**пол1. Линеарна регресија (Linear Regression)**

* **Како работи:**
  + Го моделира односот меѓу зависна променлива YYY и една или повеќе независни променливи XXX како права линија.
  + Формулата е: Y=b0+b1X1+b2X2+...+bnXn+ϵY = b\_0 + b\_1X\_1 + b\_2X\_2 + ... + b\_nX\_n + \epsilonY=b0​+b1​X1​+b2​X2​+...+bn​Xn​+ϵ каде b0b\_0b0​ е пресекот, bib\_ibi​ се коефициенти, и ϵ\epsilonϵ е грешката.
  + Целта е да се минимизира грешката (разликата меѓу предвидените и вистинските вредности).
* **Пример:**  
  Предвидување на цената на куќа врз основа на големината (во квадратни метри).
* **Толкување на резултати:**
  + **Коефициенти (bib\_ibi​)** покажуваат колку YYY се менува за една единица промена во XiX\_iXi​.
  + **R2R^2R2** покажува колкав процент од варијацијата на YYY е објаснета со моделот.
  + Ако p pp-вредноста е мала, значи дека XiX\_iXi​ е статистички значајна за моделот.

**2. Логистичка регресија (Logistic Regression)**

* **Како работи:**
  + Користи линеарна комбинација од влезните променливи за да предвиди **веројатност** за припаѓање во одредена класа (0 или 1).
  + Формулата е: P(Y=1∣X)=11+e−(b0+b1X1+b2X2+...+bnXn)P(Y=1|X) = \frac{1}{1 + e^{-(b\_0 + b\_1X\_1 + b\_2X\_2 + ... + b\_nX\_n)}}P(Y=1∣X)=1+e−(b0​+b1​X1​+b2​X2​+...+bn​Xn​)1​
* **Пример:**  
  Дали клиент ќе плати заем навреме (1 = ќе плати, 0 = нема да плати).
* **Толкување на резултати:**
  + Веројатноста е изразена во проценти (на пр., 80% веројатност за успех).
  + **Коефициентите** покажуваат како зголемување на XiX\_iXi​ влијае врз веројатноста.
  + **Confusion Matrix, Precision, Recall, F1-Score:** Мерки за оценка на перформансите.

**3. Decision Trees**

* **Како работи:**
  + Ги дели податоците на основа на карактеристики во форма на прашања (на пр., "Дали приход > 50K?").
  + Секоја поделба се прави за да се максимизира информацијата или да се минимизира грешката.
* **Пример:**  
  Класификација на вид на овошје врз основа на боја, големина, и тежина.
* **Толкување на резултати:**
  + Дрвото покажува како одлуките се донесуваат.
  + **Лисјата:** Ги претставуваат крајните предвидувања.
  + **Depth and purity:** Плитко дрво може да биде недоволно точно, а длабоко дрво може да претренира.

**4. Random Forest**

* **Како работи:**
  + Генерира многу одлуки дрва на различни подмножества од податоците и ги комбинира резултатите (на пр., со гласање).
  + Помага да се намали претренирањето кое се случува кај едно дрво.
* **Пример:**  
  Предвидување на дали ќе заврне дожд врз основа на температура, влажност, и притисок.
* **Толкување на резултати:**
  + **Feature importance:** Показател колку секоја карактеристика придонесува за одлуките.
  + **Accuracy, Precision, Recall:** Мерки за оценка.

**5. Gradient Boosting (XGBoost, LightGBM)**

* **Како работи:**
  + Генерира серија дрва каде што секое ново дрво се обидува да ги коригира грешките од претходните.
  + Корисно за табуларни податоци со сложени врски.
* **Пример:**  
  Оценување на ризик за кредит базирано на финансиски историјат.
* **Толкување на резултати:**
  + **SHAP (SHapley Additive exPlanations):** Современ метод за интерпретирање на влијанието на карактеристиките.
  + Ги користите истите мерки како кај Random Forest.

**6. Neural Networks**

* **Како работи:**
  + Ги имитира нервните клетки преку слоеви на "неврони" кои учат сложени врски од податоците.
  + Секој "неврон" применува математичка трансформација (на пр., активациона функција).
* **Пример:**  
  Класификација на слики (на пр., препознавање на лица).
* **Толкување на резултати:**
  + **Тешко за интерпретација:** Но можат да се користат визуализации како Grad-CAM за разбирање.
  + Перформанси се мерат со метрики како **accuracy, cross-entropy loss.**

**1. Греедс Сearch (Greedy Search)**

Греедс пребарувањето (или алгоримот на алчност) е една од стратегиите за решавање на оптимизациски проблеми, која работи така што на секој чекор одбира најдобар локален избор со цел да се најде глобално оптимално решение.

**Како работи?**

* **Греедс** изведува серија одлуки кои го водат кон решение, но за секој чекор одбрал најдобра опција во моментот без да се грижи за последиците.
* Главната идеја е да се направи локално оптимално решение во секој чекор со надеж дека тоа ќе води до глобално оптимално решение.
* Греедс алгоримот не гарантира глобално оптимално решение, но обично дава добро решение за многу проблеми.

**Пример:**

* **Пример:** Проблем на рюкзак (knapsack problem), каде треба да изберете кои предмети да ги ставите во рюкзак со ограничен капацитет за да го максимизирате вкупниот вредност:
  1. Изберете предмет со највисока вредност по тежина.
  2. Додајте го во рюкзакот и повторете додека не го исполните капацитетот.
* Греедс методата може да не даде најдобро решение во сите случаи, но често е многу побрза од другите алгоритми.

**Предности и Недостатоци:**

* **Предности:** Брз и лесен за имплементација.
* **Недостатоци:** Може да не го најде глобално оптималното решение (локално оптимално решение е само во одредени случаи).

**2. Градиентен Десцент (Gradient Descent)**

Градиентниот десцент е оптимизациски алгоритам кој се користи во машинско учење и статистика за минимизирање на функција (или максимизирање, ако се користи за оптимијање на некоја метрика) преку пресметување на **градиентот** на функцијата во дадена точка и затем го движите решението во насока која води до минимум.

**Како работи?**

* Почнуваме од случаен почетен пункт и на секој чекор пресметуваме **градиентот** (или деривативата) на функцијата.
* Градиентот покажува насока на најстрмниот пораст. За минимизирање, се движиме во спротивната насока на градиентот.
* Формулата за градиентен десцент е: θ=θ−α⋅∇J(θ)\theta = \theta - \alpha \cdot \nabla J(\theta)θ=θ−α⋅∇J(θ)
  + **θ\thetaθ:** Параметри на моделот
  + **α\alphaα:** Скорост на учење (learning rate), која го контролира степенот на промена во секој чекор
  + **∇J(θ)\nabla J(\theta)∇J(θ):** Градиентот на функцијата на грешка
* Процесот продолжува додека не се достигне минимум или додека не се постигне определен број на итерации.

**Пример:**

* Кога тренирате модел за линерна регресија, користите градиентен десцент за да го минимизирате вкупниот квадратен остаток (најмалите квадрати) помеѓу предвидените и реалните вредности.

**Предности и Недостатоци:**

* **Предности:** Може да се користи за различни типови на оптимизациски проблеми и е многу корисен во машинското учење.
* **Недостатоци:** Потребно е внимателно да се избере **скороста на учење**, бидејќи премала вредност може да направи процесот бавен, додека премногу голема вредност може да резултира со недостигање на минимум.

**3. Линеарно Програмирање (Linear Programming)**

Линеарното програмирање е техника за оптимизација која се користи за решавање на проблеми кои можат да бидат моделирани со линеарни функции. Главната цел на линеарното програмирање е да најдеме најдобро решение за некоја целна функција (обично за максимизација или минимизација), при што се исполнуваат одредени ограничувања.

**Како работи?**

* Линеарните програми вклучуваат оптимизација на линеарна функција (целна функција) со ограничувања кои се исто така линеарни.
* Стандардната форма на линеарното програмирање изгледа вака: Максимизирај/Минимизирај: c1x1+c2x2+⋯+cnxn\text{Максимизирај/Минимизирај: } c\_1 x\_1 + c\_2 x\_2 + \dots + c\_n x\_nМаксимизирај/Минимизирај: c1​x1​+c2​x2​+⋯+cn​xn​ под услов: a11x1+a12x2+⋯+a1nxn≤b1a\_{11} x\_1 + a\_{12} x\_2 + \dots + a\_{1n} x\_n \leq b\_1a11​x1​+a12​x2​+⋯+a1n​xn​≤b1​ a21x1+a22x2+⋯+a2nxn≤b2a\_{21} x\_1 + a\_{22} x\_2 + \dots + a\_{2n} x\_n \leq b\_2a21​x1​+a22​x2​+⋯+a2n​xn​≤b2​ x1,x2,…,xn≥0x\_1, x\_2, \dots, x\_n \geq 0x1​,x2​,…,xn​≥0
  + **c1,c2,…,cnc\_1, c\_2, \dots, c\_nc1​,c2​,…,cn​**: Коэффициенти на целната функција
  + **aija\_{ij}aij​**: Коэффициенти на ограничувањата
  + **bib\_ibi​**: Пракса за секој од условите
  + **x1,x2,…,xnx\_1, x\_2, \dots, x\_nx1​,x2​,…,xn​**: Променливи кои се оптимизираат

**Пример:**

* **Пример:** Линеарно програмирање може да се користи за проблеми како што е оптимизација на производството: како да се произведат различни видови производи со ограничени ресурси за да се максимизира профитот.

**Предности и Недостатоци:**

* **Предности:** Линеарното програмирање е многу ефективно за решавање на голем број индустриски оптимизациски проблеми.
* **Недостатоци:** Може да биде ограничено ако функциите или ограничувањата не се линеарни.

**Заклучок:**

* **Греедс Сearch**: Брзо решавање со локални оптимуми, но не секогаш глобално оптимално.
* **Градиентен Десцент**: Ефикасен за оптимизација на модели, но треба внимателно да се избере параметарот за учење.
* **Линеарно Програмирање**: Оправдано за решавање на оптимизациски проблеми со линеарни функции и ограничувања.

