constraint in enumerate(best_constraints):
        metrics['best_constraint_violation_%d' % i] = constraint
        metrics['last_max_constraint_violation'] = max(last_constraints)
    for i, constraint in enumerate(last_constraints):
        metrics['last_constraint_violation_%d' % i] = constraint
        metrics['t_stochastic_max_constraint_violation'] = max(avg_constraints)
    for i, constraint in enumerate(avg_constraints):
        metrics['t_stochastic_constraint_violation_%d' % i] = constraint
    metrics['all_errors'] = error_rate_vector
    metrics['all_violations'] = np.max(constraints_matrix, axis=1)

    return metrics

def training_helper(model,
                    train_df,
                    test_df,
                    minibatch_size,
                    num_iterations_per_loop=1,
                    num_loops=1):
    train_objective_vector = []
    train_constraints_loss_matrix = []
    train_error_rate_vector = []
    train_overall_tpr_vector = []
    train_constraints_matrix = []
    test_error_rate_vector = []
    test_overall_tpr_vector = []
    test_constraints_matrix = []
    for train, test in training_generator(
      model, train_df, test_df, minibatch_size, num_iterations_per_loop,
      num_loops):
        train_df['predictions'] = train
        test_df['predictions'] = test
```

```
      train_error_rate, train_overall_tpr, train_constraints =
      _get_error_rate_and_constraints(
        train_df, model.tpr_max_diff)
      train_error_rate_vector.append(train_error_rate)
      train_overall_tpr_vector.append(train_overall_tpr)
      train_constraints_matrix.append(train_constraints)

      test_error_rate, test_overall_tpr, test_constraints =
      _get_error_rate_and_constraints(
          test_df, model.tpr_max_diff)
      test_error_rate_vector.append(test_error_rate)
      test_overall_tpr_vector.append(test_overall_tpr)
      test_constraints_matrix.append(test_constraints)

  cand_dist = tfco.find_best_candidate_distribution(
    train_error_rate_vector, train_constraints_matrix, epsilon=0.001)
  best_cand_index = tfco.find_best_candidate_index(
    train_error_rate_vector, train_constraints_matrix)
  train_metrics = get_iterate_metrics(
    cand_dist, best_cand_index, train_error_rate_vector,
    train_overall_tpr_vector, train_constraints_matrix)
  test_metrics = get_iterate_metrics(
    cand_dist, best_cand_index, test_error_rate_vector,
    test_overall_tpr_vector, test_constraints_matrix)

  return (train_metrics, test_metrics)
```

4.5.3 Fairness CelebA

Introduzione
Questo notebook mostra come addestrare un classificatore giusto ("fair") per prevedere di rilevare il sorriso di una celebrità nelle immagini usando tf.keras e il dataset su larga scala di attributi CelebFaces. Il modello addestrato in questo notebook valuta l'equità in tutte le fasce d'età, con il tasso di falsi positivi impostato come vincolo.

Contenuto:
Basandosi su altri esempi di TensorFlow Constrained Optimization (TFCO), lo scopo di questo notebook è di dimostrare ulteriormente quanto sia facile creare e ottimizzare i problemi vincolati utilizzando la libreria TFCO. In particolare, questo notebook:

- Addestra un modello di rete neurale semplice e senza vincoli per rilevare il sorriso di una persona nelle immagini usando tf.keras e il set di dati CelebFaces Attributes (CelebA) su larga scala.
- Valuta le prestazioni del modello in tutte le fasce d'età
- Imposta un semplice problema di ottimizzazione vincolata per ottenere prestazioni più eque in tutte le fasce d'età.
- Rielabora il modello ormai vincolato e valuta nuovamente le prestazioni.

CelebA Dataset:
CelebA è un dataset di attributi di volti su larga scala con oltre 200.000 immagini di celebrità, ognuna con 40 annotazioni di attributi (come tipo di capelli, accessori di moda, lineamenti del viso, ecc.) E 5 punti di riferimento (posizioni di occhi, bocca e naso). CelebA è facilmente disponibile in TensorFlow grazie ai dataset TensorFlow (tfds), che è dove e come questo notebook scaricherà ed estrarrà il dataset. L'attributo "Sorridente" - che presumibilmente rappresenta un'espressione compiaciuta, gentile o divertita nell'immagine con gli angoli della bocca rivolti verso l'alto e i denti anteriori eventualmente esposti - verrà impostato come variabile target per il modello da prevedere. Le immagini verranno ridimensionate da 218x178 a 28x28 per ridurre il tempo di esecuzione e la memoria durante l'allenamento. Le prestazioni saranno valutate attraverso l'attributo "Young", che sarà soprannominato "fascia d'età" in questo notebook.

Costruire, allenare e valutare il modello non vincolato

