Fakultet organizacionih nauka, Beograd – Otkrivanje zakonitosti u podacima

**„Ansambl algoritmi“**

**Damir Pajaziti,**

11. April 2020.

# Uvod

U ovom radu, obrađene su različite ansambl tehnike. Neki od algoritama su detaljnije obrađeni, s obzirom da su se pokazali kao jedni od najboljih na poznatim *data science* takmičenjima. Jedan od takvih primera je i XGBoost algoritam. Nakon što je svaka tehnika objašnjena intuitivno a nakon toga i koracima algoritma pa i formulama, prikazana su i poređenja performansi na različitim skupovima podataka. Razlozi viših performansi ansambla u odnosnu na monolitne prediktore mogu se intuitivno objasniti *bernulijevim* procesom koji nije u okvirima naše teme pa ga nećemo detaljnije obraditi.

# Bagging

Prvi algoritam koji će biti obrađen je Bagging ansambl. On je jedan od prvenaca i koristi osnovne tehnike koje će se i u novijim ansamblima koristiti.

Bagging metoda je jedna od ansambl tehnika objavljena od strane Leo Brajmana [Breiman (1996)]. Ova tehnika kreira više verzija jednog prediktora odnosno klasifikatora, čiji rezultati se na kraju sakupljaju i kreira se glasanje za konačnu odluku ansambla. Svaki prediktor dobija svoj set podataka koji je generisan tehnikom uzorkovanja sa zamenom, odnosno statističkom metodom *bootstraping* [Horowitz (2001)]. Algoritam je dobio ime kombinacijom reči ***b****oostraping i* ***agg****regat****ing*** (agregacija konačnih glasova pojedinačnih prediktora ansambla). Ova metoda spada u grupu koja ima mogućnost paralelizacije, jer učenje prediktora nije zavisno jedno od drugog. Korišćenjem uzorkovanja sa zamenom, u proseku svaki bootstrap koristi oko 63% slučajeva iz originalnog seta podataka, odnosno oko 37% slučajeva ostaje ne iskorišćeno. Ideja ove metode je da se koristi slab osnovni prediktor, i da u ukupnom glasanju dostigne veću preciznost.

Koraci algoritma:

1. Kreirati uniformnu distribuciju nad slučajevima u setu:
2. Pravljenje N boostrap uzoraka nad skupom podataka iz koraka 1 (trening skup) veličine K.
3. Za svaki od N uzoraka kreirati jedan model i trenirati ga koristeći osnovni algoritam: . Svaki prediktor daje svoje predviđanje atributa .
4. Računa se očekivana vrednost. Za kategoričke izlaze to će biti kategorija sa najviše glasova, dok će za numeričke biti srednja vrednost rezultata prediktora.

# Random Forest

Ansambl *Random Forest* je formulisao Leo *Brajman*, 2001. godine. Profesor *Brajman* u svojoj publikaciji navodi još autora koji su obrađivali sličnu tematiku i čiji su radovi uticali na razvijanje ideje o *Random Forest* ansamblu (*Dieterich[1998], Ho [1998], Amit and Geman [1997]*).

Ovaj ansambl je kombinacija prediktora stabla odlučivanja, koji međusobno zavise samo od nasumično kreiranih vektora atributa sa istom distribucijom. Ideja glasanja radi dobijanja konačnog izlaza je preuzeta od ranije pomenutog ansambla *Bagging*. Sam algoritam je lako paralelizovati zbog nezavisne izgradnje stabala.

