**Support Vector Machines**

Support Vector Machines არის supervised learning მოდელი, რომელიც ხშირად და აქტიურად გამოიყენება კლასიფიკაციის ამოცანებში მისი რამდენიმე თვისების გამო. მისი არსი მდგომარეობს შემდეგში: svm იღებს training data-ს და მის სივრცეს ჰყოფს ლეიბლებისადმი შესაბამის ქვესივრცეებად. ავლებს decision boundariy-ებს.

Svm-ის დადებითი მხარეებია: ნაკლებ მეხსიერებას მოიხმარს, კარგად მუშაობს იმ შემთხვევებშიც, როცა სეიმფლების რაოდენობა ნაკლებია განზომილებაზე, იყენებს კერნელ ფუნქციებს. კერნელი არის ერთგვარი მსგავსების ფუნქცია, რომელიც გვეუბნება რამდენად ახლოსაა ერთი სეიმფლი მეორე, უმეტესად ახალ სეიმფლთან და მას ეფუძნება სვმ-ის მოდელი.

კერნელი svm-ის ცენტრალური ნაწილია. მე არჩევანი შევაჩერე gaussian (ასევე ე.წ. rbf) kernel-ზე. მისი ფორმულაა:



მისი მუშაობის პროცესი შემდეგია, ითვლება ევკლიდური მანძილი ორ სეიმფლს შორის. თუ ისინი ახლოსაა, მაშინ ექპონენციალური ფუნქცია 1-ს უახლოვდება, ხოლო თუ ისინი შორსაა, მაშინ 0-ს უახლოვდება. და ამგვარად იქმნება ახალი feature.

Rbf-ზე არჩევანი შევაჩერე იმიტომ, რომ ის არის ერთ-ერთი ყველაზე გავრცელებული კერნელი, ლინეართან ერთად, რომელიც ამ ჩვენს მონაცემებს არანაირად არ ესადაგებოდა, რადგან მათი წრფივად დაყოფა კარგ სიზუსტეს არ გვაძლევს. თუმცა, როგორც კი გადავწყვიტე rbf-ის არჩევა, წავაწყდი ერთ-ერთ მთავარ პრობლემას, რომელიც მალევე გამოვასწორე. რბფ ხდება თითქმის უსარგებლო და ბევრად უარესი ვიდრე თუნდაც წრფივი კერნელი თუ კი მონაცამებს არ გავუკეთებდთ ნორმალიზებას ანუ სქეილინგს. ზემოთ მოცემული ექსპონენციალური ფუნქცია არის აგებული ორ სეიმფლს შორის სხვაობაზე, ევკლიდურ მანძილზე. ხოლო ერთი feature შეიძლება იზომებოდეს 1-10მდე რაღაც რიცხვებში და მეორე 10 000დან 1მლნ-მდე, მაგალითად სახლის ოთახების რაოდენობა vs სახლის ფასი. სქეილინგის გაკეთების შემდეგ კი rbf 86.9% accuracy-ს იძლევა.

მეორე ასპექტი რაც svm-ის აგებისას აუცილებლადაა გასათვალისწინებელი არის არჩევა one vs one და one vs all იმპლემენტაციებს შორის. მე არჩევანი გავაკეთე უკანასკნელზე, რადგან იგი ყველაზე ხშირად გამოიყენება და ყველაზე პოპულარულია.

დიდი მნიშვნელობა აქვს C-ს არჩევასაც, ჩემს შემთხვევაში საჭირო გახდა დიდი C-ს აღება, რადგან მაღალი ვარიაცია იყო.

**ნეირონული ქსელები**

ნეირონული ქსელები არის მოდელი, რომელიც ინფორმაციის დასწავლას ცდილობს ცოცხალი ორგანიზმის ტვინის ნეირონების მოდელის მსგავსად. იგი შედგება რაღაც რაოდენობის ნეირონისგან, რომლებიც სწავლობენ სხვადასხვა მოდელს, მასში არსებულ პატერნებსა და კავშირებს შემავალ და გამომავალ ინფორმაციას შორის.

კონვოლუციური ნეირონული ქსელები არის ნეირონული ქსელების გაუმჯობესებული ვერსია, რომელიც ბევრად უკეთ ერგება კონკრეტულად სურათის კლასიფიკაციის ამოცანას. მისი სტრუქტურა არის შემდეგი: კონვოლუციური ლეიერები, მაქსიმალური პულინგი, მთლიანად დაკავშირებული ლეიერი და აუთფუთ ლეიერი.

