2. ВТОРОЙ ЭТАП

Задачи второго этапа

Задача 3.0.1. Белые или чёрные? (15 баллов)

Winning and losing isn't everything;

sometimes, the journey is just as important as the outcome.

Alex Morgan

В первой задаче вам предстоит анализировать шахматные партии. Вам дан датасет, в котором содержатся описания шахматных партий за некоторый период времени с одного сайта для онлайн-игры в шахматах. Вам гарантируется, что каждая партия закончилась победой либо чёрных, либо белых. Ваша задача будет состоять в том, чтобы для каждой партии ответить, какая сторона победила.

Формат входных данных

Вам надо будет скачать датасет с играми (`https://drive.google.com/file/d/1IsebLv8-AcLoBbIbl3gCVJ7cu_oo0eQV/view`). Он имеет типичный для машинного обучения формат .csv. Про каждую партию вам известна следующая информация:

1. 'id' – уникальный идентификатор партии
2. 'rated' – True/False – была ли партия рейтинговой
3. 'created_at' – время начала партии
4. 'last_move_at' – во сколько был совершён последний ход
5. 'turns' – сколько ходов было в партии
6. 'white_id' – уникальный идентификатор игрока белыми
7. 'white_rating' – рейтинг игрока белыми
8. 'black_id' – уникальный идентификатор игрока чёрными
9. 'black_rating' – рейтинг игрока чёрными

Формат выходных данных

В систему вам необходимо сдать файл, в котором будет один столбец с заголовком 'winner', в каждой ячейке которого будет написано 'black' или 'white'. Пример посылки. (`https://drive.google.com/file/d/1NE48oM5IuBGLbrJ1Dk9Noti4mdsmmj8X/view?usp=sharing`)

Система оценки

За эту задачу вы можете получить 0 или 15 баллов: 15 – в случае, если ваш ответ полностью верен, 0 – в противном случае.

Pandas - самая популярная библиотека для работы с табличными данными при анализе данных.

Пример программы-решения

Ниже представлено решение на языке Python3

```
import pandas as pd

df = pd.read_csv('input.csv', index_col=0)
ans_bool = df.turns.apply(lambda x: 'white' if x % 2 else 'black')
answer = pd.DataFrame({'winner' : ans_bool})
answer.to_csv('answer.csv', index=False)
```

Задача 3.0.2. Это легковушка? Это автобус? (15 баллов)

В этой задаче вам предстоит по данным GPS-навигации определять тип средства передвижения – это автобус или легковой автомобиль?

Формат входных данных

В качестве тренировочной выборки вам даны некоторые данные GPS-навигации о движении транспорта в Красноярске.
`https://docs.google.com/spreadsheets/d/1QdZJT05NiqgCDPYrPHJBWCK2SeORK2Fo6HN-ODT7cgE/edit?usp=sharing`

В качестве признаков имеются средняя скорость, время в пути, пройденная дистанция, rating – оценочный параметр, выражающий насколько данное средство передвижения удобно для перемещения по городу(3 – удобно, 2 – нормально, 1 – плохо), а также ответ – *'car'* или *'bus'*.

Помимо этого вам дана тестовая выборка(test.csv), набор столбцов которой отличается от обучающей только отсутствие столбца car_or_bus. Как вы уже могли догадаться, ваша задача – обучиться на обучающей выборке и научиться предсказывать для тестовой выборки тип каждого из указанных в ней средств передвижения (car или bus).

Формат выходных данных

Вам нужно сдать в систему файл, в котором для каждой строки из тестовой выборки указан предполагаемый вами ответ (car/bus). Пример посылки. (`https://drive.google.com/file/d/1dMOJiqO_UU7ey4o2dcVjhS23pfbKaea5/view?usp=sharing`)

Система оценки

Чем больше правильных ответов вы дадите, тем больше баллов вы заработаете. Назовём accuracy_score величину, равную отношению данных правильно ответов к общему количеству строк в test.csv.

Тогда:

1. Если 0.6 <= accuracy_score <= 0.85, то вы получаете за задачу 10 баллов
2. Если 0.85 <= accuracy_score <= 0.95, то вы получаете за задачу 20 баллов
3. Если 0.95 <= accuracy_score <= 1.0, то вы получаете за задачу 30 баллов

Обратите внимание, у вас есть только 5 попыток.

Решение

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from featexp import get_univariate_plots
```

Загрузка и исследование данных

Для загрузки данных из csv таблицы используем библиотеку pandas.