Podaci su definisani:

Parametri algoritma su:

1. Osnovni algoritam -
2. Broj osnovnih modela u ansamblu – N
3. Broj atributa koji se selektuju – F

Koraci algoritma su:

1. Kreiranje raspodele verovatnoće nad atributima skupa podataka. Uglavnom se koristi uniformna raspodela.
2. Pravljenje vektora atributa veličine F za N modela u ansamblu. Atributi se biraju nasumično.
3. Za svaki model u ansamblu i njegov odgovarajući set podataka kome je dodeljen odgovarajući vektor atributa kreira se stablo odlučivanja osnovnim algoritmom. Svaki model je u stanju da napravi predviđanje nezavisno od ostalih modela.
4. Za predviđanje celog ansambla koristi se ista tehnika kao u četvrtom koraku prethodno opisanog *Bagging ansambla.*

# Boosting – Adaboost

Adaboost ansambl algoritam (*Freund and Schapire [1995]*) je formulisan, kako bi rešio praktične probleme prethodnih boosting algoritama. Početna ideja boosting algoritama je slična kao i kod bagging ansambla. Od slabih prediktora potrebno je napraviti konačni, bolji, koji će biti izglasan u ansamblu. Još jedna ideja boosting-a je da se iterativno treniraju modeli i da se penalizuju instance u setu nad kojima se napravi greška u predviđanju modela t. Svaki naredni model uči sa većim fokusom na greškama prethodnih modela. Ovo se postiže tako što osnovni algoritam mora da ima implementiranu funkcionalnost otežavanja zapisa u setu podataka. Kao i u stvarnom životu, postoje različiti nivoi eksperata koji mogu sa različitim procentima uspešnosti rešiti zadatke. Ovaj algoritam ima sistem procene eksperata (modela), i njih takođe penalizuje prema uspešnosti predviđanja. Što znači da će bolji eksperti imati jači glas u konačnoj odluci prilikom glasanja u ansamblu. Ovaj algoritam se prilikom treniranja ansambla izvršava sekvenciono, jer kalkulacije svakog modela (težine) zavise od rezultata predviđanja prethodnog modela, dok je predviđanje dosta brže od treniranja i može da se paralelizuje. Upravo zbog svih prilagođavanja težina instanci i modela ovaj algoritam je dobio naziv ***Ada****ptive* ***Boost****ing*.

Set podataka je definisan:

Parametri algoritma su:

1. Ukupan broj iteracija - T
2. Bazni algoritam -

Koraci algoritma:

1. Inicijalizuju se težine svih instanci iz seta L prema formuli:
2. Radi se predikcija za svaki klasifikator {-1,1}.
3. Računa se greška svake instance

A nakon toga se računa ukupna greška prediktora:

kako bismo izračunali težinu prediktora (alfe) prema formuli:

1. U sledećem koraku se menjaju težine instanci:

1. U poslednjem koraku treniranja ansambla se računa konačna odluka sledećom jednačinom:

# Stacking

Ansambl Stacking je formulisao David Wolpert (*David H. Wolpert [1992]*). U literaturi algoritam se može naći i kao „Stacked generalization“ jer ga je Wolpert tako formalno nazvao. Ansambl je heterogen, odnosno sadrži više različitih osnovnih algoritama , čiji izlazi služe kao ulazni set podataka za meta algoritam (p - *eng. Parent*) algoritam koji daje konačan izlaz i on se u literaturi naziva generalizator (*eng. generalizer*). Cilj ovog ansambla je da postigne što je veću moguću generalizaciju konačnog modela. Za odabir najboljeg generalizatora, možemo napraviti testiranje i uporediti rezultate.

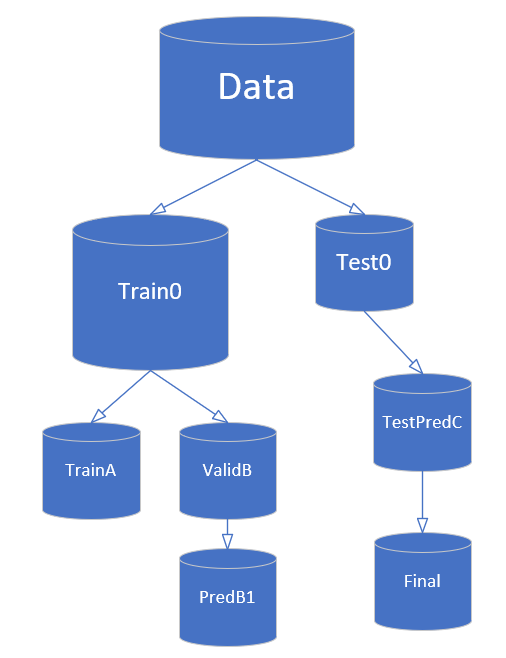
Podaci su definisani:

