Машинное обучение в науках о Земле

Михаил Криницкий

к.т.н., н.с.
Институт океанологии РАН им. П.П. Ширшова

Лаборатория взаимодействия океана и атмосферы и мониторинга климатических изменений (ЛВОАМКИ)

Задачи классификации
(продолжение)

Михаил Криницкий

к.т.н., н.с.
Институт океанологии РАН им. П.П. Ширшова

Лаборатория взаимодействия океана и атмосферы и мониторинга климатических изменений (ЛВОАМКИ)

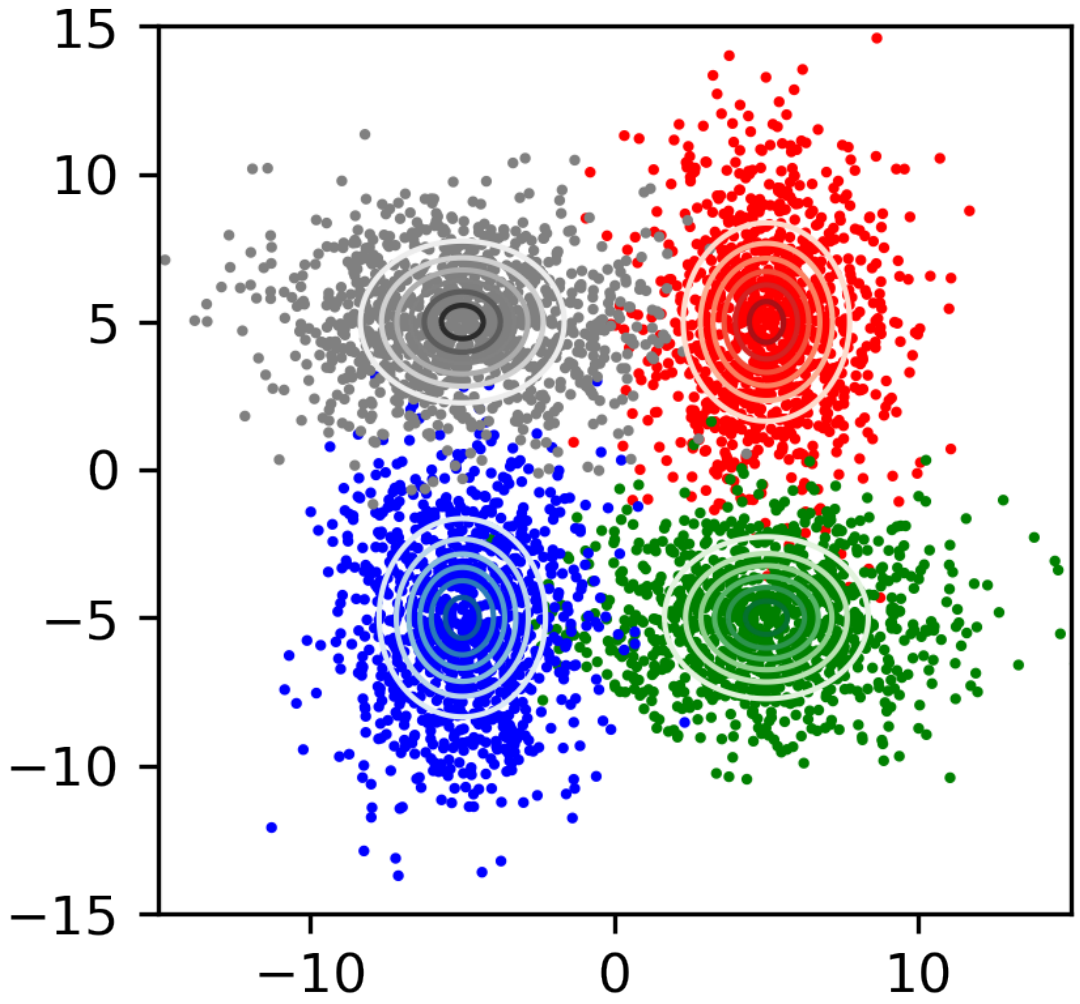
ПЛАН ЛЕКЦИИ

- Мультиномиальная логистическая регрессия
- Автоматическое вычисление градиента
- ДЗ: задача классификации рукописных цифр

Мультиномиальная логистическая регрессия

Цель: оценить **вероятность** классов (**0**, **1**, **2**, **3**) для объекта, описываемого значением x.