```
# Загружаем данные
data = pd.read_csv('train - train.csv')
data.head()
```

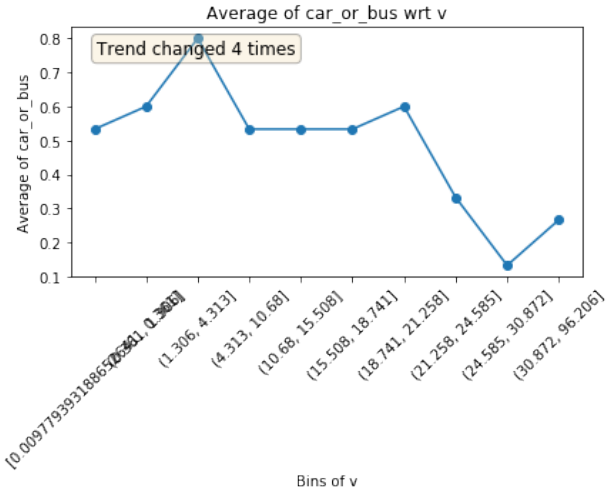
Для использования в калссификаторах классы должны быть закодированы числами. Заменим метку 'car' на 0 и 'bus' на 1

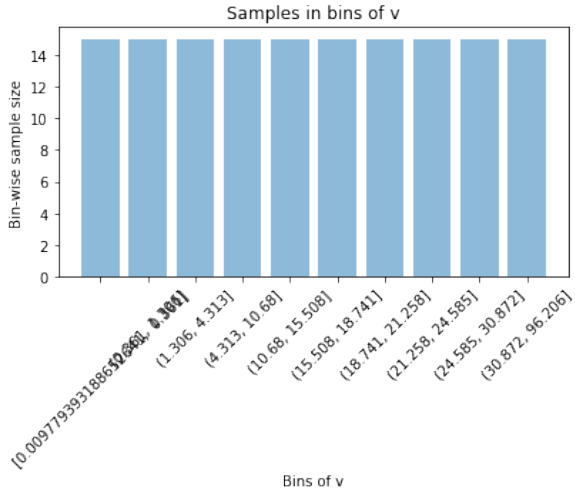
```
# Заменим метку car_or_bus на 0 или 1
mapping = {'car': 0,
           'bus': 1}
data['car_or_bus'] = data['car_or_bus'].map(mapping)
print("Dataset Size: {}".format(len(data)))
data.head()
```

Используем библиотеку featexp, чтобы построить графики зависимости класса от признаков. По оси x отложен признак, а по оси y отложено среднее значение метки. Так как 0 кодирует машину, а 1 кодирует автобус,

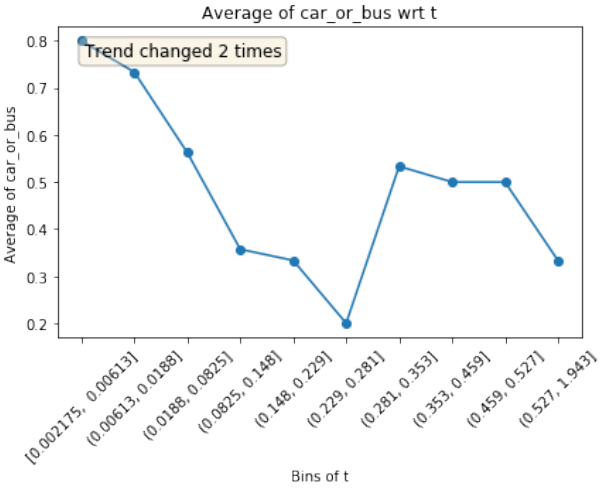
```
# Используем библиотеку featexp, чтобы получить красивые графики,
# показывающие зависимость метки от признаков
get_univariate_plots(data=data, target_col='car_or_bus',
    features_list=data.columns[:-1])
```

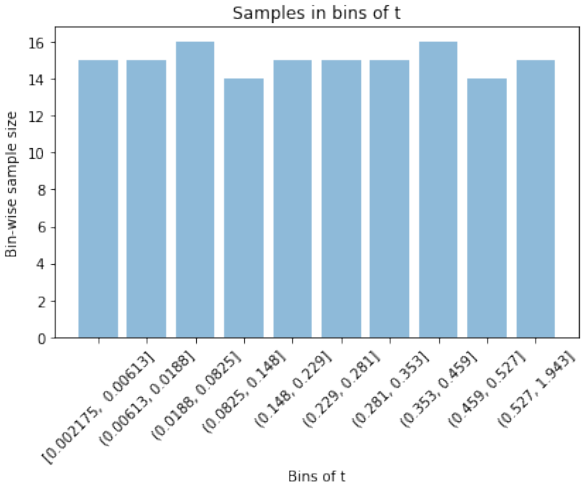
Plots for v



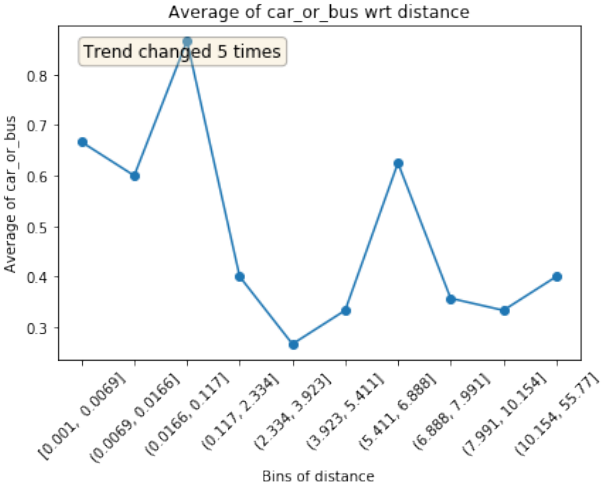


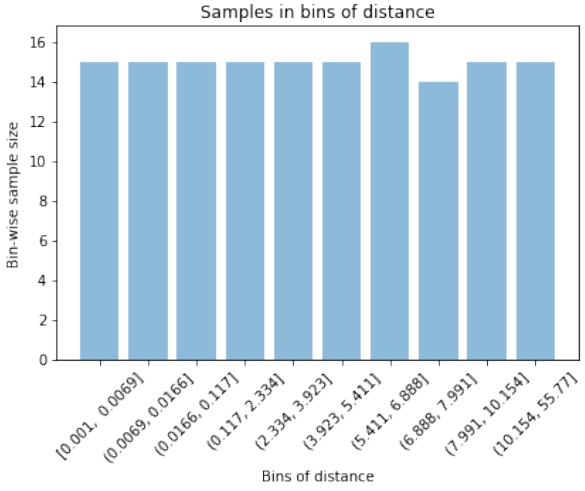
Plots for t



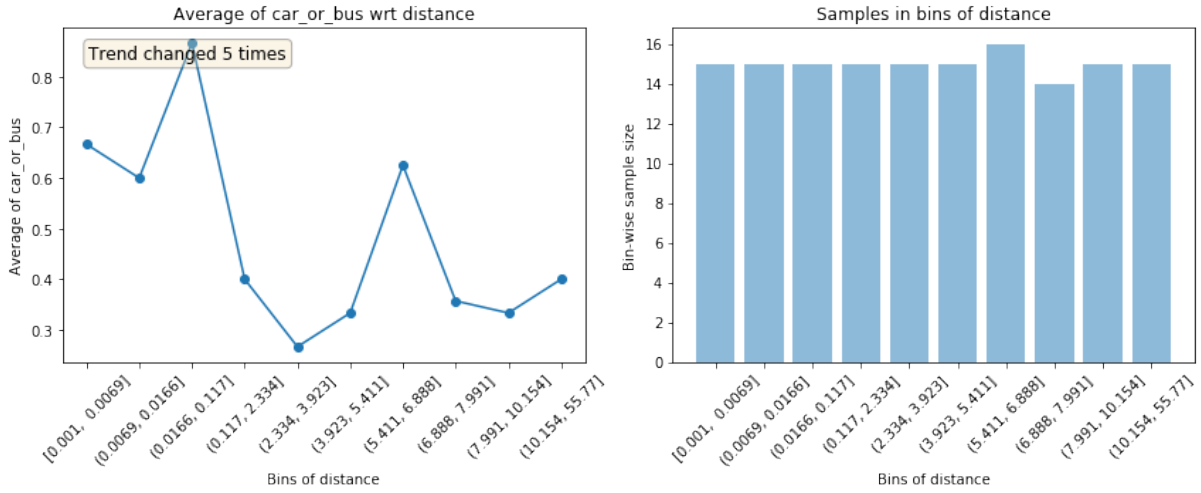


Plots for distance





Plots for rating



Все признаки имеют зависимость с классом, поэтому не стоит их выбрасывать.

Признак distance это $t \cdot v$ (в чем легко убедиться, перемножив столбцы). Он не вносит новых данных и можно попробовать его убрать. В результате эксперимента мы выяснили, что без признака distance точность всех классификаторов становится ниже, а значит, он важен для представления данных.