Parametri algoritma:

1. Osnovni algoritmi -
2. Algoritam generalizator -

Koraci algoritma:

1. Podaci se podele na 2 seta, i (test)
2. set se dalje deli na i .
3. Na setu treniraju se osnovni algoritmi .
4. Koristeći dobijene osnovne modele iz prethodnog koraka, napraviti predviđanja:
   1. Na validacionom setu i rezultate sačuvati u setu
   2. Na test setu i rezultate sačuvati u setu
5. Trenitati generalizator algoritam na setu
6. Predvideti izlaze modelom istreniranim u prethodnom koraku na setu



Slika 1 – Stacking podela podataka prilikom izvršavanja algoritma

# Gradient boosting

Ansambl *Gradient Boosting* predstavio je Fridman (Jerome H. Friedman [1999]). U ovom radu ovaj algoritam je obrađen kao osnova za *E****x****treme* ***G****radient* ***Boost*** poznatiji kao XGBoost. Gradient Boost tehnika pripada grupi sekvencionih homogenih ansambala, što znači da se treniranje jednog osnovnog modela izvršava sekvenciono. Kreirana je i unapređena verzija Gradient Boostinga, *Stochastic Gradient Boosting* *(Jerome H. Friedman [1999])* kako bi se ubrzalo izvršavanje, ali ona neće biti opisana u ovom radu. Gradient Boosting ima dve verzije algoritma, u zavisnosti da li se radi o klasifikaciji ili regresiji.

## Gradient Boosting za regresiju

Gradient Boosting za Regresiju (u daljem tekstu GBR) se može vrlo lako intuitivno razumeti ukoliko već posedujemo znanja o linearnoj regresiji i o Gradient Descent tehnici.

Intuicija algoritma

Gbr počinje tako što od svih instanci y izlazne varijable pravi jedan list (vektor), umesto stabla. Zatim se računa srednja vrednost instanci u tom listu. Ta srednja vrednost se koristi kao početno predviđanje za sve vrednosti y. U sledećem koraku računaju se pseudo residuali (*eng. Pseudo Residuals*) za svaku instancu u setu podataka, tako što se od oduzme inicijalna srednja vrednost . Reziduali se čuvaju u posebnoj koloni (kasnije će se koristiti za predviđanje). Nakon toga, Gbr kreira stablo koje uči na nezavisnim atributima i predviđa kolonu pseudo reziduala. Na taj način dobija se stablo koje u svojim listovima ima reziduale svih instanci. Dakle, svaka instanca može da se namapira do svog reziduala (greška prethodnog predviđanja) koji se nalazi u listovima stabla. Nakon kreiranja stabla računa se prosečna vrednost svakog lista (pseudo reziduali), i to će biti (gamma) vrednost (izlaz iz lista) koja će se kasnije uključiti u formulu za računanje novih predviđanja uključujući stopu učenja (nu) i prethodno predviđanje . Postupak se ponavlja dokle god se ne naravi M predviđanja i M stabala koji uče na greškama prethodnog. Iz prethodne rečenice možemo da zaključimo da postoji sličnost između AdaBoost i Gbr ansabmla, jer oba uče na greškama prethodnih modela i oba se izvršavaju sekvenciono. Nakon kreiranja svakogstabla smanjuju se vrednosti pseudo reziduala, i time se predviđanje kreće u pravom smeru. Konačna predikcija na nepoznatim podacima će biti izvršavanjem funkcije .

Podaci u setu su formulisani:

Parametri algoritma:

1. Ukupan broj stabala – M
2. Loss funkcija – F(x) koja može da se izvede
3. Maksimalan broj listova (najčešće između 8 i 32)
4. Stopa učenja –

Koraci algoritma:

1. Inicijalizacija modela sa konstantom (prvo predviđanje y instanci). Funkcija teži ta svede na minimum sumu Loss funkcije.