**5. Како се Интерпретираат Резултатите?**

* **Точност (Accuracy)**: Ова е наједноставната метрика и претставува процент на точни предвидувања (т.е., колку проценти од класифицираните примероци беа правилно класифицирани).
* **Матрица на конфузија (Confusion Matrix)**: Оваа матрица ја покажува распределбата на точните и неточните предвидувања по категории. Вклучува четири важни вредности:
  + **True Positives (TP)**: Примероци кои се точно предвидени како позитивни.
  + **True Negatives (TN)**: Примероци кои се точно предвидени како негативни.
  + **False Positives (FP)**: Примероци кои погрешно се предвидени како позитивни.
  + **False Negatives (FN)**: Примероци кои погрешно се предвидени како негативни.
* **Precison, Recall, F1 Score**: Овие метрики се користат за подлабока анализа на перформансите на класификацискиот модел. **Precision** се однесува на точноста на позитивните предвидувања, додека **Recall** покажува колку од вистинските позитивни примероци биле успешно идентификувани.

**Кога да користите која метрика?**

* **За балансирани датасети**:
  + Точноста, Precision, Recall и F1 Score се добри метрики.
  + F1 Score е корисен ако сакате да ги балансирате Precision и Recall.
* **За небалансирани датасети**:
  + Точноста може да биде незначајна, бидејќи може да покажува висок процент само за доминантната класа.
  + **Precision, Recall и F1 Score** за помалку застапената класа се посоодветни.
  + **AUC-ROC и Precision-Recall криви** се поефективни за анализа на перформансите на моделот во случаи на небалансирани податоци.

**Логистичка регресија**

**Цел**:  
Логистичката регресија се користи за предвидување на веројатноста за бинарен исход (на пример, *„default = Yes“* или *„default = No“*), наместо директно моделирање на одговорната променлива YY.

**Основи на логистичката регресија:**

* Наместо директно да се моделира YY, логистичката регресија ја моделира **веројатноста дека YY припаѓа на одредена категорија**.
* На пример, за податоците за кредитно задолжување (*Default data*), логистичката регресија ја моделира веројатноста за неплаќање на долговите (*default*).

**Пример:**

* Веројатноста за неплаќање при даден баланс (balancebalance) е напишана како: Pr(default=Yes∣balance)=p(balance)Pr(default = Yes | balance) = p(balance)
* **Дијапазон на вредности**: Веројатноста p(balance)p(balance) е секогаш помеѓу 0 и 1.
* **Одлука за предвидување**:
  + Ако p(balance)>0.5p(balance) > 0.5, може да се предвиди дека ќе има неплаќање.
  + Ако компанијата сака да биде поконзервативна, може да постави понизок праг, како p(balance)>0.1p(balance) > 0.1.

**Логистички модел**

1. **Проблем со линеарна регресија**: Ако пробаме да ја моделираме веројатноста p(X)p(X) со линеарен модел p(X)=β0+β1Xp(X) = \beta\_0 + \beta\_1X, добиваме резултати кои може да се надвор од интервалот [0, 1].
2. **Решение**: Наместо тоа, се користи *логистичката функција*, која секогаш дава вредности помеѓу 0 и 1.

**Односи во логистичкиот модел:**

1. **Оддс (Odds)**:  
   Односот p(X)1−p(X)\frac{p(X)}{1-p(X)} се нарекува *оддс* и може да варира од 0 до бесконечност.
   * Вредност блиска до 0 покажува многу мала веројатност за настан.
   * Вредност блиска до бесконечност покажува многу голема веројатност.

**Примери:**

* Ако p(X)=0.2p(X) = 0.2, тогаш Odds=0.21−0.2=14Odds = \frac{0.2}{1-0.2} = \frac{1}{4}. Ова значи дека 1 од 5 лица во ваква ситуација ќе има неплаќање.
* Ако p(X)=0.9p(X) = 0.9, тогаш Odds=0.91−0.9=9Odds = \frac{0.9}{1-0.9} = 9. Ова значи дека 9 од 10 лица ќе имаат неплаќање.

1. **Логит (Logit)**:  
   Логаритмот на оддс, log⁡(p(X)1−p(X))\log \left( \frac{p(X)}{1-p(X)} \right), е линеарна функција: log⁡(p(X)1−p(X))=β0+β1X\log \left( \frac{p(X)}{1-p(X)} \right) = \beta\_0 + \beta\_1X

**Проценка на параметрите (β0\beta\_0 и β1\beta\_1):**

* **Метод**: Максимална веројатност (*Maximum Likelihood Estimation*).
* **Интерпретација**: Се проценуваат β0\beta\_0 и β1\beta\_1 така што предвидената веројатност p^(xi)\hat{p}(x\_i) најмногу ќе одговара на набљудуваните податоци.

**Пример за пресметка:**

* **Добиени параметри**: β0=−10.6513\beta\_0 = -10.6513, β1=0.0055\beta\_1 = 0.0055.
* За лице со баланс од $1,000: p(X)=11+e−(β0+β1×1000)≈0.0077 (0.77%)p(X) = \frac{1}{1 + e^{-(\beta\_0 + \beta\_1 \times 1000)}} \approx 0.0077 \, (0.77\%)
* За лице со баланс од $2,000: p(X)=11+e−(β0+β1×2000)≈0.586 (58.6%)p(X) = \frac{1}{1 + e^{-(\beta\_0 + \beta\_1 \times 2000)}} \approx 0.586 \, (58.6\%) Ова покажува дека зголемувањето на балансот значително ја зголемува веројатноста за неплаќање.

**Важни мерки за класификација и дијагностичко тестирање**

Овие метрики се користат за проценка на перформансите на класификациските модели и тестови кои одлучуваат за присуство или отсуство на одреден настан или карактеристика (на пример, болест, дефект, итн.).

**1. Точност (Accuracy)**

* **Дефиниција**: Делот од точните предвидувања (и позитивни и негативни) од вкупниот број на примери.
* **Формула**: Accuracy=TP+TNTP+TN+FP+FN\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} Каде:
  + TPTP: Точно позитивни (правилно предвидени позитивни случаи).
  + TNTN: Точно негативни (правилно предвидени негативни случаи).
  + FPFP: Лажно позитивни (негативни случаи погрешно предвидени како позитивни).
  + FNFN: Лажно негативни (позитивни случаи погрешно предвидени како негативни).

**2. Прецизност (Precision)**

* **Дефиниција**: Делот од точните позитивни предвидувања од сите позитивни предвидувања.
* **Формула**: Precision=TPTP+FP\text{Precision} = \frac{TP}{TP + FP}
  + Оваа метрика покажува колку од предвидените позитивни случаи се навистина точни.
  + Корисна кога грешките од типот FPFP се скапи (на пример, лажно обвинение).

**3. Повик (Recall) или Осетливост (Sensitivity)**

* **Дефиниција**: Делот од вистинските позитивни случаи што се правилно идентификувани.
* **Формула**: Recall=Sensitivity=TPTP+FN\text{Recall} = \text{Sensitivity} = \frac{TP}{TP + FN}
  + Оваа метрика е важна кога сакате да ги фатите сите позитивни случаи (на пример, дијагностицирање на сериозна болест).

**4. Осетливост (Sensitivity) *(се поклопува со Recall)***

* **Дефиниција**: Истата како Recall, покажува колкав дел од вистинските позитивни случаи моделот ги детектира правилно.
* **Пример**: Ако моделот има висока осетливост, тој ретко ќе пропушти позитивни случаи.

**5. Специфичност (Specificity)**

* **Дефиниција**: Делот од вистинските негативни случаи што се правилно идентификувани.
* **Формула**: Specificity=TNTN+FP\text{Specificity} = \frac{TN}{TN + FP}
  + Оваа метрика е важна кога сакате да избегнете погрешни позитивни предвидувања (на пример, непотребно третирање за болест).

**Врска помеѓу метриките**

* **Precision** и **Recall** често се комплементарни. Зголемување на едната метрика може да ја намали другата, па затоа се користи **F1 мера** за баланс.
* **Accuracy** може да биде варлива ако класите се несразмерно застапени (на пример, ако 90% од примерите се негативни, моделот што секогаш предвидува "No" ќе има висока точност).

**Резиме**

* **Accuracy**: Колку е точен моделот воопшто.
* **Precision**: Колку од позитивните предвидувања се точни.
* **Recall/Sensitivity**: Колку од вистинските позитивни случаи се фатени.
* **Specificity**: Колку од вистинските негативни случаи се фатени.

**Како работат Precision и Recall**

Овие две метрики се користат за проценка на перформансите на класификациски модели, особено во ситуации каде што има нерамномерна распределба на класите или каде што позитивните случаи се критични.

**Precision (*Прецизност*)**

* **Што мери?**  
  Колку од предвидените позитивни случаи (TPTP) се навистина точни.
  + Се фокусира на точноста на позитивните предвидувања.
  + Се намалува кога има многу *Лажно Позитивни* (FPFP).
* **Пример**:  
  Во медицински тест, ако моделот предвиди дека 10 лица имаат болест, но само 7 од нив навистина се болни:

Precision=TPTP+FP=77+3=0.7 (70%)\text{Precision} = \frac{TP}{TP + FP} = \frac{7}{7 + 3} = 0.7 \, (70\%)

* + Прецизност од 70% значи дека 7 од 10 позитивни предвидувања се точни.
* **Кога е важна?**  
  Кога е критично да се избегнат погрешни позитивни предвидувања, на пример:
  + При криминални истраги (да не се обвини невино лице).
  + Во препораки за критични одлуки, како одобрување кредити.

**Recall (*Осетливост или Повик*)**

* **Што мери?**  
  Колку од вистинските позитивни случаи (TPTP) моделот правилно ги идентификувал.
  + Се намалува кога има многу *Лажно Негативни* (FNFN).
* **Пример**:  
  Ако 50 лица навистина имаат болест, а моделот правилно открие само 40 од нив:

Recall=TPTP+FN=4040+10=0.8 (80%)\text{Recall} = \frac{TP}{TP + FN} = \frac{40}{40 + 10} = 0.8 \, (80\%)

* + Recall од 80% значи дека моделот открил 80% од вистинските позитивни случаи.
* **Кога е важна?**  
  Кога е критично да се фатат сите позитивни случаи, на пример:
  + Дијагностицирање сериозни болести.
  + Откривање на опасности, како спам е-пошта или малициозен софтвер.