```
# Разделим данные на признаки и метки класса
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

# Мы будем использовать модели, для которых важно, чтобы признаки были численно
# одного порядка, поэтому мы нормализуем их (вычтем среднее и поделим на
# стандартное отклонение).
# Потом normalizer мы будем использовать для тестовых данных
normalizer = StandardScaler()
X = normalizer.fit_transform(X)
```

Так как у нас очень мало данных, то мы будем использовать особую стратегию для оценки моделей - Leave One Out. Мы будем выделять одно наблюдение из всего датасета, обучать модель на оставшихся наблюдениях, а затем этой моделью предсказывать метку для выделенного наблюдения.

Такая стратегия нужна для того, чтобы при валидации модель предсказывала метку только для наблюдений, которые она не видела в процессе тренировки, и при этом использовала максимум данных для обучения. В решении следующей задачи мы будем разделять выборку на две части - обучающую и валидационную, не использую такую сложную стратегию, потому что данных намного больше.

```
def loo_accuracy(model):
    """
    Производит оценку точности модели на датасете X, y (внешние переменные),
    использую стратегию Leave One Out. Возвращает сре
    """
    from sklearn.model_selection import LeaveOneOut

    loo = LeaveOneOut()
```

```
    correct = 0
    for train_ind, val_ind in loo.split(X, y):
        X_train,  y_train = X[train_ind], y[train_ind]
        X_val, y_val = X[val_ind], y[val_ind]

        model.fit(X_train, y_train)
        correct += (model.predict(X_val) == y_val)[0]

    return correct / len(X)
```

SVM

```
svm = SVC(gamma='auto')

# Выведем точность модели
loo_accuracy(svm)
```

Логистическая регрессия

```
# Инициализируем модель и обучим на данных
log_reg = LogisticRegression(C=0.1, solver='lbfgs')

# Выведем точность модели
loo_accuracy(log_reg)
```

N ближайших соседей

```
# Найдем лучшее значение для n_neighbors
best_n_neighbors = 0
best_acc = 0

for n_neighbors in range(1, 20):
    # Инициализируем модель и обучим на данных
    model = KNeighborsClassifier(n_neighbors=n_neighbors)

