Продвинутые методы машинного обучения

# Регуляризация и отбор признаков

## Содержание лекции

1. Проблема переобучения и Bias-Variance Tradeoff

2. Методы регуляризации: Ridge, Lasso, Elastic Net

3. Отбор признаков: Filter, Wrapper, Embedded методы

4. Настройка гиперпараметров

5. Практические рекомендации

## Введение

**Регуляризация** - это набор методов предотвращения переобучения путем добавления "штрафа" за сложность модели. В реальных задачах машинного обучения данные часто содержат шум, избыточные признаки и мультиколлинеарность, что приводит к нестабильным и неточным моделям.

Основные проблемы, решаемые регуляризацией:

* • Переобучение при высокой размерности данных
* • Нестабильность коэффициентов при мультиколлинеарности
* • Плохая обобщающая способность модели
* • Сложность интерпретации из-за избыточных признаков

## 1. Проблема переобучения

Переобучение возникает, когда модель слишком хорошо "запоминает" обучающие данные,

но плохо обобщает на новые данные.

### Bias-Variance Tradeoff

Ключевая концепция в машинном обучении:

* • Bias (смещение): ошибка из-за упрощения модели
* • Variance (разброс): чувствительность к изменениям в данных
* • Оптимальная модель: минимизирует общую ошибку (bias + variance)

#### Демонстрация переобучения:

# Создание данных с переобучением  
from sklearn.datasets import make\_regression  
from sklearn.linear\_model import LinearRegression  
from sklearn.model\_selection import train\_test\_split  
  
# Много признаков, мало данных - рецепт переобучения  
X, y = make\_regression(n\_samples=100, n\_features=80, noise=0.1)  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3)  
  
# Базовая модель  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
  
train\_score = model.score(X\_train, y\_train) # Очень высокий  
test\_score = model.score(X\_test, y\_test) # Низкий  
print(f"Переобучение: {train\_score - test\_score:.3f}")

## 2. Ridge регрессия (L2 регуляризация)

**Ridge** добавляет штраф за большие коэффициенты, "сжимая" их к нулю. Это предотвращает переобучение и повышает стабильность модели.

Ключевые особенности Ridge:

* • Сжимает коэффициенты, но не обнуляет их полностью
* • Эффективно при мультиколлинеарности
* • Параметр α (alpha) контролирует силу регуляризации
* • Требует стандартизации признаков

#### Практический пример Ridge:

from sklearn.linear\_model import Ridge  
from sklearn.preprocessing import StandardScaler  
from sklearn.model\_selection import validation\_curve  
import numpy as np  
  
# Стандартизация обязательна!  
scaler = StandardScaler()  
X\_scaled = scaler.fit\_transform(X\_train)  
  
# Поиск оптимального alpha  
alphas = np.logspace(-4, 4, 20)  
train\_scores, test\_scores = validation\_curve(  
 Ridge(), X\_scaled, y\_train, param\_name="alpha",  
 param\_range=alphas, cv=5)  
  
# Выбираем лучший alpha  
best\_alpha = alphas[np.argmax(test\_scores.mean(axis=1))]  
  
# Финальная модель  
ridge\_model = Ridge(alpha=best\_alpha)  
ridge\_model.fit(X\_scaled, y\_train)  
  
print(f"Ridge Test R²: {ridge\_model.score(X\_test\_scaled, y\_test):.3f}")

## 3. Lasso регрессия (L1 регуляризация)

**Lasso** не только сжимает коэффициенты, но и может полностью их обнулять, автоматически выполняя отбор наиболее важных признаков.