```
celeb_a_train_data = input_function(
    tfds.Split.TRAIN, batch_size=32, subgroup_filter='None')
model.fit(celeb_a_train_data, epochs=5, steps_per_epoch=1000)
```

```
Epoch 1/5
1000/1000 [==============================] - 41s 41ms/step - loss: 0.2385 - acc
uracy: 0.8572 - fp: 1097.0000 - tn: 15468.0000
Epoch 2/5
1000/1000 [==============================] - 40s 40ms/step - loss: 0.2384 - acc
uracy: 0.8571 - fp: 1065.0000 - tn: 15685.0000
Epoch 3/5
1000/1000 [==============================] - 40s 40ms/step - loss: 0.2370 - acc
uracy: 0.8576 - fp: 1001.0000 - tn: 15588.0000
Epoch 4/5
1000/1000 [==============================] - 40s 40ms/step - loss: 0.2354 - acc
uracy: 0.8560 - fp: 1051.0000 - tn: 15685.0000
Epoch 5/5
1000/1000 [==============================] - 41s 41ms/step - loss: 0.2297 - acc
uracy: 0.8595 - fp: 1021.0000 - tn: 15579.0000
```

Valutando il modello sui dati di test risulta che l'accuracy ha un valore che varia tra 80% e 85%.

```
print('Overall Results, Unconstrained')
celeb_a_test_data = input_function(
    tfds.Split.TEST, batch_size=32, subgroup_filter='None')
results = model.evaluate(celeb_a_test_data)
```

```
Overall Results, Unconstrained
    624/Unknown - 25s 40ms/step - loss: 0.2217 - accuracy: 0.8760 - fp: 965.000
0 - tn: 9010.0000
```

Tuttavia, le prestazioni valutate in tutte le fasce di età possono rivelare alcune carenze.

```
print('Results for Young Group, Unconstrained')
celeb_a_test_young_group_data = input_function(
    tfds.Split.TEST, batch_size=1, subgroup_filter='Young')
unconstrained_results_young = model.evaluate(celeb_a_test_young_group_data)
```

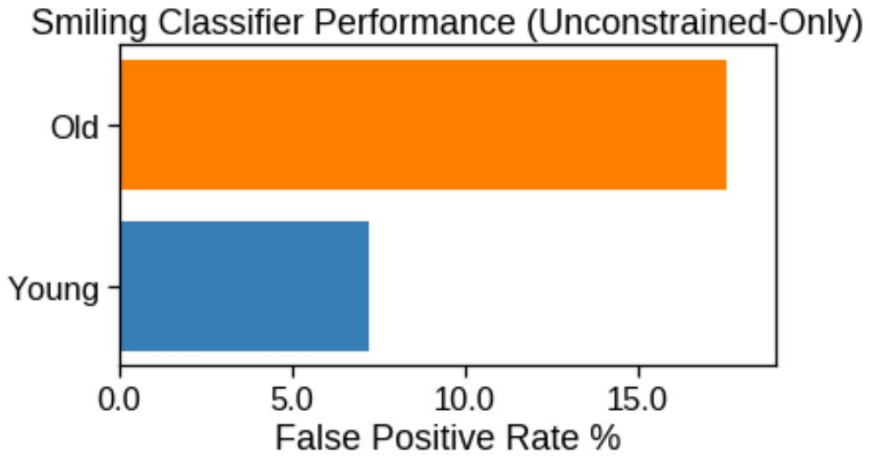
```
Results for Young Group, Unconstrained
  15114/Unknown - 104s 7ms/step - loss: 0.2929 - accuracy: 0.8857 - fp: 551.000
0 - tn: 7066.0000
```

```
print('Results for Old Group, Unconstrained')
celeb_a_test_old_group_data = input_function(
    tfds.Split.TEST, batch_size=1, subgroup_filter='Old')
unconstrained_results_old = model.evaluate(celeb_a_test_old_group_data)
```