    # Найдем точность модели
    acc = loo_accuracy(model)

    if acc > best_acc:
        best_n_neighbors = n_neighbors
        best_acc = acc

print('Best accuracy: {:0.2f} for {} neighbors'.format(float(best_acc),
    best_n_neighbors))
```

Градиентный бустинг

```
# Найдем лучшее значение для n_estimators
best_n_estimators = 0
best_acc = 0

for n_estimators in range(1, 30):
    # Инициализируем модель и обучим на данных
    model = GradientBoostingClassifier(n_estimators=n_estimators,
        learning_rate=0.1)

```

```
    # Найдем точность на валидации
    acc = loo_accuracy(model)

    if acc > best_acc:
        best_n_neighbors = n_neighbors
        best_acc = acc

print('Best accuracy: {:0.2f} for {} estimators'.format(float(best_acc),
    best_n_neighbors))
```

Наивный байес

```
# Инициализируем модель и обучим на данных
model = GaussianNB()

# Выведем точность можеди
loo_accuracy(model)
```

Итоговая модель

Так как N ближайших соседей оказался лучшим, мы будем использовать именно его с лучшим n_neighbors

```
# Инициализируем модель, используя best_n_neighbors = 4
model = KNeighborsClassifier(n_neighbors=best_n_neighbors)

# обучим на всем датасете
model.fit(X, y)
```

Получение ответа

```
# Загрузим тестовые данные
data_test = pd.read_csv('DATASET2.csv')

# Нормализуем их, используя тот же normalizer, что и для обучающей выборки
X_test = normalizer.transform(data_test.values)

# Запишем результат в pandas.Series
result = pd.Series(model.predict(X_test))

# Преобразуем 0 и 1 в метки классов, сделав словарь с обратым маппингом
inv_map = {v: k for k, v in mapping.items()}
result = result.map(inv_map)
result

# Запишем результат в файл
result.to_csv('second_task_res.csv', index=False)
```

Если вы никогда раньше не работали с pandas и sklearn, то вы можете начать свое изучение с этой статьи (`https://habr.com/post/202090/`). Так же очень полезно пользоваться документацией по sklearn и pandas, в большинстве случаев достаточно вбить в поисковую строку "pandas/sklearn + (название вещи, про которую вы хотите узнать)".

GitHub страница библиотеки featexp с документацией (`https://github.com/abhayspawar/featexp`).

Отличный курс (`https://github.com/Yorko/mlcourse.ai`) по машинному обучению от OpenDataScience (сейчас он идет на английском, но русские итерации тоже бывают)

Задача 3.0.3. О чём эта новость? (60 баллов)

В этой задаче вам предстоит для некоторого набора новостных заголовков и текстов определить тематику новости.

Новость может быть одного из 7 типов:

1. Мир – “world”
2. Наука – “science”
3. Культура – “culture”
4. Экономика – “economy”
5. Политика – “politics”
6. Общество – “society”
7. Религия – “religion”

Особенностью данной задачи является то, что обучающий датасет и модель вам предстоит сделать самим.

Для сбора и структурирования датасета новостей жюри рекомендует использовать язык программирования Python и фреймворк Scrapy. Небольшой туториал по этой библиотеке вы можете найти на официальном сайте фреймворка (`https://doc.scrapy.org/en/latest/intro/tutorial.html`).

Формат входных данных

Вам дан датасет с новостями (test.csv), в котором содержатся заголовки и тексты каждой статьи из подборки жюри (все эти новости реальные и взяты из архивов новостных сайтов разных годов).

Формат выходных данных

Вы должны сдать файл, в котором для каждой новостной статьи указана одна из тематик из списка выше. Правильный формат сдаваемого файла вы можете увидеть в файле (sample_submission.csv
`https://drive.google.com/file/d/1EX5eUmNjgr5E3YpjCuZ6dmfohVIh1Izq/view?usp=sharing`).

Система оценки

Всего за эту задачу вы можете набрать до 60 баллов.

Определим accuracy_score как долю новостей, которые вы классифицировали правильно к общему количеству новостей в test.csv. Чем эта метрика выше, тем больше баллов вы получите, а именно:

1. Если 0.3 <= accuracy_score <= 0.5, то вы получаете 10 баллов
2. Если 0.5 <= accuracy_score <= 0.6, то вы получаете 20 баллов.
3. Если 0.6 <= accuracy_score <= 0.7, то вы получаете 30 баллов.
4. Если 0.7 <= accuracy_score <= 0.8, то вы получаете 40 баллов.
5. Если 0.8 <= accuracy_score <= 0.9, то вы получаете 50 баллов.
6. Если 0.9 <= accuracy_score <= 1.0, то вы получаете 60 баллов.

Решение

```
# Флаг, определяет, использовать ли видеокарту при подсчёте
USE_CUDA = True

import pandas as pd
import numpy as np
import time

### Загрузим данные небольшой порцией
#data_text = pd.read_json('./news2.jsonlines', lines=True, chunksize=1000,
#encoding='utf8')
# data_text = pd.read_csv('./news2.csv', chunksize=1000, encoding='utf8')
# data_text = next(iter(data_text))

### Для загрузки всего датасета в оперативную память (её может не хватить)
data_text = pd.read_csv('./news2.csv', encoding='utf8')

from sklearn.preprocessing import LabelEncoder

label_enc = LabelEncoder()
data_text['class'] = label_enc.fit_transform(data_text['class'])
```

Загруженные данные разделим на train и test, сохраним их, т.к. DataLoader'ы считывают данные из файлов