$$P(Y = k|X = x)$$



Мультиномиальная логистическая регрессия

(1) Предполагаем, что переменная Y для каждого класса распределена согласно распр.-ю Бернулли с параметром p (зависящий от x_i);
(2) Вероятности всех классов для одного объекта должны в сумме давать единицу
(3) Предполагаем, что оценка вероятности p_C класса C для объекта x_i оценивается согласно обобщенной линейной модели: $p(\theta, x_i, k) \propto \exp(\theta_k \cdot x_i)$

$$Y \sim \mathcal{B}\big(p(\theta, x)\big)$$

$$p(\theta, x_i, k) \propto \exp(\theta_k \cdot x_i)$$

$$p(\theta, x_i, k) = \frac{\exp(\theta_k \cdot x_i)}{\sum_j \exp(\theta_j \cdot x_i)}$$

$$\ell(\mathcal{T}, \theta) = -\sum_{i=1}^{N} \sum_{k=1}^{C} y_{ik} * \log(p(\theta, x_i, k))$$

Autograd, JAX

In [1]:
```
import jax.numpy as jnp
from jax import grad, jit, vmap
```

[9]:
```
def sum_logistic(x):
  return np.sum(1.0 / (1.0 + np.exp(-x)))

x_small = np.arange(3.)
derivative_fn = grad(sum_logistic)
print(derivative_fn(x_small))
```

```
[0.25       0.19661197 0.10499357]
```

JAX: Autograd and XLA build passing

Quickstart | Transformations | Install guide | Change logs | Reference docs

Announcement: JAX 0.1.58 has dropped Python 2 support, and requires Python 3.5 or newer. See docs/CHANGELOG.rst.

What is JAX?

JAX is Autograd and XLA, brought together for high-performance machine learning research.

With its updated version of Autograd, JAX can automatically differentiate native Python and NumPy functions. It can differentiate through loops, branches, recursion, and closures, and it can take derivatives of derivatives of derivatives. It supports reverse-mode differentiation (a.k.a. backpropagation) via `grad` as well as forward-mode differentiation, and the two can be composed arbitrarily to any order.

What's new is that JAX uses XLA to compile and run your NumPy programs on GPUs and TPUs. Compilation happens under the hood by default, with library calls getting just-in-time compiled and executed. But JAX also lets you just-in-time compile your own Python functions into XLA-optimized kernels using a one-function API, `jit`. Compilation and automatic differentiation can be composed arbitrarily, so you can express sophisticated algorithms and get maximal performance without leaving Python. You can even program multiple GPUs or TPU cores at once using `pmap`, and differentiate through the whole thing.

Dig a little deeper, and you'll see that JAX is really an extensible system for composable function transformations. Both `grad` and `jit` are instances of such transformations. Others are `vmap` for automatic vectorization and `pmap` for single-program multiple-data (SPMD) parallel programming of multiple accelerators, with more to come.

This is a research project, not an official Google product. Expect bugs and sharp edges. Please help by trying it out, reporting bugs, and letting us know what you think!

Реализация мультиномиальной логистической регрессии

$$p(\theta, x_i, k) = \frac{\exp(\theta_k \cdot x_i)}{\sum_j \exp(\theta_j \cdot x_i)}$$

$$\ell(\mathcal{T}, \theta) = -\sum_{i=1}^{N}\sum_{k=1}^{C} y_{ik} * \log(p(\theta, x_i, k))$$

преобразования для вычислительной стабильности

$$z_{ki} = \theta_k \cdot x_i$$

$$p(\theta, x_i, k) = \frac{\exp(z_{ki})}{\sum_j \exp(z_{ji})} = \frac{C * \exp(z_{ki})}{C * \sum_j \exp(z_{ji})} = \{C^* = \ln C\} = \frac{\exp(z_{ki} + C^*)}{\sum_j C * \exp(z_{ji})} = \frac{\exp(z_{ki} + C^*)}{\sum_j \exp(z_{ji} + C^*)} = \{C^* = -\max_{j,i}(z_{ji})\} =$$

$$= \frac{\exp(z_{ki} - \max(z))}{\sum_j \exp(z_{ji} - \max(z))}$$

Преобразование для снижения порядка чисел z: сделаем все признаки x в интервале от 0 до 1. Причины:

- Для стабилизации вычислений: при небольших числах x есть шанс, что значения z не будут слишком большими, при условии, что θ инициализируется не слишком большими числами; здесь $\theta^{t=0} \sim \mathcal{N}(0,1)$.
- Для стабилизации вычислений: при больших различиях в порядках величин x градиент функции потерь может иметь сильные различия по порядку величин для различных компонент вектора параметров θ, что может приводить к большой неопределенности в оценке θ при применении градиентных методов оптимизации.

Мультиномиальная логистическая регрессия