**Разлика меѓу Precision и Recall**

* **Precision** гледа само позитивните предвидувања и оценува дали тие се точни.
* **Recall** гледа сите вистински позитивни случаи и оценува дали моделот успеал да ги пронајде.

**Баланс меѓу Precision и Recall**

* Зголемување на **Precision** може да намали **Recall** и обратно.
  + Ако моделот е многу строг (ретко предвидува позитивно), Precision ќе биде висок, но Recall низок.
  + Ако моделот е попустлив (често предвидува позитивно), Recall ќе биде висок, но Precision низок.
* **F1-Score**:  
  Метрика која го балансира Precision и Recall:

F1=2⋅Precision⋅RecallPrecision+RecallF1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}

Со ова разбирање, Precision е важна за избегнување лажни позитивни, а Recall е важна за минимизирање лажни негативни.

**Матрица на конфузија (Confusion Matrix)**

Матрицата на конфузија е **табела** која визуелно прикажува како перформира класификацискиот модел, организирајќи ги точните и погрешните предвидувања во однос на вистинските вредности.

**Што е ROC (Receiver Operating Characteristic) крива?**

**ROC кривата** е графички приказ кој го мери перформансот на класификациски модел во однос на различни **прагови** за одлука.

* Покажува компромис меѓу **осетливоста (sensitivity)** и **1 - специфичноста (false positive rate)**.

**Оските на ROC графикот**

1. **X-оска (False Positive Rate - FPR):**  
   Колкав процент од негативните случаи погрешно се класифицирани како позитивни.

FPR=FPFP+TN\text{FPR} = \frac{FP}{FP + TN}FPR=FP+TNFP​

1. **Y-оска (True Positive Rate - TPR или Sensitivity):**  
   Колкав процент од позитивните случаи се правилно класифицирани.

TPR=TPTP+FN\text{TPR} = \frac{TP}{TP + FN}TPR=TP+FNTP​

**Генерирање на ROC крива**

* За секоја можност, класификацискиот модел враќа **веројатност** за позитивна класа (P(Y=1∣X)P(Y = 1|X)P(Y=1∣X)).
* Со промена на прагот (threshold), можеш да го контролираш бројот на TPTPTP, FPFPFP, TNTNTN, и FNFNFN.
* На секој праг, се пресметуваат:
  + **TPR (осетливост)**
  + **FPR (лажна позитивна стапка)**

Со графички приказ на **TPR** против **FPR**, се добива ROC кривата.

**Клучни точки**

1. **Совршен модел:**
   * Достига TPR=1TPR = 1TPR=1 со FPR=0FPR = 0FPR=0, што значи дека правилно ги класифицира сите позитивни случаи без грешки.
   * Кривата е најблиску до горниот лев агол.
2. **Случаен модел:**
   * Достига права линија со наклон од 45° (дијагонала), што значи дека нема разлика меѓу позитивните и негативните предвидувања.
3. **Практичен модел:**
   * Кривата е над дијагоналата, што покажува дека моделот има способност да разликува меѓу класите.

**AUC (Area Under the Curve)**

* **AUC** го мери просторот под ROC кривата.
* Претставува целокупниот перформанс на моделот.
  + **AUC = 1:** Совршен модел.
  + **AUC = 0.5:** Нема дискриминациска моќ (случајно предвидување).
  + **AUC < 0.5:** Лош модел (предвидува спротивно).

**Што е Lift Plot?**

**Lift plot** е визуелизација која се користи за да се оцени ефективноста на класификациски модел во споредба со случајно предвидување (базирано на случајни прогнози).  
Тој покажува како моделот го подобрува процесот на предвидување во однос на "базичното" или случајно предвидување, наречено **baseline model**.

**Како работи Lift Plot?**

* **Lift** мерката покажува колку е подобро моделот отколку случајно предвидување, кога станува збор за класификација на позитивни случаи.
  + **Lift = 1**: Моделот нема подобрување од случајно предвидување (слично како дијагоналата на ROC).
  + **Lift > 1**: Моделот прави подобри предвидувања од случајно.
  + **Lift < 1**: Моделот прави лоши предвидувања од случајно.

**MACHINE LEARNING 2**

**Што е Полиномијална Регресија?**

**Полиномијална регресија** е метод за регресија кој се користи за моделирање на зависноста помеѓу зависната и независната променлива преку полиномски функции. Тоа е проширување на линeарната регресија, каде што моделот не е ограничен на права линија, туку може да биде кривулја која најдобро ја прилагодува зависноста.

**Полиномијална Регресија во Практика**

Во класичната **линеарна регресија**, моделот изгледа вака:

y=β0+β1x+ϵy = \beta\_0 + \beta\_1 x + \epsilon

**Полиномијалната регресија** користи полиномски функции на независната променлива xx, на пример:

y=β0+β1x+β2x2+β3x3+⋯+βnxn+ϵy = \beta\_0 + \beta\_1 x + \beta\_2 x^2 + \beta\_3 x^3 + \dots + \beta\_n x^n + \epsilon

Оваа форма на модел е способна да опишува податоци кои имаат нелинеарни однесувања.

**Како Работи Полиномијалната Регресија?**

1. **Трансформација на податоците:** Полиномијалната регресија започнува со трансформирање на независната променлива xx во поголеми степени (на пример, x2,x3,…x^2, x^3, \dots). Ова овозможува моделот да биде пофлексибилен и да го опише нелинеарното однесување на податоците.
2. **Фитирање на моделот:** Моделот се учи како и во обичната линeарна регресија, користејќи методи како **метод на најмали квадрати** (least squares). Овде, наместо само xx, ќе се вклучат и степени на xx.
3. **Прогноза:** Кога ќе се обучи моделот, за нови вредности на xx можеме да пресметаме yy, каде што yy ќе биде вредност добиена преку полиномијалната функција.

**Пример на Полиномијална Регресија:**

Ако имате податоци кои изгледаат како кривулја (на пример, растот на популацијата низ годините), може да користите полиномијална регресија да ја опишете зависноста, бидејќи едноставната линeарна регресија може да не ја опише добро.

Предпоставете дека податоците се како следниве:

* xx (независна променлива): Години
* yy (зависна променлива): Популација

Имајќи податоци од неколку години и нивната популација, ако се обидете со линeарен модел, можеби нема да добиете добра прогноза. Полиномијалната регресија ќе вклучи x2x^2 (и можеби повисоки степени на xx) за да создаде пофлексибилен модел.

**Предности на Полиномијалната Регресија**

* **Флексибилност**: Може да се користи за моделирање на сложени и нелинeарни односи.
* **Подобрување на точноста**: Може да даде подобри резултати за податоци кои не следат линеарна форма.

**Недостатоци на Полиномијалната Регресија**

* **Преголема флексибилност (overfitting)**: Кога се користат високи степени на полиномијал, моделот може да се прилагоди премногу на податоците и да не генерализира добро на нови податоци.
* **Тешкотија при интерпретација**: Како што се зголемува степенот на полиномијалната регресија, така и интерпретацијата на моделот станува потешка.

**Пример:**

Да речеме дека имате податоци за висина на дрвја во зависност од годините:

* xx = години
* yy = висина на дрво

Ако се обидете да ги прилагодите овие податоци со полиномијален модел, можеби ќе добиете нешто како ова:

y=2+3x−0.5x2y = 2 + 3x - 0.5x^2

Овој модел ќе изгледа како паребола и ќе се соочува со проблеми на променливиот раст на висината на дрвото со текот на времето.

**Заклучок**

Полиномијалната регресија е моќна техника за моделирање на податоци со нелинеарни трендови. Сепак, треба да се внимава да не се користи премногу високи степени на полином, бидејќи тоа може да доведе до прекумерно прилагодување на моделот (overfitting).

**Overfitting и како да се избегне**

**Overfitting** (прекумерно прилагодување) се јавува кога моделот премногу се прилагодува на специфичните податоци за тренинг, што доведува до ниска грешка на тренинг сетот, но лоша способност за предвидување на нови, невидени податоци. Ова може да се случи кога моделот е премногу сложен и вклучува многу карактеристики или високи степени на полиномијални функции, што резултира со "приспособување" на секој мал варијабилитет во податоците, вклучувајќи и случајни флуктуации кои не претставуваат вистински образец.

**Што е Overfitting?**

* **Цитат од Алберт Ајнштајн:** *"Сè треба да биде направено колку што е можно поедноставно, но не и поедноставено."*
* Overfitting се однесува на случај кога моделот е премногу сложен и се прилагодува на тренинг податоците на таков начин што резултира со многу мала грешка на тренинг сетот, но не може да се генерализира на нови податоци. Ова доведува до лоши перформанси на тест сетот или на нови податоци.

**Како да го избегнеме Overfitting?**

1. **Висока сложеност на моделот:**
   * Кога моделот има премногу параметри или кога се користат високи степени на полиномијални модели, може да се случи да се прилагоди премногу на податоците за тренинг. На пример, користењето на високи степени на полиномска регресија може да резултира во криви кои премногу се "навлекуваат" на тренинг податоците.
2. **Преоптоварување со карактеристики:**
   * Ако се користат премногу независни променливи (карактеристики), моделот може да "запамети" специфични детали кои не се репрезентативни за нови податоци, што доведува до пренагласено прилагодување.
3. **Голема димензионалност:**
   * Кога има многу карактеристики, и тие се високо поврзани, моделот може да стане "тешко да се прилагоди" и прекумерно да го запамети секој детаљ од податоците. Ова води до лоша способност на генерализација.
4. **Високи коефициенти:**
   * Ако моделот вклучува екстремни вредности на коефициентите, тогаш моделот може да се прилагоди на специфични податоци на тренингот, но да не биде корисен за нови податоци.