```
from sklearn.model_selection import train_test_split
data_train, data_val = train_test_split(data_text, test_size=0.5)

data_train.to_csv('train.csv', encoding='utf-8', index=False)
data_val.to_csv('val.csv', encoding='utf-8', index=False)

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
```

pymorphy2 - библиотека, схожая с NLTK, но предназначенная для русского языка

morphy_tokenizer - лемматизирует слова в предложении и возвращает их список

Лемматизация — процесс приведения словоформы к лемме — её нормальной (словарной) форме.

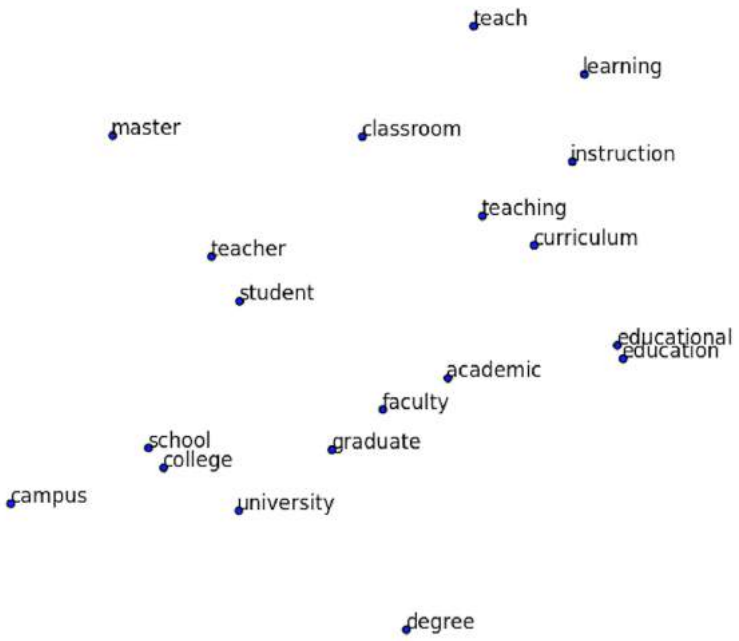
```
import pymorphy2
morph = pymorphy2.MorphAnalyzer()
def morphy_tokenizer(s):
    import re
    lemmatized_words = []

    for word in re.split("[^a-яA-ЯёЁA-Za-я]", s):
        word_repr = morph.parse(word.replace('.',''))
        if len(word_repr) != 0:
            normal_form = word_repr[0].lexeme[0].word
            lemmatized_words.append(normal_form)
    return lemmatized_words
```

Создадим класс рекуррентного слоя

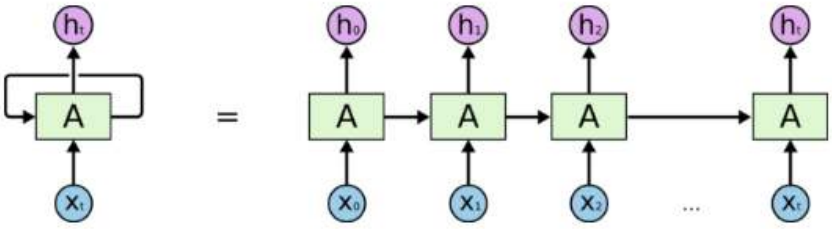
Embedding - замена каждого слова на вектор из некоторого количества вещественных чисел.

Indices	Latent Factors					
1	.32	.02	.48	.21	.56	.15
2	.65	.23	.41	.57	.03	.92
3	.45	.87	.89	.45	.12	.01
4	.65	.21	.25	.45	.78	.82

Так может выглядеть embedding матрица, в которой каждому индексу сопоставляется несколько чисел. Если смотреть embedding вектора как на вектора в некотором пространстве, то окажется, что близкие по смыслу слова окажутся близко и в этом пространстве.

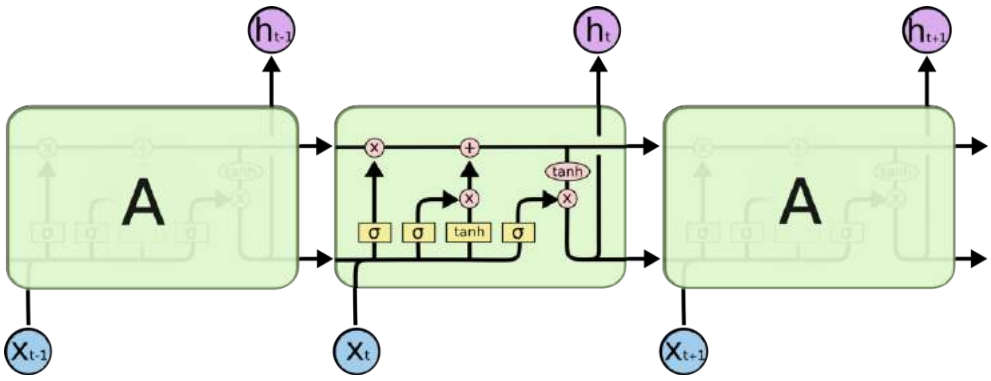


Сверху двумерные вектора слов после embedding'а.



An unrolled recurrent neural network.

Обычная рекуррентная нейросеть.



LSTM ячейка. Благодаря начилию долгосрочной памяти, она лучше обрабатывает длинные последовательности.