Sve instance stabla se ubacuju u jedan inicijalni sveobuhvatan list, vektor. Računa se diferencijalna loss funkcija . U ovom radu koristićemo popularnu funkciju koja smanjuje ukupan broj kalkulacija: *Loss funkcija*: , nakon toga izračunamo derivaciju loss funkcije = . Zatim izračunamo sumu za sve instance i dobijamo da je krajnji izlaz jednak prosečnoj vrednosti lista. (Primer kada imamo 2 instance)

-() + -() = 0

- - = 0

=

=

1. U drugom koraku, koristimo varijable m (označava broj individualnog stabla) i M (označava broj poslednjeg stabla). U ovom koraku se računaju pseudo reziduali:

za

Gde je r rezidual, i index lista u stablu a F() je vrednost predviđanja.

1. U trećem koraku sa pravi stablo odlučivanja nad x atributima da bi predvideli pseudo reziduali.
2. U četvrtom koraku se računaju izlazne vrednosti za svaki list stabla koje minimizuju ukupnu grešku:

Kada se ubaci loss funkcija, ponovo se dobije da je za svaki list stabla minimalna greška srednja vrednost.

1. Radi se ponovno predviđanje po sledećoj formuli:

Gde je stopa učenja, *m* broj stabla, prethodno predviđanje, izlazna vrednost za list *j* u stablu *m*. Suma je stavljena za slučaj da se instanca nađe u više listova.

1. Koraci 2,3,4,5 se izvršavaju dok se ne napravi maksimalan broj stabala. Predviđanje se vrši tako što se pozove poslednji

## Gradijent Boosting za klasifikaciju

Gradient Boosting za klasifikaciju (u daljem tekstu Gbk) se razlikuje od Gbr i potrebne su određene transformacije kako bi se gradijent mogao izračunati. Svi koraci algoritma ostaju isti, sa malim izmenama određenih koraka koji su navedeni u donjem tekstu:

1. Prvi koraj prilikom računanja inicijalne predikcije koristi funkciju: . Da bi se taj rezultat pretvorio u verovatnoću koristi se logistička funkcija:
2. Drugi korak prilikom računanja izlaznih vrednosti listova koristi sledeću transformaciju:

gde je prethodna verovatnoća.

1. U Petom koraku se rezultat predviđanja ponovo mora pretboriti u verovatnoće.

# XGBoost

Ovaj relativno nov skalabilni sistem mašinskog učenja, XGBoost, predstavio je instraživač *Tianqi Chen* [2016], a inicijalni release bio je 2014 godine.

Autor je napravio unapređenu verziju gradijent busting-a, i uveo nekoliko značajnih poboljšanja. Predstavio je *novel sparcity aware* algoritam, *weighted quantile* pristup odabira graničnih vrednosti, različite paterne za pristup keš memoriji računara, paralelno izvršavanje, kompresiju podataka, u cilju da se izgradi skalabilan *tree-boosting* sistem. Korišćenjem spomenutih nadogradnji, algoritam koristi mnogo manje resursa za obradu velike količine podataka.

Objektivna funkcija algoritma je:

Gde levi deo jednačine označava sumu loss funkcija za određenu instancu a desni deo je:

Iznad predstavljeni deo jednačine je regularizacioni deo, *w* je izlazna vrednost lista.

Parametri algoritma:

1. (lambda) – regulizacioni parametar
2. (gamma) – parametar za kontrolu orezivanja
3. (eta) – stopa učenja
4. - ukupan broj stabla
5. − maksimalna dubina stabla
6. – minimalno reziduala u listu
7. − loss funkcija
8. – ukupno kvantila

## XGBoost za regresiju

U trećem koraku opisan je jedan od više načina kreiranja stabala.

Podaci za rad algoritma su definisani:

Koraci za algoritma:

Zbog opširnosti algoritma, bolje preglednosti i jednostavnosti podeljen je na nekoliko segmenata.