**Како да се избегне Overfitting?**

1. **Корисење на повеќе податоци:**
   * Поголем број на податоци ќе му овозможи на моделот да научи повеќе општи образци и да избегне да се прилагоди на случајни флуктуации во малите податоци.
2. **Редукција на сложеноста на моделот:**
   * Понекогаш е најдобро да се поедностави моделот, да се користат помалку независни променливи или да се намали степенот на полиномот.
3. **Крос валидација:**
   * Употребата на крос валидација помага да се процени како моделот ќе се изведува на нови податоци. Кога се тестира моделот на податоци кои не се користени за тренинг, тоа може да ја покаже неговата способност за генерализација.
4. **Применување на техники за редукција на димензионалноста:**
   * Алгоритми како што се Principal Component Analysis (PCA) или SelectKBest може да се користат за намалување на бројот на карактеристики, што може да помогне во спречување на прекумерно прилагодување.
5. **Регуларизација:**
   * Техниките за регуларизација, како **L1 (Lasso)** и **L2 (Ridge)** регуларизација, додаваат казни за големите вредности на коефициентите и помагаат да се задржи моделот поедноставен.

**Пример: Како изгледа Overfitting?**

* **Линеарен модел**: Ако податоците имаат линеарна зависност, тогаш линеарниот модел ќе се прилагоди и на тест сетот и на новите податоци.
* **Полиномијален модел со високи степени**: Ако користиме високи степени на полином, моделот може да "седи" премногу близу на тренинг податоците, создавајќи многу сложена кривула која нема да биде добра за нови податоци.

**Заклучок:**

Overfitting е голем проблем во машинското учење и статистиката, бидејќи го намалува капацитетот на моделот да генерализира и да прави точни предвидувања на нови податоци. Важно е да се избере соодветната сложеност на моделот, да се користат техники како крос валидација и редукција на димензионалност, и да се примени регуларизација за да се избегне прекумерното прилагодување.

**K-Fold Cross Validation (K-Fold CV)** е техника за проценка на перформансите на моделите во машинското учење, која се користи за да се добијат поопшти проценки за точноста на моделот и да се намали ризикот од прекумерно прилагодување (overfitting). Оваа техника ја дели целокупната обучувачка серија на k подмножества (folds) и секое од нив ја добива улогата на тест сет на различен начин.

**Како функционира K-Fold Cross Validation?**

1. **Разделување на податоците (Split the data):**
   * На почетокот, податоците се разделуваат на k подмножества (folds), каде што секое подмножество е еднакво големо. На пример, ако имате 1000 примероци на податоци и изберете k=5, тогаш ќе имате 5 подмножества, секое со 200 примероци.
2. **Тренинг и тестирање (Training and testing):**
   * За секое од k подмножества (folds), правите следно:
     + **Тренинг:** Обучувањето на моделот се врши со користење на останатите k-1 подмножества (folds). Пример: Ако се користат 5 подмножества, во првиот круг, моделот ќе се обучува на 4 подмножества, а едно ќе се користи како тест.
     + **Тестирање:** Моделот се тестира на едно подмножество, што значи дека ќе се користи само едно подмножество како тест сет. Тоа значи дека секое подмножество ќе има улога на тест сет за време на еден круг од крос валидирацијата.
3. **Правење предвидувања (Make Predictions):**
   * Моделот, обучен на k-1 подмножества, потоа прави предвидувања на подмножеството што се користи како тест. Тестот се прави со претставувањето на целта (target).
4. **Чување на предвидувањата (Store predictions):**
   * Предвидувањата направени за секое подмножество се чуваат во посебен масив, наречен "CV предвидувања" (CV predictions). Овие предвидувања ќе бидат користени за да се процени точноста на моделот.
5. **Поглед на точноста (Assess accuracy):**
   * По сите k кругови, се прави споредба на CV предвидувањата со вистинските вредности (target) за сите подмножества. Користејќи ги овие податоци, може да се пресмета точноста (accuracy) или други мерки како што се прецизноста (precision), сетилноста (sensitivity), и спецификата (specificity) на моделот.

**Предности на K-Fold Cross Validation:**

* **Поголема проценка на перформансите:** Намалува зависноста од случајниот избор на тренинг и тест сет.
* **Сите податоци се користат за тестирање:** Секое подмножество ќе се користи барем еднаш за тестирање, така што целокупниот податочен сет е искористен.
* **Покажува поголема стабилност на проценката:** Овој метод дава стабилни резултати дури и ако се променат некои податоци, затоа што резултатите се засновани на повеќе различни тренинг-сетови и тест-сетови.

**Недостатоци на K-Fold Cross Validation:**

* **Висока извршна цена:** Пошто моделот треба да се обучува k пати, ова може да биде скапо за големи податоци.
* **Може да биде чувствителен на k:** Изборот на оптимален број на k може да влијае на резултатите. Мал број на k може да доведе до погрешни проценки, додека многу голем број на k може да направи процесот побавен.

**Пример:**

1. Ако имате 10 податоци и изберете k=5, ќе имате 5 подмножества од 2 податоци секое.
2. Во првиот круг, ќе се обучува моделот на 8 податоци (освен првите два), а тестот ќе се изврши на првите два.
3. Во вториот круг, ќе се обучува на 8 податоци (освен вторите два), а тестот ќе се изврши на вторите два.
4. Овој процес ќе продолжи сè додека сите подмножества не бидат искористени како тест податоци.

**Заклучок:**

K-Fold Cross Validation е корисна техника која помага да се проценат перформансите на моделот на стабилен и објективен начин, обезбедувајќи намалена зависност од случајниот избор на тренинг и тест сетови.

**Bias (Извршна грешка)** и **Variance (Различност)** се два важни концепти во машинското учење и статистиката, кои се користат за да се опишат два типа на грешки што може да влијаат на перформансите на моделот. Разбирањето на овие два термина е клучно за создавање модели кои добро се генерализираат на нови, непознати податоци.

**Bias (Извршна грешка)**

* **Што е Bias?**
  + Bias претставува разликата помеѓу просечната прогноза на моделот и вистинската вредност која сакаме да ја предвидиме.
  + Висок bias значи дека моделот има **погрешни претпоставки** за природата на податоците и може да направи **едноставни погрешни проценки**. Моделот не успева да ја улови сложеноста на податоците.
  + Во случај на висок bias, моделот е **премногу едноставен** и не е способен да ги улови важните одлики на податоците.
* **Пример:**
  + Ако се обидуваме да користиме линеарен модел за проблем кој има нелинеарна природа, тогаш нашиот модел ќе има висок bias бидејќи не може да ги улови сите важни детали во податоците.
* **Ефект на висок Bias:**
  + **Под-прилагодување (Underfitting):** Моделот не успева да ја долови целосната сложеност на податоците и го прави лошо претставување на тренинг податоците и новите податоци.

**Variance (Различност)**

* **Што е Variance?**
  + Variance се однесува на променливоста на моделот кога се користат различни тренинг сетови.
  + Модел со висок variance има способност да се **приспособува премногу** на малите флуктуации во тренинг податоците (шум), што значи дека ќе направи многу добри прогнози на тренинг сетот, но **лоши на нови податоци**.
  + Високиот variance значи дека моделот е **премногу сложен** и многу чувствителен на тренинг податоците.
* **Пример:**
  + Кога се користи сложен модел, како што е полином со висока степен, тој може да се прилагоди на секој мал деталь во тренинг сетот. Иако тоа може да доведе до мала грешка на тренинг податоците, нови податоци ќе бидат многу погрешно предвидени.
* **Ефект на висок Variance:**
  + **Прекумерно прилагодување (Overfitting):** Моделот се прилагодува на специфичните карактеристики на тренинг податоците и може да не се генерализира добро за нови, непознати податоци.

**Bias-Variance Tradeoff (Трговија помеѓу Bias и Variance)**

* **Како работат Bias и Variance заедно?**
  + **Кога се зголемува Bias, варијансата се намалува**, и обратно.
  + **Модел со висок Bias и низок Variance** ќе биде едноставен и стабилен, но нема да може да го улови сложеноста на податоците, што ќе доведе до под-прилагодување.
  + **Модел со низок Bias и висок Variance** ќе биде многу сложен и ќе го улови секој шум во податоците, што ќе доведе до прекумерно прилагодување.
* **Оптимален модел:**
  + Целта е да се најде баланс помеѓу Bias и Variance, каде што моделот ќе има доволно сложеност за да ги улови важните аспекти на податоците, но не премногу сложен за да не се прилагоди на шумот.
  + **Оптимален Bias/Variance Tradeoff** значи дека моделот е способен да ги предвидува новите податоци со добра точност без да се прилагодува премногу на тренинг сетот.

**Пример со графици:**

1. **Low Bias, High Variance (Прекумерно прилагодување / Overfitting):**
   * Моделот добро ги предвидува податоците во тренинг сетот, но лошо ги предвидува новите податоци.
   * Изгледа како да се прилагодува на секоја флуктуација во податоците.
2. **High Bias, Low Variance (Под-прилагодување / Underfitting):**
   * Моделот е премногу едноставен и не успева да ги улови важните детали во податоците.
   * Погрешен е и за тренинг и за нови податоци, но стабилен бидејќи не се прилагодува премногу.
3. **Оптимален модел:**
   * Моделот има добро предвидување на нови податоци, со балансиран Bias и Variance.

**Заклучок:**

* **Bias** и **Variance** се два клучни аспекти кои треба да се балансат кога се гради модел. За оптимален модел, треба да се избегнуваат екстреми од двата краја, т.е., треба да се избалансираат понискиот bias и разумниот variance.

**Regularization: LASSO and Ridge**

**Regularization** е техника која се користи за да се спречи прекумерно прилагодување (overfitting) на моделите, особено кога имаме големи количини на податоци и многу карактеристики (features). Таа вклучува додавање на **регуларизациски термини** во функцијата на губење (loss function), кои ги казнуваат специфичните својства на параметрите на моделот. Целта е да се создаде модел што не само што ги минимизира грешките на тренинг сетот, туку и има **добра генерализација** на нови податоци.