```
from torch.nn.utils.rnn import pack_padded_sequence, pad_packed_sequence

    class RNN(nn.Module):

        def __init__(self, hidden_size, encoder, num_layers=1):
            '''
            Args:
                hidden_size: размер вектора внутреннего состояния LSTM
                encoder: ссылка на слой эмбеддинга
                num_layers: количество LSTM слоев
            '''
            super(RNN, self).__init__()
            # embedding
            self.encoder = encoder
            # dropout после embedding
            self.drop_en = nn.Dropout(p=0.6)
            # рекуррентный слой
            self.rnn = nn.LSTM(input_size=encoder.weight.size(1),
                hidden_size=hidden_size, num_layers=num_layers, dropout=0.5,
                batch_first=True, bidirectional=True)
            # batchnorm после рекуррентных слоев
            self.bn2 = nn.BatchNorm1d(hidden_size*2)

        def forward(self, x, seq_lengths):
            '''
            Args:
                x: (batch, time_step, input_size)
                seq_lengths: длины последовательностей в x (чтобы их можно было
                    положить в тензор, они дополнены нулями справа)
            Returns:
                Выход полученный на последнем полносвязном слое без применения Softmax.
            '''
            # Применяем embedding и dropout
            x_embed = self.encoder(x)
            x_embed = self.drop_en(x_embed)
            # Запаковываем последовательности, чтобы рекуррентный слой мог работать с
            # ними, несмотря на то, что они имеют разные длины
            packed_input = pack_padded_sequence(x_embed, seq_lengths.cpu().numpy(),
                batch_first=True)

            # Получаем выход с рекуррентного слоя
            packed_output, ht = self.rnn(packed_input, None)
            # Распаковываем полученный выход
            out_rnn, _ = pad_packed_sequence(packed_output, batch_first=True)
```

```
45
46              row_indices = torch.arange(0, x.size(0)).long()
47              col_indices = seq_lengths - 1
48
49              if USE_CUDA:
50                  row_indices = row_indices.cuda()
51                  col_indices = col_indices.cuda()
52
53              # Из всех выходов LSTM получаем только выход последнего ненулевого символа
54              last_tensor=out_rnn[row_indices, col_indices, :]
55              # Применяем BatchNorm
56              out = self.bn2(last_tensor)
57
58              return out
```

Класс рекуррентной нейросети

Зададим архитектуру нейронной сети: у неё будет два входа для заголовка и текста. Для кодирования слов здесь используем word embedding

```
1  class Model(nn.Module):
2      def __init__(self, embedding_matrix, lstm_hdn, num_classes):
3          '''
4          Args:
5              embedding_matrix: матрица предобученных embedding'ов
6              lstm_hdn: размер внутреннего состояния для LSTM
7              num_classes: размерность вектора-реультата (т.е. количество классов,
8              на которые надо классифицировать)
9          '''
10         super().__init__()
11         # Создаем embedding слой и выключаем для него градиенты
12         self.encoder = nn.Embedding(embedding_matrix.size(0), embedding_matrix.size(1),
13             _weight=embedding_matrix)
14         self.encoder.weight.requires_grad = False
15
16         # предобученный embedding задает размер
17         self.text_rnn = RNN(lstm_hdn, self.encoder, num_layers=1)
18         self.title_rnn = RNN(lstm_hdn, self.encoder, num_layers=1)
19
20         # Выходной слой
21         self.l_out = nn.Linear(lstm_hdn*4, len(label_enc.classes_))
22
23     def forward(self, X_title, X_text, title_lengths, text_lengths,
24                 title_perm, text_perm):
25         # Так как для LSTM слоя важно, чтобы длины последовательностей в батче
26         # шли в невозрастающем порядке, наш dataloader возвращает перестановку
27         # (permutation), которая упорядочивает последовательности.
28         # В рекуррентную часть мы передаем последовательности отсортировано,
29         # а затем возвращаем исходный порядок
30         title_outp = self.title_rnn(X_title[title_perm], title_lengths[title_perm])
31         title_outp[title_perm] = title_outp
32
33         # Делаем то же самое для текста
34         text_outp = self.text_rnn(X_text[text_perm], text_lengths[text_perm])
35         text_outp[text_perm] = text_outp
36
37         # Объединяем выходы с двух рекуррентых модулей и отправляем в полносвязный слой
38         return self.l_out(torch.cat([title_outp, title_outp], dim=1))
```

Создадим класс-загрузчик данных

Класс-генератор загрузит данные из файла и предобработает их морфизатором и токенайзером. А также приведёт данные к виду, пригодному для использования в рекуррентной нейросети

```
class TextClassDataLoader(object):

def __init__(self, path_file, word_to_index, batch_size=32, shuffle=True):
    """
    Args:
        path_file:
        word_to_index:
        batch_size:
    """

    self.batch_size = batch_size
    self.word_to_index = word_to_index
    self.shuffle = shuffle

    if word_to_index is None:
        self.word_to_index = {'null': 0}

    # read file
    df = pd.read_csv(path_file).fillna('')
    df['text'] = df['text'].apply(morphy_tokenizer)
    df['text'] = df['text'].apply(self.generate_indexifyer())

    print("Columns: {}".format(list(df.columns.values)))

    df['title'] = df['title'].apply(morphy_tokenizer)
    df['title'] = df['title'].apply(self.generate_indexifyer())
    self.samples = df[['title', 'text', 'class']].values.tolist()