```
Results for Old Group, Unconstrained
   4848/Unknown - 52s 11ms/step - loss: 0.0000e+00 - accuracy: 0.8457 - fp: 41
4.0000 - tn: 1944.0000
```

```
#@title Calculate FPR and Compare Unconstrained Performance
young_fpr_unconstrained = calculate_fpr(unconstrained_results_young)
old_fpr_unconstrained = calculate_fpr(unconstrained_results_old)

title = 'Smiling Classifier Performance (Unconstrained-Only)'
plot_results([young_fpr_unconstrained, old_fpr_unconstrained], title)
```



Un falso positivo è quando il modello prevede erroneamente la classe positiva. In questo contesto, ciò significa che un esito falso positivo si verifica quando l'immagine rappresenta una celebrità "Non sorridente" e il modello prevede "Sorridere". Per estensione, il tasso di falsi positivi, che viene utilizzato nell'immagine sopra, è una misura di accuratezza per un test. Dall'immagine si può notare una grande differenza tra le categorie giovani e anziani ovvero all'interno di fasce d'età differenti. È qui che il TFCO può aiutare a colmare tale divario vincolando il rate di falsi positivi per diventare un criterio più accettabile.

Modello vincolato

Come documentato dalla libreria TFCO ci sono diverse funzioni che aiutano a vincolare il problema, tra le quali utilizzeremo:

1. $tfco.rate\_context()$
2. $tfco.RateMinimizationProblem()$
3. $tfco.ProxyLangragianOptimizerV2()$

Esse permetteranno di porre dei vincoli di equità sul modello.

```
print('Overall Results, Constrained')
celeb_a_test_data = input_function(
    tfds.Split.TEST, batch_size=32, subgroup_filter='None')
results = model.evaluate(celeb_a_test_data)
```

```
Overall Results, Constrained
    624/Unknown - 25s 39ms/step - loss: 0.2014 - accuracy: 0.8648 - fp: 400.0000 - tn: 9575.0000
```

Non è sempre come in questo caso, ma qui, le prestazioni complessive del modello vincolato sono migliori del modello non vincolato. Ma ciò che è più importante qui è migliorare le prestazioni tra i gruppi.

```
print('Results for Young Group, Constrained')
celeb_a_test_young_group_data = input_function(
    tfds.Split.TEST, batch_size=1, subgroup_filter='Young')
constrained_results_young = model.evaluate(celeb_a_test_young_group_data)
```

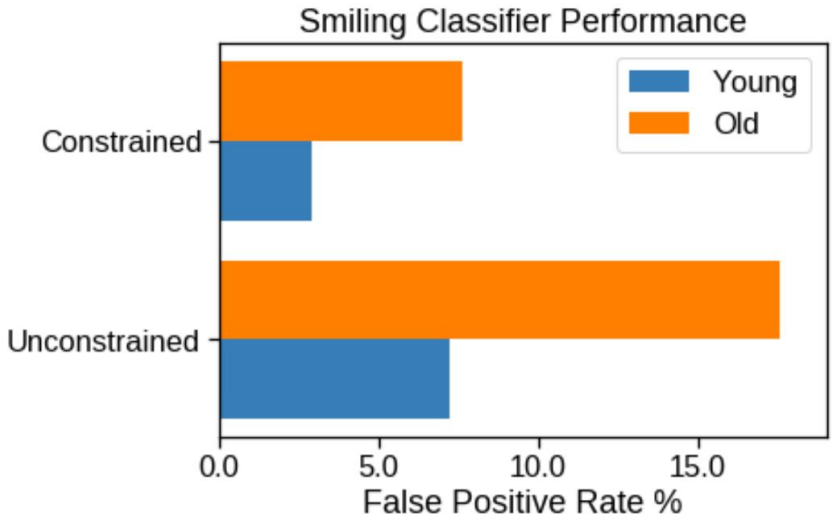
```
Results for Young Group, Constrained
  15114/Unknown - 105s 7ms/step - loss: 0.2660 - accuracy: 0.8718 - fp: 220.0000 - tn: 7397.0000
```

```
print('Results for Old Group, Constrained')
celeb_a_test_old_group_data = input_function(
    tfds.Split.TEST, batch_size=1, subgroup_filter='Old')
constrained_results_old = model.evaluate(celeb_a_test_old_group_data)
```