**Основен пристап**

* Вложување на редуцирање на вредностите на параметрите, така што тие да не бидат **премногу големи** или **екстремни**.
* Со тоа, се подобрува способноста на моделот за **генерализација** и се избегнува прекумерно прилагодување на шумот во податоците.

**Облик на модифицираната функција на губење (Loss function):**

Lreg()=L()+R()L\_{reg}() = L() + R()

* L()L() е оригиналната функција на губење, а R()R() е **регуларизацискиот термин** кој ја казнува сложеноста на моделот.
* **Тежината** на регуларизацијата е контролирана од параметарот λ\lambda, што го одредува колку ќе биде важно да се казнуваат големите вредности на параметрите.

**LASSO Регресија (Least Absolute Shrinkage and Selection Operator)**

**Функција на губење за LASSO:**

LLASSO()=1n∑i=1n(yi−xi⊤β)2+λ∑j=1J∣βj∣L\_{LASSO}() = \frac{1}{n} \sum\_{i=1}^{n} \left(y\_i - x\_i^\top \beta \right)^2 + \lambda \sum\_{j=1}^J |\beta\_j|

* Првиот дел е класичната **МSE (Mean Squared Error)**, која го минимизира растојанието помеѓу предвидувањата и вистинските вредности.
* Вториот дел е **регуларизацискиот термин** кој се состои од **l1 нормата** (сума на апсолутните вредности на параметрите βj\beta\_j), што ги казнува големите вредности на параметрите.
* **Регуларизацискиот параметар λ\lambda** контролира колку ќе се казнуваат големите параметри:
  + **Голем λ** ќе резултира во повеќе казнување и може да доведе до повеќе нули во параметрите (селекција на карактеристики).
  + **Мал λ** ќе дозволи параметрите да бидат поголеми, но може да се зголеми ризикот од прекумерно прилагодување.

**Повеќе за LASSO:**

* **LASSO и селекција на карактеристики:** Поради природата на l1 нормата, LASSO може да ги елиминира некои параметри сосема (поставувајќи ги на нула), што го прави многу корисен за **селекција на карактеристики** во случаи кога имате голем број на независни променливи.
* **Процесот на оптимизација:** Целта е да се минимизира **LASSO функцијата на губење**, што резултира во оптимални вредности за параметрите βLASSO\beta\_{LASSO}.

**Ridge Регресија**

Ridge регресија, исто така, е техника на регуларизација, но користи **l2 нормата** за казнување на параметрите. За разлика од LASSO, Ridge нема да ги постави параметрите на нула, туку ќе ги **намали** сите параметри, но ќе ги задржи сите.

**Функција на губење за Ridge:**

LRidge()=1n∑i=1n(yi−xi⊤β)2+λ∑j=1Jβj2L\_{Ridge}() = \frac{1}{n} \sum\_{i=1}^{n} \left(y\_i - x\_i^\top \beta \right)^2 + \lambda \sum\_{j=1}^J \beta\_j^2

* Првиот дел е повторно **MSE**, а вториот дел е **l2 нормата** (сума на квадратите на параметрите), кој ја казнува големината на параметрите.

**Основна разлика помеѓу LASSO и Ridge:**

* **LASSO** користи **l1 нормата**, што може да ги постави параметрите на нула и да направи селекција на карактеристики.
* **Ridge** користи **l2 нормата**, што ги намалува параметрите, но не ги поставува на нула, па сите карактеристики остануваат вклучени.

**Применета цел:**

* Кога имате **многу карактеристики**, LASSO може да биде корисен за **селектирање на важните карактеристики** и елиминирање на ненадолжните.
* Ако сакате да **задржите сите карактеристики**, но да ги намалите нивните вредности, тогаш **Ridge** ќе биде подобар избор.

**Заклучок:**

* LASSO и Ridge се две различни техники на регуларизација, секоја со свои предности. LASSO е поодговарачки кога сакате да направите **селекција на карактеристики**, додека Ridge е корисен кога сакате да го **контролирате големината на параметрите** без да ги елиминирате.

Изборот на параметарот **λ (лаmbda)** за регуларизација е клучен за оптимизација на моделот и контрола на прекумерното прилагодување (overfitting) или недоволното прилагодување (underfitting). Параметарот λ ја контролира тежината на регуларизацијата, односно како многу ќе се казнуваат големите вредности на параметрите. Повеќе λ значи поголема казна за параметрите и обично помал број на значајни карактеристики, додека помал λ значи помала казна и поголема флексибилност за моделот.

**Како да го изберете λ:**

1. **Крос-валидација (Cross-Validation):** Најчесто користената техника за избор на λ е **крос-валидација**, најчесто со **k-fold крос-валидација**.
   * **Поделете го податочниот сет** на k подмножества (folds).
   * За секој избор на λ, **тренинг на моделот** се изведува на сите подмножества освен едно (тест подмножество), и се пресметува грешката на тест подмнозеството.
   * **Повторете ја оваа постапка** за сите вредности на λ.
   * Изберете ја вредноста на λ која го минимизира **просечниот тестен error** преку сите крос-валидациони фази.
2. **Графички пристап:** Можете да ја претставите грешката на тренинг и тест сетот како функција на λ и да ја изберете оптималната вредност на λ што минимизира грешката на тест сетот, додека избегнувате премногу висока грешка на тренинг сетот. Ако λ е премногу големо, моделот ќе има **мала флексибилност** и ќе се користат само неколку карактеристики, а ако е премалку, моделот ќе има **прекумерно прилагодување** на податоците (overfitting).
3. **Справување со предвремен избор:**
   * **Ниски вредности на λ**: Моделот е помалку редуциран и може да страда од **overfitting**, со што ќе има висока варијанса.
   * **Високи вредности на λ**: Моделот ќе има **помала варијанса**, но ќе страда од **underfitting**, што значи дека нема да биде доволно флексибилен за да ги улови важните карактеристики на податоците.

**Пример за избор на λ преку крос-валидација:**

1. Поделете ги податоците на k подмножества.
2. За секој λ, направете тренинг на моделот со k-1 подмножества, а тестирајте го на преостанатото подмножество.
3. Повторете го ова за секој λ и пресметајте го просечниот тестен error.
4. Изберете ја вредноста на λ која минимизира овој error.

**Експериментирање со различни вредности на λ:**

* За **мала λ**, моделот ќе има многу параметри кои не се казнувани, што може да доведе до **overfitting**.
* За **голема λ**, моделот ќе има помалку параметри и ќе биде **поједноставен**, но може да доведе до **underfitting**, што значи дека моделот не е доволно сложен за да ја улови комплексноста на податоците.

**Заклучок:**

Изборот на λ треба да се направи внимателно, користејќи крос-валидација или графички пристап за да се најде оптимален баланс помеѓу **bias** и **variance**. Добриот избор на λ ќе доведе до **поголема способност за генерализација на моделот** на нови податоци.

**Геометрија на податоци и одлучувачки дрвја**

**Геометрија на податоци во класификација:**

Геометријата на податоците се однесува на структурата и разделливоста на податоците во просторот на карактеристиките. Кога се користи **логистичка регресија** за класификација, методата работи најдобро кога:

* Класите се добро разделени во просторот на карактеристиките.
* Граничната линија помеѓу класите може лесно да се претстави со едноставни геометриски форми (на пример, линеарни граници).

Меѓутоа, во многу реални податоци, иако класите можеби се сепарирани, границите на одлука не се секогаш лесно претставени со едноставни линеарни еднаквости.

**Примери на граници на одлука:**

* **Линеарни граници**: За добро разделени класи во едноставни простори на карактеристики, логистичката регресија може да најде линеарни граници кои ефективно ги разделуваат класите.
* **Нелинеарни граници**: Кога податоците не се линеарно разделени, логистичката регресија може да има проблеми, па може да се користат посложени модели (како одлучувачки дрвја или SVM).

**Интерпретирани модели:**

Во машинското учење, моделите кои се **интерпретирани** се често претпочитани, особено кога е важно да се разбере како моделот доаѓа до своите предвидувања. Еден таков модел е **одлучувачко дрво**.

**Одлучувачки дрвја:**

**Одлучувачко дрво** е структура слична на тековната шема, која се користи за класификација или регресија. Структурата на дрвото се состои од:

1. **Внатрешни јазли** кои ги тестираат вредностите на атрибутите (карактеристики).
2. **Рамнишни гранки** кои водат до понатамошни тестови базирани на резултатите од претходните проверки.
3. **Листови** кои претставуваат одлуки за класификација.

Одлучувачките дрвја се многу интерпретирани бидејќи секој тест е лесно разбирлив и може да се претстави како упатство за донесување одлука.

**Одлучувачки дрвја (Decision Trees)**

Одлучувачките дрвја се едни од најпопуларните и најразбирливите модели за класификација и регресија во машинското учење. Тие ги делат податоците на различни сегменти преку серија на прашања (т.е. тестови) на различни атрибути, што создава структура која изгледа како дрво. За секој внатрешен јазол во дрвото, има прашање кое дели податоците во две или повеќе групи, додека на крајот на дрвото се наоѓаат **листи** кои го претставуваат финалниот резултат (класа или вредност).

**Структура на одлучувачко дрво**

1. **Корен (Root Node)**: Почетниот јазол во дрвото кој претставува целиот сет на податоци. Овој јазол го поставува првиот прашање за поделба на податоците.
2. **Внатрешни јазли (Internal Nodes)**: Секој внатрешен јазол претставува тест на еден атрибут (карактеристика). Во овој јазол, податоците се делат на две или повеќе подгрупи врз основа на резултатите од тестот.
3. **Листови (Leaf Nodes)**: На крајот на дрвото се листовите, кои претставуваат конечни класификација или предвидување на вредноста (за регресија).
4. **Гранки (Branches)**: Гранките поврзуваат јазли и покажуваат како податоците се делат според резултатот од тестовите.