Loss funkcija:

Čija je derivacija predstavljena u Gbr ansamblu.

Inicijalizacija:

1. Prvi korak je pravljenje inicijalne predikcije. Prema podrazumevanoj konfiguraciji , početno predviđanje za sve instance je 0.5.
2. Računanje pseudo reziduala.
3. Inicijalno kreiranje jedinstvenog lista u kome se nalaze svi reziduali. Računa se kvalitet lista ss score (pogledati formulu u sledećem segmenu za računanje kvaliteta).

Računanje kvaliteta stabla,:

U ovom segmentu predstavljena je formula za računanje *Similarity Score*-a:

gde je regulazacioni parametar a ukupan broj reziduala. Posebno je bitno obratiti pažnju da se kvadrira suma a ne svaki pseudo rezidual.

1. Kreira se stablo sa jednim *root* čvorom i dva lista.
2. Sortiraju se vrednosti za kolonu koja predstavlja.
3. Bira se granična vrednost čvora za dve susedne vrednosti u koloni i , tako što se počinje redom sa vrha ili dna liste instanci podataka u koloni. Indeksi instanci susednih vrednosti mogu da se ili inkrementiraju ili dekrementiraju za 1, dok se ne prođu sve instance iz kolone. Granična vrednost (*gv*) se računa sledećom formulom:
4. Prema gore navedenoj formuli za Similarity Score (*ss*), računa se vrednost za root čvor i za listove. List sa leve strane označićemo sa *Lss* a list sa desne *Rss* dok će root biti *RootSs*.
5. Nakon toga izračunava se dobit prema sledećoj formuli:

, gde je *Lss* jednaka ss skoru lista sa leve a *Rss* skoru sa desne strane. *RootSs* je ss skor za čvor iznad.

Kriterijum odabira graničnih vrednosti prema dobiti:

1. Algoritam bira graničnu vrednost koja ima najveću dobit, i po tom kriterijumu smešta pseudo reziduale u listove

Orezivanje stabla prema dobiti:

Parametar (gamma) predstavlja vrednost koja služi za odluku da li je potrebno uklonite čvor sa listovima prema dobitu koji oni zajedno nose. Ukoliko je razlika dobiti i gama vrednosti negativan broj (ukoliko je dobit manja od gamma-e) algoritam će orezati taj čvor iz stabla.

U trenutku orezivanja, sve instance listova vraćaju se u čvor koji ponovo postaje list.

Korisnički definisan parametar ***cover***, označava minimalni broj reziduala u listu. Podrazumevana vrednost je 1.

Korisnički definisan parametar ***deep*** označava maksimalnu dubinu stabla.

Kontrolisanje kvaliteta listova parametrom (lambda):

Glavni cilj za kontrolisanje kvaliteta listova lambda parametrom je sprečavanje pretreniranja modela regularizacijom i .

Formula za računanje kvaliteta vrednosti pseudo reziduala (segment za računanje kvaliteta stabla), u sebi sadrži lambda parametar koji ukoliko je veći, *ss* vrednost će se smanjiti u određenoj meri koja zavisi od ukupnog broja pseudo reziduala u listu. Što znači da ukoliko je lambda parametar veći, veće su šanse da će čvor biti orezan sa obzorom na to da će ss vrednost biti manja. Takođe zbog definisanja lambda parametra, iako je gamma jednak nuli, to ne znači da grana ne može da se oreže, jer ss može biti i negativan ukoliko je lambda veće od 0.

U formuli za računanje izlaznih vrednosti parametar lambda se koristi za regularizaciju odnosno smanjuje senzitivnost.

Računanje izlaznih vrednosti listova:

Računanje se vrši prema sledećoj formuli:

Predviđanje:

## XGBoost za klasifikaciju

Princip rada algoritma je jako sličan regresiji, s tim što u klasifikaciji koristimo verovatnoće kao konačnu predikciju pa formule moraju da se prilagode i neke vrednosti transformišu. Zapravo jedina razlika između klasifikacije i regresije je u loss funkciji, koja mora da se prilagodi izlazima.