```
Results for Old Group, Constrained
   4848/Unknown - 52s 11ms/step - loss: 0.0000e+00 - accuracy: 0.8432 - fp: 180.0000 - tn: 2178.0000
```

```
#@title Calculate FPR and Compare Constrained Performance
young_fpr_constrained = calculate_fpr(constrained_results_young)
old_fpr_constrained = calculate_fpr(constrained_results_old)

title = 'Smiling Classifier Performance'
fpr_results = [young_fpr_unconstrained, old_fpr_unconstrained,
               young_fpr_constrained, old_fpr_constrained]
plot_results(fpr_results, title)
```



Con la capacità di TFCO di esprimere un requisito più complesso come vincolo di rate, ha aiutato questo modello a raggiungere un risultato più desiderabile. Certo, c'è ancora spazio per migliorare, ma almeno TFCO è stato in grado di trovare il modello con le migliori prestazioni che soddisfa il vincolo (almeno, rispetto al gruppo "Young"; il gruppo "Old" è vicino a soddisfare il vincolo) riducendo il più possibile la disparità tra i gruppi.

Capitolo 5

Conclusioni

La questione che ci siamo posti all'inizio di questo paper era come e quanto bene un classificatore deterministico può approssimare un classificatore stocastico. Il motivo della problematica nasceva dal fatto che nonostante la larga diffusione e utilità dei classificatori stocastici la loro natura causale li rende problematici per svariati motivi. Abbiamo constatato e analizzato diversi approcci per approssimare un classificatore stocastico con un classificatore deterministico: approccio hash, approccio threshold e approccio VarBin. Abbiamo definito una metrica che ci consentisse di valutare la bontà dell'approssimazione. Abbiamo indagato sperimentalmente i pro e i contro di questi metodi, non solo riguardo al successo di ciascun classificatore deterministico nel riuscire ad approssimare il classificatore stocastico originale, ma anche in termini di quanto bene ciascun classificatore deterministico risolve gli altri problemi per riuscire a valutare quanto siano indesiderabili i classificatori stocastici. Questi tre approcci li abbiamo analizzati sia con le ROC curve matching, in cui l'hashing e il VarBin si avvicinavano alle performace del classificatore stocastico diversamente dal thresholding, sia con i matching regression histograms in cui abbiamo notato le ottime performance del classificatore deterministico hash. Poi ci siamo focalizzati su uno di essi che è il metodo hash che gode di una garanzia teorica che si confronta favorevolmente con il limite inferiore che abbiamo calcolato, in termini di quanto l'approssimazione conserva le metriche dei rate aggregati. Sebbene, le ragioni per le quali un classificatore deterministico sia preferibile a un classificatore stocastico siano diverse tra le quali: la stabilità, la possibilità di debuggare, l'equità in varie sfumature e resistenza in quanto non può essere manipolato sfruttando l'uso ripetuto. Vista in questi termini, un classificatore disordinato, come quello che risulta dal metodo hash potrebbe essere insoddisfacente. Applicare pre-clustering all'approccio hash parzialmente risolve questo problema (come abbiamo visto con l'approccio variable binning) ma lascia delle questioni importanti aperte tra le quali come bisognerebbe misurare la somiglianza e se possiamo migliorare la proprietà del ""local orderliness" (ordinamento locale) proprietà di cui gli approcci godono e se ci sono casi speciali in cui si può costruire un accurato classificatore deterministico senza perdere l'orderliness. Un altro possibile affinamento sarebbe considerare metriche più generali rispetto ai rate aggregati che consideriamo.

Ad esempio, si potrebbero potenzialmente utilizzare funzioni piatte regolari per gestire le metriche F-score o G-mean [21]. Oppure, per supportare la classifica o impostazioni di regressione, si potrebbero definire rate metrics su coppie di esempi [19], [20].

Capitolo 6

Considerazioni personali

Argomento trattato:
L'argomento trattato è decisamente una questione centrale nel supervised learning e della classificazione. Il fatto che molti classificatori siano stocastici con la relativa natura casuale li rende problematici nell'utilizzo nel reale. Riuscire ad approssimare un classificatore stocastico con un classificatore deterministico sarebbe davvero comodo; tuttavia a parer mio è molto difficile se non impossibile. Il motivo risiede nel comportamento stocastico e nella sua natura intrinseca casuale che è "inimitabile"; un classificatore deterministico può riuscire ad approssimare bene in alcune iterazioni un classificatore stocastico ma valutato su diverse iterazioni molto probabilmente sarà lontano dall'avere lo stesso comportamento.

Approccio del paper:
L'approccio del paper lo trovo giusto. In prima istanza si è occupato di definire una metrica per valutare la bontà dell'approssimazione. Poi si sono definiti dei bound all'approssimazione. Tutto ciò ci ha permesso di definire i "confini" e dei parametri oggettivi. Poi ha scelto tre approcci semplici (hashing, thresholding e VarBin) ma concreti di approssimazione ed ha valutato le performance. Non avrei fatto diversamente nel procedere e ora porterei avanti lo studio analizzando più precisamente le performance e utilizzando ulteriori classificatori deterministici più complessi.

Avvenire:
Il lavoro è inedito nel senso che non viene portato avanti un lavoro e/o problema precedentemente elaborato bensì si lavora su un argomento nuovo in seguito al quale potranno esserci ulteriori sviluppi in futuro. Inoltre essendo un lavoro recente del 2019 non sono ancora presenti lavori pubblicati che citino o studi correlati per questo paper. Visto l'utilità e quanto sia interessante la questione sicuramente ci saranno ulteriori sviluppi. Ad esempio si potrebbero usare metriche più generali rispetto ai rate aggregati utilizzati tutt'ora oppure andare a studiare come misurare la somiglianza oppure indagare come migliorare la proprietà del "local orderliness".

Capitolo 7

Riferimenti bibliografici

In questa sezione verrà inserita tutta la bibliografia utilizzata per svolgere l'elaborato. Vi saranno sia siti internet che verranno indicati con link ipertestuali sottolineati in blu sia paper scientifici che verranno scritti in blu e non sottolineati.

[1] https://www.intelligenzaartificiale.it/machine-learning/ Per definire il machine learning e per definire il supervised learning

[2] https://www.simonefavaro.it/2017/04/07/introduzione-machine-learning/ Per trattare del supervised learning e la classificazione

[3] http://www.andreaminini.com/ai/machine-learning/classificazione-supervisionata-nel-machine-learning Per la classificazione

[4] Paper Satisfying real-world goals with dataset constraints. Gabriel Goh, Andrew Cotter, Maya Gupta, and Michael P Friedlander. In NIPS, pages 2415–2423. 2016.

[5] Paper Optimization with non-differentiable constraints with applications to fairness, recall, churn, and other goals. Andrew Cotter, Heinrich Jiang, and Karthik Sridharan.

[6] Paper Two-player games for efficient nonconvex constrained optimization. Andrew Cotter, Heinrich Jiang, Serena Wang, Taman Narayan, Maya Gupta, Seungil You, and Karthik Sridharan. In Algorithmic Learning Theory, pages 300–332, 2019.

[7] Paper Robust optimization for non-convex objectives. Robert S. Chen, Brendan Lucier, Yaron Singer, and Vasilis Syrgkanis. In NIPS, 2017.

[8] Paper Evaluating text categorization. Robert S. Chen, Brendan Lucier, Yaron Singer, and Vasilis Syrgkanis. In NIPS, 2017.

[9] Paper Optimizing non-decomposable performance measures: a tale of two classes. H. Narasimhan, P. Kar, and P. Jain. In ICML, 2015.

[10] Paper PAC-Bayes and margins John Langford and John Shawe-Taylor. In NIPS, 2002.

[11] Paper Pac-bayes bounds for the risk of the majority vote and the variance of the gibbs classifier. Alexandre Lacasse, François Laviolette, Mario Marchand, Pascal Germain, and Nicolas Usunier. In Advances in Neural information processing systems, pages 769–776, 2007.

[12] Paper Robust classification for imprecise environments Foster Provost and Tom Fawcett. Machine learning, 42(3):203–231, 2001.

[13] https://www.analyticsvidhya.com/blog/2019/08/detailed-guide-7-loss-functions-machine-learning-python-code/ Loss function

