**> Введение**

В этом домашнем задании необходимо реализовать обучение ListNet на подвыборке датасета MSRANK\_10K (его можно взять из catboost, см. пример в шаблоне кода). Перед выполнением домашней работы полезно освежить знания по PyTorch в рамках официальных учебных материалов по ссылкам:

[Основы](https://pytorch.org/tutorials/beginner/basics/intro.html) (блоки 0, 1, 4 и 6)

[60-минутный блиц](https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html) (блоки Tensors и Neural Networks)

В первую очередь необходимо согласно разобранному на практике материалу определить структуру ListNet (в self.model). В рамках выполнения задания попробуйте разные конфигурации слоёв (Dropout, нелинейности) и разное количество нейронов. Однако имейте в виду, что даже с простой архитектурой из практической части вполне реально обучить модель, проходящую проверку.

**Класс Solution**

Весь остальной код должен быть представлен в классе Solution. Параметры класса при инициализации:

n\_epochs — количество эпох или проходов по тренировочному датасету (сколько раз нейросеть увидит каждый объект).

listnet\_hidden\_dim — размерность скрытого слоя ListNet.

lr — Learning Rate при обучении модели (коэффициент, на который умножаются градиенты ошибки при обучении во время градиентного спуска).

ndcg\_top\_k — количество объектов, на которых рассчитывается NDCG при валидации.

Метод \_get\_data уже реализован — ознакомьтесь с кодом и более внимательно рассмотрите датасет MSRANK\_10K, данные которого будут храниться в датафреймах. Если вам интересно, описание всех признаков можно найти на официальном сайте [по ссылке](https://www.microsoft.com/en-us/research/project/mslr/). Именно на основе этих фичей и будет производиться оценка релевантности. Помимо признаков (X\_train, X\_test) и меток релевантности (y\_train, y\_test) также используются уникальные идентификаторы запросов (query\_ids\_train, query\_ids\_test). Именно в рамках группировки по этим идентификаторам и будет рассчитываться метрика NDCG.

Метод \_prepare\_data нужно дописать: необходимо отнормировать признаки X\_train и X\_test в рамках каждого отдельного идентификатора запроса. Для этого нужно имплементировать метод \_scale\_features\_in\_query\_groups, после чего вызвать его в рамках \_prepare\_data. Для нормализации необходимо использовать StandardScaler из библиотеки sklearn. После применения каждый признак в рамках группы будет иметь среднее нуль и стандартное отклонение 1. Это необходимо для корректного обучения нейронной сети.

**Не лик ли это? Как можно нормировать данные в валидации?**

Нет, это не лик, так как происходит нормировка в рамках каждой группы для каждого запроса. Как говорилось на одной из лекций, модели ранжирования лишь реранжируют уже некоторый подготовленный топ, и потому к моменту применения нейросети вектора признаков уже известны. Таким образом, модель, наоборот, получается более робастной, имея возможность сравнивать выровненные распределения в рамках каждого запроса. Главное, что сохраняется относительная информация (порядок), так как стандартизация&nbsp;— монотонное преобразование. Такой трюк часто применяется в моделях ранжирования и матчинга.

Затем нужно поместить все переменные с признаками и таргетами в тензоры (torch.FloatTensor) для удобства их дальнейшего использования во время обучения и валидации. Их размерности должны быть N∗D и N для признаков и лейблов соответственно, где N — количество объектов в выборке, а D — количество признаков. Целевые метки нужно разместить в атрибутах с названиями ys\_train и ys\_test соответственно для тренировочной и тестовой выборок.

В методе \_create\_model необходимо инициализировать (создать модель) реализованный вами ListNet согласно параметрам из init-метода класса Solution.

**Приступаем к реализации методов обучения и валидации модели**

Для начала нужно реализовать вычисление NDCG@k в методе \_ndcg\_k. Здесь разрешается переиспользовать код из домашнего задания к прошлой лекции. Важно внести изменения, чтобы расчёт шел только в топ-K позиций (включая IdealDCG).

В этой домашней работе и далее предполагается использование экспоненциальной версии вычисления NDCG, проверьте параметры функции во избежание недоразумений.

Метод \_calc\_loss принимает на вход целевые метки и предсказания и возвращает значение функции потерь. Можете использовать любую ListNet-like функцию потерь из разобранных на лекции (даже KL-дивергенцию).

В рамках \_train\_one\_epoch необходимо организовать разовый проход по всем группам из тренировочного датасета. Таким образом, на каждом шаге обучения ListNet подаются лишь те объекты выборки, которые относятся к одному id (ведь функция ошибки считается по набору документов, относящихся обязательно к одному запросу). Используйте для вычисления ошибки и оптимизации метод \_calc\_loss.

Для контроля хода обучения пригодится метод \_eval\_test\_set — он также проходит по группам объектов, получает предсказания модели, после чего рассчитывает NDCG@K и возвращает усреднённое по всем id значение. Если NDCG рассчитать невозможно или по каким-то причинам появляется ошибка, то NDCG=0 (а не пропускается).

И жемчужина класса Solution — метод fit, вызывающий обучение и валидацию модели в течение N эпох. Метод возвращает валидационные значения NDCG после каждой эпохи. Необходимо, чтобы ваша модель получала значение NDCG@10 не ниже 0.41 после тренировки в течение 5 эпох при lr=0.001. На тренировку отведено не более 60 секунд.