Loss funkcija (negative likelihood):

Računanje kvaliteta stabla:

Računanje parametra **cover**:

Izlazna vrednost:

Transformacija izlaznih vrednosti u verovatnoću:

## Approximate Greedy Algorithm

Provera granične vrednosti za svaku susednu vrednost u svakog koloni kada je set podataka veliki može da traje jako dugo. Umesto toga, XGBoost koristi kvantile kao kandidate za granične vrednosti. Podrazumevana vrednost parametra *quantile* je oko 33 kvantila.

## Weighted quantile sketch

XGBoost ne koristi obične kvantile, već kreira otežane. Inače, kvantili se kreiraju tako što svaki sadrži isti broj instanci, u ovom ansamblu svaka instanca ima težinu, tako da je suma težina ista u svakom kvantilu. Za regresiju težine svih instanci su jednake 1, ali za klasifikaciju se računaju na osnovu prethodno dobijenih verovatnoća, i instance koje čija je prethodna predviđena verovatnoća blizu nule ili jedinice je manja (jer su predviđanja sigurnija), a instance gde je predviđanje bliže verovatnoći od 0.5 težina je veća jer su predviđanja nesigurna.

## Paralelno učenje

XGBoost ima mogućnost za paralelno učenje zbog korićnjenja Approximate Greedy algoritma.

## Sparsity-aware Split Finding

XGBoost može da radi sa nedostajući podacima i to dosta olakšava samu pripremu podataka. Ova tehnika pridružuje nedostajuće podatke listovima u koraku kada se računa kvalitet čvora. To radi na način da ih pridruži i levoj i desnoj strani. Na kraju nepoznate vrednosti se pridružuju strani sa najvećom dobiti.

## Cache-Aware Access

XGBoost stavlja skorove i gradijente u CPU keš memoriju koja u tom slučaju veoma brzo radi proračune.

## Out-Of-Core Computations

XGBoost koristi kompresiju podataka kako bi se što manje memorije zauzimalo na hard diskovima (CPU će brže da dekompresuje nego što će da se podaci iščitaju sa hard diska). Takođe XGBoost može koristiti Shreading tehniku da istovremeno čita iz više baza.

# Evaluacije ansambla

U prilogu je .xlsx fajl u kome se nalaze rezultati merenja performansi različitih ansambla.



Kao mere evaluacije korišćene su sledeće metrike:

1. Klasifikacija
   1. Accuracy **(TP+TN)/(TP+TN+FP+FN)**
   2. Precision **(TP)/(TP+FP)**
   3. Recall **(TP)/(TP+FN)**
   4. F1Score **2\*((PRE\*RCL)/(PRE+RCL))**
   5. Trajanje u sekundama
2. Regresija
   1. Mean squered error

* 1. Trajanje u sekundama

Prilikom testiranja korišćeni su ansambli:

* XGBoost
* RandomForest
* Adaboost
* GradientBoosting

Specifikacija testne mašine:

16GB RAM, I7 8550U CPU DualCore 1.99GHz, Win10 os, SSD 500GB

Napomena: XGBoost ansambl nije mogao da koristi napredne funkcije paralelizacije jer su testovi vršeni na jednoj mašini.

## Opažanja prilikom testiranja i moguće hipoteze

Rezultati merenja variraju u zavisnosti od seta podataka. Ovde će biti razmatrane hipoteze koje su u većini testova (svaki na različitom setu podataka) bile potvrđene kao tačne (80% ili više).

AdaBoost ansambl se pokazao kao najbrži algoritam za treniranje modela u 80% setova podataka za klasifikaciju.

XGBoost ansambl se pokazao kao model za najboljom Accuracy ocenom na testnim podacima u 80% setova podataka za klasifikaciju.

XGBoost ansambl se pokazao kao model sa najboljom Precision ocenom na testnim podacima u 80% setova podataka za klasifikaciju.