[14] https://www.gohacking.com/what-is-md5-hash/ MD5 hash di GHadmin.

[15] Paper Fairness through awareness C. Dwork, M. Hardt, T. Pitassi, O. Reingold, and R. Zemel. In Proc. 3rd Innovations in Theoretical Computer Science, pages 214–226. ACM, 2012.

[16] Paper A reductions approach to fair classification Alekh Agarwal, Alina Beygelzimer, Miroslav Dudík, John Langford, and Hanna M. Wallach. In ICML, pages 60–69, 2018.

[17] Paper Elements of Statistical Learning T. Hastie, R. Tibshirani, and J. Friedman. Springer, 2016.

[18] http://archive.ics.uci.edu/ml Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017

[19] Paper Fairness in recommendation through pairwise experiments A. Beutel, J. Chen, T. Doshi, H. Qian, L. Wei, Y. Wu, L. Heldt, Z. Zhao, L. Hong, E. H. Chi,and C. Goodrow. KDD Applied Data Science Track, 2019.

[20] https://arxiv.org/abs/1906.05330. Pairwise fairness for ranking and regression, 2019. M. Gupta S. Wang H. Narasimhan, A. Cotter.

[21] Paper Optimizing generalized rate metrics with three players M. Gupta H. Narasimhan, A. Cotter In NeurIPS, 2019.

[22] L. Wightman. LSAC national longitudinal bar passage study. Law School Admission Council, 1998.

[23] Preserving statistical validity in adaptive data analysis. Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Leon Roth. In Proceedings of the Forty- Seventh Annual ACM on Symposium on Theory of Computing, pages 117–126. ACM, 2015.

[24] Paper Measuring and mitigating unintended bias in text classification L. Dixon, J. Li, J. Sorensen, N. Thain, and L. Vasserman In AIES, 2018.

[25] Machine bias: There's software used across the country to predict future criminals, and it's biased against blacks, May 2016. Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner.

[26] Paper Preserving statistical validity in adaptive data analysis. In Proceedings of the Forty- Seventh Annual ACM on Symposium on Theory of Computing, pages 117–126. ACM, 2015.Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Leon Roth.

[27] https://research.cs.vt.edu/AVresearch/hashing/binning.php Hashing tutorial.

[28] Tesi https://www.tesionline.it/tesi/Machine-Learning-Techniques-Applied-to-Spam-/22433 Tratta di spamming e dei metodi di apprendimento automatico utilizzati per riconoscerli, analizzando infine i vantaggi economici per un'azienda nell'acquisto di un sistema anti-spam.

[29] Paper Learning with complex loss functions and constraints. In AIStats, 2018. Harikrishna Narasimhan.

[30] Paper The Genia event extraction shared task, 2013 editionoverview. ACL 2013, 2013. J-D. Kim, Y. Wang, and Y. Yasunori.

[31] Boosting for learning multiple classes with imbalanced class distribution. In ICDM, 2006. Y. Sun, M.S. Kamel, and Y. Wang.

[32] Multiclass imbalance problems: Analysis and potential solutions. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 42(4):1119–1130, 2012. S. Wang and X. Yao.

[33] Neural network classification and prior class probabilities. In Neural Networks: Tricks of the Trade, LNCS, pages 1524:299–313. 1998.S. Lawrence, I. Burns, A. Back,

A-C. Tsoi, and C.L. Giles.

[34] PAC-Bayesian stochastic model selection In Machine Learning, 2003. David McAllester.

[35] Stochastic feature mapping for PAC-Bayes classification. In Machine Learning, 2015. Xiong Li, Bin Wang, Yuncai Liu, and Tai Sing Lee.

[36] Equality of opportunity in supervised learning. In NIPS, 2016. Moritz Hardt, Eric Price, and Nathan Srebro.

[37] What makes a lottery fair? In Nous, pages 203–216, 1980. G. Sher.

[38] The equality of lotteries. In Philosophy, volume 83, pages 359–372, 2008. B. Saunders.

[39] https://arxiv.org/abs/1711.05144. Preventing fairness gerrymandering: Auditing and learning for subgroup fairness, 2017. Michael Kearns, Seth Neel, Aaron Roth, and Zhiwei Steven Wu.

[40] Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pages 1532–1543, 2014. Jeffrey Pennington, Richard Socher, and Christopher Manning.

[41]https://people.csail.mit.edu/ronitt/COURSE/S12/handouts/lec5.pdf. Notes for lecture 5 of MIT 6.842: Randomness and computation, February 2012. Ronitt Rubinfeld.