**Процес на учење на одлучувачко дрво**

Процесот на изградба на одлучувачко дрво вклучува следниве чекори:

1. **Поделба на податоците**: На почетокот, сите податоци се во еден јазол (корен). За да се изгради дрвото, треба да се изберат атрибути и прашања кои ќе ја поделат податочната сетка на групи.
2. **Избор на најдобар атрибут**: За секој внатрешен јазол, треба да се избере атрибут врз основа на кој ќе се направи поделба на податоците. Овој избор најчесто се прави со примена на **критериуми за избор** како што се:
   * **Gini Index** (за класификација)
   * **Entropy** или **Information Gain** (за класификација)
   * **Mean Squared Error (MSE)** (за регресија)
3. **Рекурзивна поделба**: Постапката за избор на атрибути и поделба се повторува рекурзивно за секој нов јазол. Процесот завршува кога:
   * Податоците во јазолот се од истата класа (за класификација).
   * Јазолот има премалку податоци за да продолжи поделбата.
   * Нема други атрибути за тестирање.

**Пример: Изградба на одлучувачко дрво**

Представете си дека имате податоци за тоа дали луѓето чекаат за маса во ресторан, и сакате да изградите одлучувачко дрво за да ги класифицирате како "Чекаат" или "Не чекаат". Атрибутите може да вклучуваат:

* **Има ли алтернатива (Alternate)** - Да/Не
* **Тип на ресторан (Type)** - Италијански, Француски, и др.
* **Погодување на број на гости (Patrons)** - Нема, Неколку, Полн
* **Ден од неделата (Fri/Sat)** - Петок или Сабота

Пример на дрво за одлука:

* **Корен**: "Ден од неделата"
  + Ако е **Петок или Сабота**:
    - Ако има **алтернатива**: "Не чекаат".
    - Ако **нема алтернатива**: "Чекаат".
  + Ако **не е Петок или Сабота**:
    - Ако има **погодување на број на гости**: "Не чекаат".
    - Ако **нема погодување на број на гости**: "Чекаат".

**Алгоритми за изградба на одлучувачки дрвја**

* **ID3 (Iterative Dichotomiser 3)**: Овој алгоритам користи **Information Gain** како критериум за избор на најдобар атрибут.
* **C4.5**: Подобрената верзија на ID3, која користи **Gain Ratio** за избор на атрибут.
* **CART (Classification and Regression Trees)**: Овој алгоритам користи **Gini Index** за класификација и **MSE** за регресија. Изградува бинарни дрвја (секој внатрешен јазол има две гранки).

**Примери на предности и недостатоци**

**Предности:**

* **Интерпретабилност**: Одлучувачките дрвја се лесни за разбирање и визуелизација, што ги прави одлични за задачи каде објаснувањето на моделот е важно.
* **Не бараат нормализација**: Атрибутите не треба да бидат нормализирани или стандартизирани, бидејќи одлучувачките дрвја работат со релативните вредности на атрибутите.

**Недостатоци:**

* **Прекумерно приспособување (Overfitting)**: Одлучувачките дрвја често можат да се прекумерно приспособат на тренинг податоците, особено ако се дозволат многу длабоки дрвја.
* **Несигурност**: Одлучувачките дрвја можат да бидат нестабилни, бидејќи мали промени во податоците може да доведат до значајни промени во структурата на дрвото.

**Како да се реши прекумерното приспособување?**

* **Прекратување (Pruning)**: Прекратувањето на дрвото се врши за да се избегне прекумерно приспособување. Ова може да се направи преку елиминирање на некои гранки кои не придонесуваат значително за точноста на моделот.
* **Минимален број на примероци за поделба**: Поставување минимален број на примероци кои треба да бидат присутни во еден јазол за да може да се изврши поделба.
* **Максимална длабочина на дрвото**: Ограничување на длабочината на дрвото може да помогне да се спречи создавање на многу комплексни и нестабилни модели.

Со оваа структура и процес, одлучувачките дрвја се многу моќен алат во машинското учење и се користат за различни задачи како што се класификација, регресија и сегментација на податоци.

**Критериуми за Сплитување во Одлуки Дрвја**

При креирање на одлуки дрвја, целта е да се изберат најдобрите атрибути и прагови за сплитување на податоците. **Критериумите за сплитување** го водат процесот на мерење на квалитетот на сплитот. Процесот на сплитување на податоците може да се оцени со различни метрики кои ја квантитизираат како добро се разделуваат класите или како се намалува неизвесноста.

**Оптималност на Сплитувањето**

Не постои универзален "правилен" начин да се дефинира оптимален сплит, но постојат некои општи насоки кои се следат:

1. **Прогресивно Повиќе Чисти Региони**: Секој регион креиран со сплит треба да содржи доминантно една класа како што расте дрвото. Ова води до повеќе специјализирани региони и подобра разделба помеѓу класите.
2. **Диференциабилна Метричка за Фитнес**: Критериумите за сплитување треба да бидат диференциабилни за да дозволат оптимизација (т.е. треба да биде можно да се најде најдобриот праг за сплитување преку оптимизациски техники).
3. **Без Празни Региони**: Сплитот не треба да резултира со региони кои немаат тренинг податоци, бидејќи ова ќе доведе до губење на информации и неважечки прогнози.

**Класификациска Грешка**

**Класификациската грешка** го мера квалитетот на поделбата во дадената подгрупа податоци. За сплит кој ги дели податоците во два региона, класификациската грешка за секој регион се пресметува како:

* Да претпоставиме дека пропорцијата на тренинг точки во регион RR кои се етикетирани со класа kk е p(k∣R)p(k|R).
* Грешката за еден регион RR се пресметува како:

Error(i∣j,tj)=1−max⁡kp(k∣R)\text{Error}(i|j, t\_j) = 1 - \max\_k p(k|R)

Оваа формула го пресметува максималниот процент на класа која ќе биде доделена и грешката е комплемент на оваа вредност.

За да го најдеме оптималниот сплит, го избираме атрибутот и прагот tjt\_j кој ја минимизира **просечната класификациска грешка** преку двата создадени региона:

minj,tj(N1N⋅Error(1∣j,tj)+N2N⋅Error(2∣j,tj))\text{min}\_{j,t\_j} \left( \frac{N\_1}{N} \cdot \text{Error}(1|j, t\_j) + \frac{N\_2}{N} \cdot \text{Error}(2|j, t\_j) \right)

Каде N1N\_1 и N2N\_2 се бројот на тренинг точки во региони R1R\_1 и R2R\_2, а NN е вкупниот број на тренинг точки.

**Гини Индекс**

**Гини индексот** е друга честопати користена метрика за мерење на квалитетот на сплитот. Гини индексот квантитизира **нечистота** на еден регион: што помал е Гини индексот, толку помалку е нечист регионот.

* Гини индексот за регион RR се пресметува како:

Gini(i∣j,tj)=1−∑kp(k∣R)2Gini(i|j, t\_j) = 1 - \sum\_k p(k|R)^2

каде p(k∣R)p(k|R) е пропорцијата на тренинг точки во регион RR кои припаѓаат на класа kk.

За да го најдеме оптималниот сплит, го избираме атрибутот и прагот tjt\_j кој ја минимизира **просечната Гини индекс** преку двата региона, тежено според популацијата на регионот:

minj,tj(N1N⋅Gini(1∣j,tj)+N2N⋅Gini(2∣j,tj))\text{min}\_{j,t\_j} \left( \frac{N\_1}{N} \cdot Gini(1|j, t\_j) + \frac{N\_2}{N} \cdot Gini(2|j, t\_j) \right)

**Пример**

За сплит со два региона R1R\_1 и R2R\_2, Гини индексот за секој регион се пресметува како:

* Gini(R1)=(662)2+(062)2=0Gini(R\_1) = \left( \frac{6}{62} \right)^2 + \left( \frac{0}{62} \right)^2 = 0 (чист)
* Gini(R2)=(513)2+(813)2=0.4769Gini(R\_2) = \left( \frac{5}{13} \right)^2 + \left( \frac{8}{13} \right)^2 = 0.4769 (нечист)

Целта е да се минимизира тежената просечна вредност на овие Гини индекси.

**Теорија на Информација (Ентропија)**

Ентропијата е друга метрика за мерење на нечистотата на сплитовите. Таа доаѓа од теоријата на информација и ја квантитизира **неизвесноста** или **нечистотата** на распределбата на класите во еден регион.

* **Ентропијата** на една распределба на класите се дефинира како:

H(X)=−∑x∈Xp(x)log⁡2p(x)H(X) = - \sum\_{x \in X} p(x) \log\_2 p(x)

каде p(x)p(x) е веројатноста за класата xx во регионот XX.

Целта е да се минимизира **просечната ентропија** преку двата региона креирани со сплитот, што дава најинформативен сплит:

minj,tj(N1N⋅Entropy(1∣j,tj)+N2N⋅Entropy(2∣j,tj))\text{min}\_{j,t\_j} \left( \frac{N\_1}{N} \cdot \text{Entropy}(1|j, t\_j) + \frac{N\_2}{N} \cdot \text{Entropy}(2|j, t\_j) \right)

**Пример**

За два региона:

* R1:R\_1: Класа 1 има 0, Класа 2 има 6
* R2:R\_2: Класа 1 има 5, Класа 2 има 8

Ентропијата за секој регион ќе биде:

* H(R1)=0H(R\_1) = 0 (чист регион)
* H(R2)≈1.38H(R\_2) \approx 1.38 (помалку чист)

Целта е да се минимизира тежената сума на овие ентропии.

**Споредба на Критериуми**

Секој критериум за сплитување (Класификациска грешка, Гини индекс и Ентропија) има свои предности и слабости:

* **Класификациската грешка**: Едноставна, но не е толку чувствителна на сплитови во региони со силно нерамнотежни класи.
* **Гини индексот**: Повеќе чувствителен на нечистотата на регионите и добро функционира за бинарни сплитови.
* **Ентропијата**: Силно казнува нечистотија, но понекогаш може да резултира во подлабоки дрвја поради нејзината чувствителност.

**Фактори кои влијаат на изборот на критериумите**

* **Големина на дрвото**: Критериумот за сплитување влијае на комплексноста на добиеното одлука дрво. На пример, ентропијата обично резултира во подлабоки дрвја.
* **Перформанси**: Иако послабите региони се пожелни, премногу чисти лисја може да доведат до презаситеност, па е потребно да се постигне баланс.