XGBoost ansambl se pokazao kao model sa najboljom F1 ocenom na testnim podacima u 80% setova podataka za klasifikaciju.

Na jako malim setovima podataka kao što je *HeartDisease* AdaBoost ansambl se pokazao kao najlošiji prediktor za sve mere evaluacije (sem za brzinu treniranja i predviđanja).

Na velikom setu podataka IOT XGBoost ansambl je pokazao svoju moć prediktivnosti, i u svim merama evaluacije zadržao je vodeću poziciju.

Na testiranju podataka sa nebalansiranim podacima ansambl GradientBoosting je pokazao dosta lošije ocene Recall-a i F1 score-a, od drugih prediktora, dok je XGBoost imao najbolje ocene svih mera evaluacija.

Prilikom testiranja setova podataka za regresiju, najlošije ukupno vreme izvršavanja imao je ansambl RandomForest.

# Dalji rad

## Testiranje na klasterovanim mašinama

Kako bi ceo kapacitet XGBoost ansambla bio istestiran, potrebno je koristiti veliku računarsku moć, što u prevodu znači veliki proj procesora, velika količina RAM memorije, distribuirane baze podataka. Cilj ovakvog testiranja bi bio izvođenje zaključaka o performansama paralelizacije.

## Pronalaženje najboljih parametara ansambla

Koristeći tehniku *GridSearch* pronaći najbolje parametre ansambla i ponovo evaluirati modele. Ovaj zadatak iziskuje velike performanse računara pogotovo na velikim setovima podataka.

## Pronalaženje najkorisnijih atributa

Koristeći tehniku SelectKBest pronaći najkorisnije atribute iz dataset-a i ponovo evaluirati modele.

## Redukovati dimenzije radi povećanja brzine izvršavanja

Koristeći tehniku *Analiza Glavnih Komponenti* koristiti transformisani skup atributa koji kumulativno nosi 98% varijanse i ponovno evaluirati modele. Uporediti ukupno vreme izvršavanja treniranja modela pre i posle redukovanja dimenzija.

# Reference

1. Leo Breiman (1996), *Bagging Predictors* , Machine learning 24, 123-140, Statistics department, University of California, Berkeley, CA 94720
2. Jerome H. Friedman (1999), *Greedy Function Approximation: A Gradient Boosting Machine*, IMS Reitz Lecture.
3. Jerome H. Friedman (1999), Stochastic Gradient Boosting
4. Yoav Freund, Robert E. Shapire (1999), *A Short Introduction To Boosting*, AT&T Labs – Research Shannon Laboratory, 180 Park Avenue, NJ, USA
5. Robert E. Shapire, *Explaining Adaboost*
6. David H. Wolpert, *Stacked Generalization*, Complex Systems Group, Theoretical Division, And Center For Non-linear Studies, Los Alamos
7. Leo Breiman (2001), *Random Forests*, Statistics Department, University of California Berkeley, CA 94720
8. Michael Kearns (1988), *Thoughts on Hypothesis Boosting*, Machine Learning Class Project
9. Tianqi Chen, Carlos Guestin (2016), *XGBoost: A Scalable Tree Boosting System*, University of Washington.

# Izvori podataka

1. <http://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe+Endgame>
2. <http://archive.ics.uci.edu/ml/datasets/Skin+Segmentation>
3. <http://archive.ics.uci.edu/ml/datasets/Somerville+Happiness+Survey>
4. <http://archive.ics.uci.edu/ml/datasets/Mammographic+Mass>
5. <https://www.kaggle.com/mlg-ulb/creditcardfraud>
6. <http://archive.ics.uci.edu/ml/datasets/Airfoil+Self-Noise>
7. <http://archive.ics.uci.edu/ml/machine-learning-databases/parkinsons/telemonitoring/>
8. <https://www.kaggle.com/uciml/red-wine-quality-cortez-et-al-2009>
9. <http://archive.ics.uci.edu/ml/datasets/Computer+Hardware>