კონვოლუციური ლეიერის მეშვეობით ვიღებთ რაღაც a x a ზომის ნაწილს მონაცემებიდან და რაღაც ბიჯებით გადაუყვება მთელ მონაცემებს ამ ზომის კვადრატებად და მათგან იღებს ინფორმაციას. იგი მოსახერხებელია, იმიტომაც რომ აუთფუთი რომელიც შემდეგ ლეიერს გადაეცემა მცირდება, რაც fully connected layer-ების შემთხვევაში არ ხდება და მეხსიერების ძალიან გაზრდას ვუშლით ხელს კონვოლუციური ლეიერებით. ჩვენი ამოცანის შემთხვევაში ავირჩიე სამი კონვოლუციური ლეიერი, 32, 32 და 64 ფილტრების რაოდენობებით და 3 x 3 ზომით.

შემდეგ ვიყენებ რელუ აქტივაციას, რომელიც ამზადებს ინფუთს შემდეგი ლეირისთვის.

ყოველი კონვოლუციის შემდეგ გამოიყენება მაქსიმალური პულინგ ლეიერი რათა ნაკლები პარამეტრი იყოს გამოყენებული, რომ არ მოხდეს ოვერფითინგი. ყველაზე ხშირი და ასევე ჩემს კოდში გამოყელებული მაქს ფულინგ ლეიერია 2x2 ზომის მათგანი, ნაბიჯით 2, და იგი დაახლოებით ორჯერ ამცირებს ინფუთის ზომას.

ასევე გამოყენებული მაქვს დროფაუთი, ოვერფიტინგისგან თავის ასარიდებლად.

ბოლოს არის fully connected layer Dense ლეიერი 512 ნეირონით და ბოლოს სოფთმაქსი რომელიც უკვე მიღებული, ბოლო ინფუთის მინიჭება/კლასიფიკაციას ახდენს ამ შემთხვევაში 10 ლეიბლის მიხედვით.

ლეიერების რაოდენობა და შესაბამისი რიცხვები აღებულია სტანდარტული და ხშირად გამოყენებული მაგალითების მიხედვით და მათი “უკეთესობა” ძირითადად გადარჩევის პრინციპს ემყარება, თუმცა ზემოთ მოცემული სტრუქტურა (კონვ - აქტივაცია - პულინგი, შემდეგ დროფაუთი, შემდეგ დენსი და სოფთმაქსი) ყველაზე ოპტიმალურია და რიცხვების შერჩევაზე დადის ამოცანა.

მაქვს აღებული 15 ეპოქი, ისე რომ არ მომხდარიყო ოვერფითინგი, რაც დავადგინე ლერნინგ ქარვის ზრდა/კლების მიხედვით. ასევე აღსანიშნავია, რომ ნეირონულ ქსელებშიც ვიყენებ სქეილინგს, რამაც, 2-3%-ით აწია ჩემი აქურასი, და მისი გამოყენება თითქმის ყოველთვის ძალიან მომგებიანია.

მაქვს გამოყენებული sparse\_categorical\_crossentropy, რათა თავი ამერიდებინა Y-ის one hot-ად გადაყვანისთვის. და ოტპიმიზატორად მაქვს გამოყენებული ადამი, რომელიც ასევე სტანდარტული და ყველაზე გავრცელებულია. ასევე უკეთესი შედეგი მომცა, ვიდრე 'rmsprop'-მა.

**Notes:**

გამოყენებული მაქვს cross validation ორივე მოდელის გამოყენებისას, რათა გამეგო ტესტზე მორგებული იყო თუ არა ჩემი იმპლემენტაცია და რომ ნამდვილად მუშაობდა იგი სხვა ტიპის ტესტებზეც. ამისათვის ავიღე თრეინინგ სეთის რაღაც ნაწილი და გამოვიყენე ტესტად, ხოლო დანარჩენი ჩვეულებრივ ვაჭამე.

ასევე გამოვიყენე მოდელის შენახვა, რათა ჩემმა კლასიფიკაციის ამოცანამ უფრო კომპაქტურად იმუშაოს და არ იყოს საჭირო ყოველ ახალ ტესტზე ახლიდან შესწავლა თრეინინგ სეთის.

ნეირონული ქსელებისთვის გამოვიყენე keras ბიბლიოთეკა მისი კომპაქტურობისა და მოხერხებულობის გამო. ხოლო, svm-ისთვის scikit learn, რომელიც უკონკურენტოა.

პერსონალური ფავორიტია: svm.