**Услови за Запирање на Одлуки Дрвја**

За да се спречи презаситеност и да се задржи дрвото под контролирана големина, потребно е да се дефинираат услови за запирање. Овие може да вклучуваат:

* Минимален број на податоци во регион пред да се изврши сплит.
* Максимална длабочина на дрвото.

Минимално намалување на метрика по сплитот.

Ова се основни насоки за избор на критериум за сплитување во одлуки дрвја.

Прераскажување на твојот текст на македонски:

**Состојби на запирање и обликување на дрвото**

* Ако не ја запреме процедурите на учење на одлучувачкото дрво рачно, дрвото ќе продолжи да расте сè додека секоја област дефинирана од моделот не содржи точно еден примерок за обука, што ќе резултира со 100% точност на обуката.
* За да го спречиме ова, можеме едноставно да го запреме алгоритмот на одредена длабочина.
* Но како да ја определиме соодветната длабочина?

**Предности и Недостатоци**

* Ако имаме дрво со мала длабочина (например длабочина 4), можеби тоа нема да биде добро прилагодено за податоците за обука и нема да успее да ги улови нелинеарните граници меѓу класите.
* Дрво со висока длабочина ќе може да го класифицира секој примерок правилно, но истовремено има висок ризик од прекумерно прилагодување (overfitting), бидејќи ќе реагира премногу на ситни варијации во податоците.

**Услови за запирање**

* Најчесто користени услови за запирање вклучуваат:
  + Немој да правиш поделба ако сите примероци во областа припаѓаат на истата класа.
  + Немој да правиш поделба ако бројот на примероци во под-областите е помал од претходно дефинираниот праг (min\_samples\_leaf).
  + Немој да правиш поделба ако вкупниот број на лисја во дрвото ќе ја надмине претходно дефинирана граница.

**Претходно дефинирани прагови**

* Прогнозирањето на овие прагови може да се направи со користење на податоци кои не се користени во обуката (held-out dataset), или со користење на крос-валидација.

**Алтернатива на користење на услови за запирање**

* Голем проблем со претходно дефинираните услови за запирање е што можеме да запреме премногу рано или премногу доцна.
* Можеме да решиме ова преку избор на различни критериуми за запирање и да ги провериме преку крос-валидација.

**Мотивација за обликување на дрвото**

* Наместо да го спречиме растот на сложениот модел, можеме да ја примениме методата за обликување (pruning), која овозможува да го поедноставиме моделот и да го подобриме неговото општо перформанси.

**Обликување преку комплетност на трошоците**

* Еден од начините за обликување на дрвото е преку методата на комплетност на трошоците (cost complexity pruning), која бара да се изберат помали поддрвја со цел да се оптимизира балансот помеѓу перформансите и ефикасноста.

Ако има нешто конкретно што не е јасно или треба да го развиеме понатаму, слободно кажи!

**Bagging (Bootstrap Aggregating)** е техника за ансамблно учење која се користи за подобрување на перформансите на машинските модели, особено на одлучувачките дрвја. Таа работи така што ги комбинира предвидувањата од повеќе модели (обично ист тип) за да донесе финално предвидување. Еве како функционира bagging:

1. **Bootstrap Sampling**: Се креираат повеќе подмножества на податоците за учење со случајно земање примероци со замена. Тоа значи дека некои податоци ќе се појават повеќе пати, додека други можеби ќе бидат изоставени.
2. **Обучување на повеќе модели**: Секој модел (обично ист тип на модел) се тренира на различен случаен примерок од податоците. На пример, ако се користат одлучувачки дрвја, секое дрво се тренира на различен случаен примерок од податоците.
3. **Агрегација**: Кога сите модели ќе бидат обучени, нивните предвидувања се агрегираат. За задачи со регресија, финалното предвидување е обично **просек** на предвидувањата на сите модели. За задачи со класификација, вообичаен пристап е **мајоритарно гласање**, каде што класата која најмногу модели ја предвидуваат се избира како финален резултат.

**Клучни предности на Bagging: (Bootstrap + Aggregate)**

* **Намалување на варијансата**: Со просекување на предвидувањата на повеќе модели, bagging ја намалува варијансата на моделот, правејќи го постабилен и помалку подложен на прекумерно прилагодување (overfitting).
* **Подобрување на точноста**: Во многу случаи, комбинирањето на неколку модели на овој начин води до подобри перформанси отколку користењето на еден модел.

**Пример:**

* **Random Forest** е популарен алгоритам кој користи bagging со одлучувачки дрвја како основни модели.

**Random Forest** е алгоритам за ансамблно учење кој комбинира повеќе одлучувачки дрвја за да донесе подобри и посигурни предвидувања. Тој се заснова на техниката **bagging (Bootstrap Aggregating)** и користи дополнителни рандомизации за да ги подобри перформансите и да ја намали корелацијата помеѓу моделите.

**Детално објаснување:**

**1. Што е Random Forest?**

Random Forest е збирка од повеќе **одлучувачки дрвја**. Тој ги користи предвидувањата на секое дрво и ги комбинира за да донесе финално предвидување. Основната идеја е дека група од „слаби предвидувачи“ (индивидуални дрвја) може да се комбинира во „силен предвидувач“ (целата шума).

**2. Како функционира Random Forest?**

Еве ги чекорите што ги следи алгоритмот:

1. **Креирање на подмножества од податоците:**
   * Од оригиналниот сет на податоци, Random Forest креира повеќе подмножества користејќи **bootstrap sampling** (случајно земање примероци со замена).
   * Ова значи дека секое подмножество може да содржи дупликат записи, додека некои записи од оригиналниот сет може да бидат изоставени.
2. **Градење на одлучувачки дрвја:**
   * За секое подмножество, се гради едно одлучувачко дрво.
   * За време на градењето на дрвото, Random Forest додава уште една рандомизација: при секое гранување на дрвото, се зема случаен подмножество од карактеристиките (features), а од нив се избира најдобрата карактеристика за поделба.
   * Ова помага да се намали корелацијата меѓу дрвјата.
3. **Комбинирање на предвидувањата:**
   * Откако сите дрвја ќе бидат изградени, нивните предвидувања се комбинираат:
     + За **класификација**, се користи **мајоритарно гласање** – класата што ја предвидуваат најмногу дрвја е финалниот резултат.
     + За **регресија**, се зема **просекот** на предвидувањата од сите дрвја.

**3. Зошто Random Forest е моќен?**

* **Намалување на пренагласеното прилагодување (overfitting):** Одлучувачките дрвја сами по себе се подложни на overfitting, но со комбинирање на повеќе дрвја и користење на рандомизација, Random Forest го намалува овој ризик.
* **Намалување на варијансата:** Просекувањето (или гласањето) на предвидувањата од повеќе дрвја ја прави целата шума помалку чувствителна на шумот во податоците.
* **Раководење со повеќе карактеристики:** Random Forest добро работи и со голем број на карактеристики (features), бидејќи користи само случајни подмножества од нив при градењето на секое дрво.
* **Работи и со категоријални и континуирани податоци:** Без потреба од посебно предобработување на податоците.

**4. Клучни хиперпараметри:**

При користење на Random Forest, некои од најважните параметри кои можат да се прилагодат се:

* **n\_estimators**: Број на дрвја во шумата (поголем број дрвја често води до подобри резултати, но го зголемува времето за извршување).
* **max\_depth**: Максималната длабочина на секое дрво (за да се спречи overfitting).
* **min\_samples\_split**: Минимален број на податоци потребни за поделба на гранките.
* **max\_features**: Број на карактеристики кои ќе бидат земени предвид при секоја поделба.
* **random\_state**: Семе за рандомизацијата за репродуктивност.

**5. Пример:**

Замислете дека сакате да предвидите дали некој клиент ќе купи производ врз основа на податоци како возраст, приходи и локација. Random Forest ќе направи следново:

* Ќе создаде повеќе „шуми“ од подмножества на податоците.
* Секое дрво ќе донесе предвидување дали клиентот ќе купи.
* Финалното предвидување ќе биде класата што ја избрале најмногу дрвја (на пример, „да“ или „не“).

**6. Предности и слабости на Random Forest:**

**Предности:**

* Висока точност.
* Робустен на шум во податоците.
* Добро раководење со голем број карактеристики.

**Слабости:**

* Може да биде бавен за големи датасети со многу дрвја.
* Потешко за интерпретација отколку едноставно одлучувачко дрво.

**Boosting** е техника за ансамблно учење која комбинира повеќе „слаби модели“ (најчесто одлучувачки дрвја) за да создаде „силен модел“ со подобра точност. За разлика од **bagging** (како кај Random Forest), каде моделите работат независно, кај boosting моделите се градат последователно, и секој нов модел се обидува да ги коригира грешките на претходните.

**Како функционира Boosting?**

1. **Создавање на првичен модел:**
   * Алгоритмот започнува со создавање на прв „слаб модел“ (често едноставно одлучувачко дрво со мала длабочина).
2. **Фокусирање на грешките:**
   * После тренинг на првиот модел, се пресметува грешката за секој пример во датасетот.
   * Boosting ги дава поголемите тежини на примерите кои биле погрешно предвидени, со цел следниот модел да се фокусира на нив.
3. **Градење на нови модели:**
   * Со додавање на секој нов модел, тој се обучува да ги коригира грешките на претходниот.
   * На крај, сите модели се комбинираат (најчесто преку тежински просек или збир на предвидувањата).
4. **Финално предвидување:**
   * Се добива комбинирано предвидување каде секој модел придонесува според својата точност.

**Примери на Boosting алгоритми:**

1. **AdaBoost (Adaptive Boosting):**
   * Овој алгоритам придава тежини на примерите базирано на тоа дали биле правилно или погрешно класифицирани.
   * Моделите кои даваат подобри резултати добиваат поголема тежина во финалната одлука.
2. **Gradient Boosting:**
   * Користи техника на оптимизација наречена **градиентен спуст** за да го минимизира исходот на загубната функција.
   * Секое следно дрво предвидува разлики (резидуали) од претходното дрво.
3. **XGBoost (Extreme Gradient Boosting):**
   * Подобрен и побрз алгоритам базиран на Gradient Boosting, кој е многу популарен поради неговата брзина и точност.
   * Поддржува паралелно процесирање и користи напредни техники за спречување на пренагласување (overfitting).
4. **LightGBM (Light Gradient Boosting Machine):**
   * Оптимизиран за брзо процесирање и користи „Leaf-wise“ поделби наместо „Level-wise“, што го прави побрз за големи податочни сетови.
5. **CatBoost:**
   * Дизајниран за работа со категоријални податоци, елиминирајќи ја потребата за посебно енкодирање на категоријалните променливи.