    # for batch
    self.n_samples = len(self.samples)
    self.n_batches = int(self.n_samples / self.batch_size)
    self.max_length = self._get_max_length()
    self._shuffle_indices()

    self.report()

def _shuffle_indices(self):
    if not self.shuffle:
        self.indices = np.arange(0, self.n_samples, 1)
    else:
        self.indices = np.random.permutation(self.n_samples)
    self.index = 0
    self.batch_index = 0

def _get_max_length(self):
    length = 0
    for sample in self.samples:
        length = max(length, len(sample[1]))
    return length

def generate_indexifyer(self):
    def indexify(lst_text):
        indices = []
        for word in lst_text:
            if word in self.word_to_index:
                indices.append(self.word_to_index[word])
```

```
            else:
                indices.append(0)
        return indices

    return indexify

@staticmethod
def _padding(batch_x):
    batch_s = sorted(batch_x, key=lambda x: len(x))
    size = len(batch_s[-1])
    for i, x in enumerate(batch_x):
        missing = size - len(x)
        batch_x[i] =  batch_x[i] + [0 for _ in range(missing)]
    return batch_x

def _create_batch(self):
    batch = []
    n = 0
    while n < self.batch_size:
        _index = self.indices[self.index]
        batch.append(self.samples[_index])
        self.index += 1
        n += 1
    self.batch_index += 1

    title, text, label = tuple(zip(*batch))

    # получаем длины всех текстов в батче
    text_seq_lengths = torch.LongTensor(list(map(len, text)))

    # создаем тензор, в котором в каждой строчке находится последовательность,
    # дополненная нулями справа.
    # NOTE: длина строки тензора должна быть такой же, как и самая длинная
    # последовательность мы могли бы взять большую длину, но в этом нет смысла,
    # так как все последовательности уже точно поместятся
    text_seq_tensor = torch.zeros((len(text), text_seq_lengths.max())).long()
    for idx, (seq, seqlen) in enumerate(zip(text, text_seq_lengths)):
        text_seq_tensor[idx, :seqlen] = torch.LongTensor(seq)

    # Проделывем то же самое для заголовков
    title_seq_lengths = torch.LongTensor(list(map(len, title)))

    # создаем тензор, в котором в каждой строчке находится последовательность,
    # дополненная нулями справа.
    # NOTE: длина строки тензора должна быть такой же, как и самая длинная
    # последовательность мы могли бы взять большую длину, но в этом нет смысла,
    # так как все последовательности уже точно поместятся
    title_seq_tensor = torch.zeros((len(title), title_seq_lengths.max())).long()
    for idx, (seq, seqlen) in enumerate(zip(title, title_seq_lengths)):
        title_seq_tensor[idx, :seqlen] = torch.LongTensor(seq)

    # Запомним перестановку, которая делает последовательности упорядоченными по длине,
    # чтобы передать ее в саму модель. Это важно так как в стандартный модуль LSTM
    # можно передавать только последовательности, идущие по уменьшению длины.
    _, text_perm_idx = text_seq_lengths.sort(0, descending=True)
    _, title_perm_idx = title_seq_lengths.sort(0, descending=True)

    # Переведем лэйблэ в тензор, чтобы не пришлось делать этого потом.
    # (Для работы функции потерь лэйблы должны лежать в pytorch тензоре)
    label = torch.LongTensor(label)
```

```
117
118     return text_seq_tensor, title_seq_tensor, label, text_seq_lengths, \
119             title_seq_lengths, text_perm_idx, title_perm_idx
120
121 def __len__(self):
122     return self.n_batches
123
124 def __iter__(self):
125     self._shuffle_indices()
126     for i in range(self.n_batches):
127         if self.batch_index == self.n_batches:
128             raise StopIteration()
129         yield self._create_batch()
130
131 def show_samples(self, n=10):
132     for sample in self.samples[:n]:
133         print(sample)
134
135 def report(self):
136     print('# samples: {}'.format(len(self.samples)))
137     print('max len: {}'.format(self.max_length))
138     print('# vocab: {}'.format(len(self.word_to_index)))
139     print('# batches: {} (batch_size = {})'.format(self.n_batches, self.batch_size))
```

Word embedding

Для эмбеддинга будем использовать часть предобученной на Википедии эмбеддинг-матрицы из fasttext. Она сопоставит каждому слову специальный вектор из 300 чисел.

`https://fasttext.cc/docs/en/crawl-vectors.html`

(файл прилагается с тетрадкой)

```
1  import pickle
2  word_to_index = pickle.load(open('word_index.pkl', 'rb'))
3  emb_tensor = torch.FloatTensor(np.load('embedding_matrix.npy', encoding='bytes'))
4
5  batch_size = 32
6
7  dataloaders = {'train': TextClassDataLoader('./train.csv', word_to_index, batch_size),
8                 'val': TextClassDataLoader('./val.csv', word_to_index, batch_size)}
9  dataset_sizes = {'train': dataloaders['train'].n_samples,
10                  'val': dataloaders['val'].n_samples}
```