Преимущества Lasso:

* • Автоматический отбор признаков
* • Создает разреженные модели
* • Повышает интерпретируемость
* • Эффективно при избыточных признаках

#### Практический пример Lasso:

from sklearn.linear\_model import Lasso  
  
# Lasso с поиском оптимального alpha  
alphas\_lasso = np.logspace(-4, 1, 20) # Меньшие alpha  
  
best\_lasso = None  
best\_score = -np.inf  
  
for alpha in alphas\_lasso:  
 model = Lasso(alpha=alpha, max\_iter=5000)  
 model.fit(X\_scaled, y\_train)  
 score = model.score(X\_test\_scaled, y\_test)  
 if score > best\_score:  
 best\_score = score  
 best\_lasso = model  
  
# Анализ отбора признаков  
selected\_features = np.sum(np.abs(best\_lasso.coef\_) > 1e-6)  
total\_features = len(best\_lasso.coef\_)  
  
print(f"Отобрано: {selected\_features}/{total\_features} признаков")  
print(f"Lasso Test R²: {best\_score:.3f}")

## 4. Elastic Net - лучшее из двух миров

**Elastic Net** комбинирует преимущества Ridge и Lasso, позволяя одновременно сжимать коэффициенты и выполнять отбор признаков.

Параметры Elastic Net:

* • α (alpha): общая сила регуляризации
* • l1\_ratio: баланс между L1 и L2 (0=Ridge, 1=Lasso)
* • Оптимален для высокоразмерных данных

#### Практический пример Elastic Net:

from sklearn.linear\_model import ElasticNet  
from sklearn.model\_selection import GridSearchCV  
  
# Сетка параметров  
param\_grid = {  
 "alpha": np.logspace(-4, 1, 10),  
 "l1\_ratio": [0.1, 0.3, 0.5, 0.7, 0.9]  
}  
  
# Grid Search с кросс-валидацией  
elastic\_net = ElasticNet(max\_iter=5000)  
grid\_search = GridSearchCV(  
 elastic\_net, param\_grid, cv=5, scoring="r2")  
  
grid\_search.fit(X\_scaled, y\_train)  
  
# Лучшая модель  
best\_en = grid\_search.best\_estimator\_  
test\_score = best\_en.score(X\_test\_scaled, y\_test)  
  
print(f"Лучшие параметры: {grid\_search.best\_params\_}")  
print(f"Elastic Net Test R²: {test\_score:.3f}")

## 5. Методы отбора признаков

**Отбор признаков** - это процесс выявления наиболее информативных характеристик данных. Правильный отбор улучшает качество модели и снижает вычислительные затраты.

### Типы методов отбора признаков:

**1. Filter методы** - статистические тесты, не зависящие от модели:

* • SelectKBest с F-тестом
* • Взаимная информация (Mutual Information)
* • Корреляционный анализ

**2. Wrapper методы** - используют конкретную модель для оценки:

* • Recursive Feature Elimination (RFE)
* • Forward/Backward Selection

**3. Embedded методы** - встроенные в алгоритм обучения:

* • Lasso регуляризация
* • Random Forest feature importance
* • Gradient Boosting feature importance

#### Практические примеры отбора признаков:

from sklearn.feature\_selection import SelectKBest, f\_regression, RFE  
from sklearn.ensemble import RandomForestRegressor  
  
# 1. Filter: SelectKBest  
selector = SelectKBest(score\_func=f\_regression, k=20)  
X\_selected = selector.fit\_transform(X\_scaled, y\_train)  
  
# 2. Wrapper: RFE  
rfe = RFE(LinearRegression(), n\_features\_to\_select=20)  
X\_rfe = rfe.fit\_transform(X\_scaled, y\_train)  
  
# 3. Embedded: Random Forest  
rf = RandomForestRegressor(n\_estimators=100, random\_state=42)  
rf.fit(X\_scaled, y\_train)  
  
# Важность признаков  
importances = rf.feature\_importances\_  
top\_features = np.argsort(importances)[::-1][:20]  
X\_rf\_selected = X\_scaled[:, top\_features]  
  
print(f"SelectKBest: {X\_selected.shape[1]} признаков")  
print(f"RFE: {X\_rfe.shape[1]} признаков")  
print(f"Random Forest: {X\_rf\_selected.shape[1]} признаков")

## 6. Настройка гиперпараметров

**Правильная настройка гиперпараметров** критически важна для эффективности регуляризации. Основные стратегии поиска: Grid Search и Random Search.