**Предности на Boosting:**

* **Висока точност:** Boosting често дава подобри резултати од bagging, особено за сложени проблеми.
* **Прилагодување:** Секој нов модел активно ги коригира грешките на претходните.
* **Флексибилност:** Може да се користи и за класификација и за регресија.

**Слабости на Boosting:**

* **Чувствителност на шум:** Boosting може да ги пренагласи примерите со шум во податоците, што може да доведе до пренагласување (overfitting).
* **Побрз процесор:** Алгоритмот е посложен и често побавен во споредба со Random Forest.

**Пример (Gradient Boosting во Python):**

from sklearn.ensemble import GradientBoostingClassifier

# Примерни податоци

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Тренирање на Gradient Boosting модел

model = GradientBoostingClassifier(n\_estimators=100, learning\_rate=0.1, max\_depth=3)

model.fit(X\_train, y\_train)

# Предвидувања

y\_pred = model.predict(X\_test)

# Евалуација

accuracy = accuracy\_score(y\_test, y\_pred)

print("Точност:", accuracy)

Доколку сакаш дополнителни објаснувања или практичен пример за некој специфичен boosting алгоритам, само кажи!

**1. AdaBoost (Adaptive Boosting):**

* **Како функционира:**
  + Секој слаб модел (обично одлучувачко дрво со мала длабочина) се тренира последователно.
  + На примерите кои се погрешно класифицирани од претходниот модел им се доделуваат поголеми тежини.
  + Следниот модел ќе се фокусира повеќе на тие примери за да ги подобри предвидувањата.
  + Финалната предвидена вредност е тежински просек од сите модели.
* **Главна карактеристика:**
  + Работи со тежини на примерите.
  + Корекцијата на грешките е базирана на тоа кои примери се тешки за класификација.
* **Пример:** Ако одредени примери постојано се погрешно класифицирани, AdaBoost ќе ги даде повисоки тежини, така што следниот модел ќе се обиде да ги предвиди подобро.
* **Кога е корисен:**
  + Кога грешките во класификацијата доаѓаат од погрешно класифицирани податоци, а не од шум.
  + За побрзи модели на мали податочни сетови.

**2. Gradient Boosting:**

* **Како функционира:**
  + Секој нов слаб модел се тренира за да ги минимизира **резидуалите** (разликата помеѓу вистинските вредности и предвидувањата на претходниот модел).
  + Ги користи градиентите (први изводи) за да ја минимизира загубната функција (на пример, Mean Squared Error за регресија или Log Loss за класификација).
  + Финалниот резултат е збир на предвидувањата од сите модели, но со минимизација на загубната функција.
* **Главна карактеристика:**
  + Работи со **градиенти** за да ги поправи грешките.
  + Корекцијата на грешките е базирана на разликите помеѓу вистинските и предвидените вредности.
* **Пример:** Наместо да се фокусира на тежини, Gradient Boosting пресметува колку секое следно дрво може да ја поправи моменталната загуба и да ја намали грешката.
* **Кога е корисен:**
  + За поголеми и покомплексни податочни сетови.
  + Кога е потребно фино подесување на перформансите, бидејќи е флексибилен.

Изборот на **learning rate** (стапка на учење) е еден од најважните чекори при тренирање на машински модели, особено кај алгоритми за бустинг како Gradient Boosting, AdaBoost или дури и кај невронски мрежи. **Learning rate** (α\alpha) го контролира колку моделот се ажурира како одговор на грешките за време на тренингот. Еве како може правилно да се избере:

**1. Основни насоки за learning rate**

* **Почнете со мали вредности:** Learning rate најчесто се бира во опсегот од 0.010.01 до 0.10.1 за бустинг алгоритми.
  + 0.1 (стандардна вредност во многу библиотеки, како XGBoost)\text{0.1 (стандардна вредност во многу библиотеки, како XGBoost)} е добар почеток.
  + Ако користите многу мала вредност (на пр. 0.0010.001), моделот ќе бара повеќе итерации, што го зголемува времето на пресметка.
* **Компромис меѓу learning rate и број на итерации:**
  + **Поголема learning rate**: бара помалку итерации, но може да предизвика „прескокнување“ на оптималното решение или нестабилност.
  + **Пониска learning rate**: бара повеќе итерации, но нуди постабилно и постепено учење.

**2. Проба и грешка (Grid Search/Cross-Validation)**

* Користете **Grid Search** или **Cross-Validation** за да експериментирате со различни вредности.
  + На пример, тестирајте: α=[0.001,0.01,0.05,0.1,0.2]\alpha = [0.001, 0.01, 0.05, 0.1, 0.2].
  + Одберете ја онаа вредност која резултира со најдобра прецизност на валидацискиот сет.

**3. Пристап со намалување на стапката на учење**

* **Зголемете број на итерации:** Ако користите многу мала learning rate (на пр. 0.010.01), зголемете го бројот на модели (итерации).
  + На пример, во Gradient Boosting, ако го намалите α\alpha на половина, зголемете го бројот на базни модели.

**4. Практични совети:**

* **Ако learning rate е преголема:** Моделот може да не конвергира (т.е., грешката останува висока и нестабилна).
* **Ако е премала:** Моделот може да конвергира премногу бавно и да бара премногу ресурси.
* **Користете адаптивни алгоритми:** Алатки како AdaBoost автоматски ја прилагодуваат стапката на учење.

**5. Графичка евалуација:**

* За време на тренингот, визуализирајте ја грешката (loss function) преку итерации.
  + Ако грешката брзо паѓа, но потоа осцилира или расте → learning rate е висока.
  + Ако грешката опаѓа многу бавно → learning rate е прениска.

На кратко, изберете балансирана вредност и секогаш проверувајте со валидациски податоци за оптимални резултати.

**Наивен Баесов класификатор (Naive Bayes Classifier)**

Наивниот Баесов класификатор е еден од наједноставните и најбрзите алгоритми за машинско учење, кој се базира на **теоремата на Баес** и се користи за решавање на проблеми на **класификација**.

**Како функционира?**

**Теорема на Баес:**

Најпрво, ја користи следната формула за пресметка на веројатноста на даден клас CC, базирана на дадените карактеристики XX:

P(C∣X)=P(X∣C)⋅P(C)P(X)P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)}

* **P(C∣X)P(C|X)**: Веројатноста на класот CC за дадените карактеристики XX (постериорна веројатност).
* **P(X∣C)P(X|C)**: Веројатноста на карактеристиките XX за даден клас CC (веројатност условена на класата).
* **P(C)P(C)**: Веројатноста на класот CC (приоритетна веројатност).
* **P(X)P(X)**: Веројатноста на карактеристиките XX (маргинална веројатност).

**Наивност на алгоритмот**

* **Наивна претпоставка**: Се претпоставува дека сите карактеристики X1,X2,...,XnX\_1, X\_2, ..., X\_n се **независни** една од друга.
* Иако оваа претпоставка е ретко вистинита во реални податоци, алгоритмот работи многу добро за многу проблеми.

**Видови на Наивен Баесов класификатор**

1. **Gaussian Naive Bayes**:
   * Користи **Гаусова распределба** за континуирани податоци.
   * Претпоставува дека податоците следат нормална распределба.
2. **Multinomial Naive Bayes**:
   * Најчесто се користи за текстуални податоци.
   * Класично се применува кај **анализата на фреквенции на зборови** (e.g., број на зборови во документи).
3. **Bernoulli Naive Bayes**:
   * Работи со бинарни податоци (0 и 1).
   * Погоден за проблеми каде што карактеристиките се бинарни (присуство или отсутност на некоја карактеристика).

**Примена**

* **Класификација на текст**: E-mail spam detection, sentiment analysis.
* **Препознавање на слики**: Препознавање ракопис, OCR.
* **Дијагноза во медицина**: Прогноза на болести базирано на симптоми.

**Предности**

1. **Едноставен и брз** за имплементација.
2. **Скалабилен**: Може да работи со многу карактеристики.
3. Добро функционира за **мали податоци**.
4. Лесен за разбирање и интерпретација.

**Недостатоци**

1. Претпоставката за **независност на карактеристиките** ретко е точна.
2. **Невалидни резултати за 0-фреквенции**: Ако некоја карактеристика нема вредност за одреден клас во тренинг сетот, веројатноста P(X∣C)P(X|C) ќе биде 0. Ова се решава со **Laplace Smoothing**.

**Пример со Python**

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB

# Вчитување на податоци

data = load\_iris()

X = data.data

y = data.target

# Делба на тренинг и тест сет

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Креирање на модел

model = GaussianNB()

# Тренирање на моделот

model.fit(X\_train, y\_train)

# Тестирање и точност

accuracy = model.score(X\_test, y\_test)

print(f"Точност: {accuracy:.2f}")

Наивниот Баесов класификатор е посебно корисен за **текстуални податоци** и проблеми на класификација каде што независноста на карактеристиките не е критичен услов.

A screenshot of a blue screen

Description automatically generated

MULTI-CLASS -> секоја редица има една вредност

MULTI-LABEL -> во една редица може да има повеќе вредности

A diagram of different types of objects

Description automatically generated

Логистичка регресија – иако е регресија станува збор за класификатор. Претставува регресија за класификација.

Метрики за евалуирање - R2, Intercept, Slope, F-statistiv, Log-likelihood

A diagram of different colored lines

Description automatically generated with medium confidence

A diagram of a diagram of a number of dots

Description automatically generated with medium confidence

Entropija

ЏИНИ ИНДЕКС

A diagram of a high risk and high risk customer

Description automatically generated with medium confidence