```
1  from tqdm import tqdm_notebook
2  def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
3      since = time.time()
4
5      for epoch in range(num_epochs):
6          print('Epoch {}/{}'.format(epoch, num_epochs - 1))
7          print('-' * 10)
8
9          # каждя эпоха имеет обучающую и тестовую стадии
10         for phase in ['train', 'val']:
11             if phase == 'train':
12                 scheduler.step()
13                 model.train(True)  # установаить модель в режим обучения
14             else:
15                 model.train(False)  # установить модель в режим предсказания
```

```
16
17              running_loss = 0.0
18              running_corrects = 0
19
20              # итерируемся по батчам
21              for data in tqdm_notebook(dataloaders[phase]):
22                  new_data = list(data)
23                  if USE_CUDA:
24                      for i in range(len(data)):
25                          new_data[i] = data[i].cuda()
26
27                  # получаем картинки и метки
28                  X_title, X_text, label, title_lengths, text_lengths,
29                      title_perm, text_perm = new_data
30
31                  # инициализируем градиенты параметров
32                  optimizer.zero_grad()
33
34                  # forward pass
35                  outputs = model(X_title, X_text, title_lengths, text_lengths,
36                                  title_perm, text_perm)
37                  _, preds = torch.max(outputs.data, 1)
38                  loss = criterion(outputs, label)
39
40                  # backward pass + оптимизируем только если это стадия обучения
41                  if phase == 'train':
42                      loss.backward()
43                      optimizer.step()
44
45                  # статистика
46                  running_loss += loss.item()
47                  running_corrects += int(torch.sum(preds == label.data))
48
49              # Выводим статистику о качестве модели во время обучения
50              epoch_loss = running_loss / dataset_sizes[phase]
51              epoch_acc = running_corrects / dataset_sizes[phase]
52
53              print('{} Loss: {:.4f} Acc: {:.4f}'.format(
54                  phase, epoch_loss, epoch_acc))
55
56          print()
57
58      time_elapsed = time.time() - since
59      print('Training complete in {:.0f}m {:.0f}s'.format(
60          time_elapsed // 60, time_elapsed % 60))
61
62      return model, losses
```

Обучение

Передадим в нашу модель embedding матрицу

```
1  model = Model(emb_tensor, 32, len(label_enc.classes_))
2  if USE_CUDA:
3      model = model.cuda()
4
5  # В качестве cost function используем кросс-энтропию
6  loss_fn = nn.CrossEntropyLoss()
7
8  # В качестве оптимизатора - стохастический градиентный спуск
```

```
optimizer_ft = optim.Adam(model.parameters(), lr=0.001)

# Умножает learning_rate на 0.1 каждые 7 эпох
# (это одна из эвристик, не было на лекциях)
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
```

Обучение происходит здесь

Процесс необходимо остановить, как только val Loss перестанет убывать

```
    train_model(model, loss_fn, optimizer_ft, exp_lr_scheduler, num_epochs=25)
```

Сохраним модель

```
torch.save(model.state_dict(), './model.pth')

model.load_state_dict(torch.load('./model.pth'))
```

Предсказание на тестовой выборке

```
data_test = pd.read_csv('FINAL_TEST.csv')
data_test['class'] = 0
data_test.to_csv('./test_with_dummy_class.csv', index=False)
data_test

# Создадим массив из нулей для будущих предсказаний
predictions = np.zeros((762,))
index_to_write = 0

model.train(False)
for data in tqdm_notebook(TextClassDataLoader('./test_with_dummy_class.csv',
                                              word_to_index, 32, shuffle=False)):
    new_data = list(data)
    if USE_CUDA:
        for i in range(len(data)):
            new_data[i] = data[i].cuda()

    # получаем картинки и метки
    X_title, X_text, label, title_lengths, text_lengths, title_perm,
            text_perm = new_data

    # forward pass
    outputs = model(X_title, X_text, title_lengths, text_lengths, title_perm, text_perm)
    _, preds = torch.max(outputs.data, 1)

    predictions[index_to_write: min(index_to_write+32, 762)] = preds.cpu().numpy()
    index_to_write += 32

predictions = pd.DataFrame({'category': predictions})
predictions.category = predictions.category.map({i: class_name for i, class_name
    in enumerate(label_enc.classes_)})
predictions.to_csv('res.csv', encoding='utf-8', index=False)
```

Основы для кода RNN модуля и dataloader'а взяты из репозитория Pytorch-RNN-text-classification. (`https://github.com/keishinkickback/Pytorch-RNN-text-classification`)

Статья на русском про LSTM. (http://datareview.info/article/issleduem-lstm-seti-chast-1/)

Чтобы быстро учить нейросеть вы можете использовать Google Collab или, если у вас есть видеокарта с поддержкой CUDA, установить CUDA Toolkit и PyTorch с поддержкой CUDA на свой компьютер. Небольшое руководство (https://habr.com/post/348058/) по использованию GoogleCollab (в нем используется TensorFlow, но разница с PyTorch незначительна). Руководство по настройке CUDA (https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html)
на официальном сайте достаточно полное, но у многих возникают ошибки, решение которых придется искать самому.