### Стратегии поиска гиперпараметров:

**Grid Search** - исчерпывающий поиск по заданной сетке:

* • Гарантирует нахождение оптимума в заданной области
* • Может быть медленным при большом количестве параметров
* • Подходит для небольших сеток параметров

**Random Search** - случайный поиск в пространстве параметров:

* • Эффективнее Grid Search при многих параметрах
* • Может пропустить оптимум, но находит хорошие решения
* • Рекомендуется для начального исследования

#### Сравнение стратегий поиска:

from sklearn.model\_selection import RandomizedSearchCV  
from scipy.stats import loguniform  
import time  
  
# Grid Search  
param\_grid = {  
 "alpha": np.logspace(-4, 2, 10),  
 "l1\_ratio": np.linspace(0.1, 0.9, 5)  
}  
  
start = time.time()  
grid\_search = GridSearchCV(  
 ElasticNet(), param\_grid, cv=5, n\_jobs=-1)  
grid\_search.fit(X\_scaled, y\_train)  
grid\_time = time.time() - start  
  
# Random Search  
param\_dist = {  
 "alpha": loguniform(1e-4, 100),  
 "l1\_ratio": np.linspace(0.1, 0.9, 100)  
}  
  
start = time.time()  
random\_search = RandomizedSearchCV(  
 ElasticNet(), param\_dist, n\_iter=50, cv=5, n\_jobs=-1)  
random\_search.fit(X\_scaled, y\_train)  
random\_time = time.time() - start  
  
print(f"Grid Search: {grid\_time:.1f}s, Score: {grid\_search.best\_score\_:.3f}")  
print(f"Random Search: {random\_time:.1f}s, Score: {random\_search.best\_score\_:.3f}")

## 7. Практические рекомендации

### Когда использовать каждый метод:

**Ridge регрессия:**

* • Много коррелированных признаков
* • Нужны все признаки в модели
* • Стабильность важнее интерпретируемости

**Lasso регрессия:**

* • Много избыточных признаков
* • Нужна простая интерпретируемая модель
* • Автоматический отбор признаков

**Elastic Net:**

* • Высокоразмерные данные (p >> n)
* • Группы коррелированных признаков
* • Баланс между отбором и стабильностью

### Пошаговый алгоритм применения регуляризации:

1. 1. Проанализируйте данные:

• Соотношение объектов к признакам (n vs p)

• Наличие мультиколлинеарности

• Количество потенциально избыточных признаков

1. 2. Подготовьте данные:

• Обязательная стандартизация признаков

• Обработка пропущенных значений

• Разделение на train/validation/test

1. 3. Выберите метод регуляризации:

• Ridge: если нужны все признаки

• Lasso: если нужен отбор признаков

• Elastic Net: универсальный выбор

1. 4. Настройте гиперпараметры:

• Используйте кросс-валидацию

• Начните с Random Search

• Уточните Grid Search в перспективной области

1. 5. Оцените результаты:

• Качество на независимом тестовом наборе

• Стабильность коэффициентов

• Интерпретируемость модели

### Частые ошибки и как их избежать:

* ❌ Забывают стандартизировать признаки

✅ Всегда применяйте StandardScaler перед регуляризацией

* ❌ Настраивают параметры на тестовом наборе

✅ Используйте кросс-валидацию на обучающем наборе

* ❌ Используют слишком широкий диапазон α

✅ Начните с логарифмической шкалы от 1e-4 до 100

* ❌ Игнорируют интерпретируемость результатов

✅ Анализируйте коэффициенты и отобранные признаки

## Заключение

**Регуляризация** - это мощный инструмент современного машинного обучения, который помогает создавать стабильные и обобщающие модели. Правильное применение методов регуляризации требует понимания специфики данных и задачи, но результат того стоит.

**Ключевые принципы:**

* • Регуляризация предотвращает переобучение
* • Выбор метода зависит от специфики задачи
* • Кросс-валидация обязательна для настройки
* • Стандартизация данных критически важна
* • Интерпретируемость не менее важна качества