



# Deep Learning School

Физтех-Школа Прикладной математики и информатики (ФПМИ)  
МФТИ

---

## ▼ Embeddings

Привет! В этом домашнем задании мы с помощью эмбедингов решим задачу семантической классификации твитов.

Для этого мы воспользуемся предобученными эмбедингами word2vec.

Для начала скачаем датасет для семантической классификации твитов:

```
1 !gdown https://drive.google.com/uc?id=1eE1FiUkXkcbw0McId4i7qY-L8hH-\_Qph&export=download
2 !unzip archive.zip
```

```
Downloading...
```

```
From: https://drive.google.com/uc?id=1eE1FiUkXkcbw0McId4i7qY-L8hH-\_Qph
```

```
To: /content/archive.zip
```

```
100% 84.9M/84.9M [00:00<00:00, 132MB/s]
```

```
Archive: archive.zip
```

```
replace training.1600000.processed.noemoticon.csv? [y]es, [n]o, [A]ll, [N]one, [r]en
```



Импортируем нужные библиотеки:

```
1 import math
2 import random
3 import string
4
5 import numpy as np
6 import pandas as pd
7 import seaborn as sns
8
9 import torch
10 import nltk
11 import gensim
12 import gensim.downloader as api
13
14 random.seed(42)
15 np.random.seed(42)
16 torch.random.manual_seed(42)
17 torch.cuda.random.manual_seed(42)
18 torch.cuda.random.manual_seed_all(42)
19
20 device = "cuda" if torch.cuda.is_available() else "cpu"

1 data = pd.read_csv("training.1600000.processed.noemoticon.csv", encoding="latin", head=
```

## ▼ Обработка данных

Double-click (or enter) to edit

Посмотрим на данные:

```
1 data.head()
```

	emotion	id	date	flag	user	text
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	<a href="http://twitpic.com/2y1zl">http://twitpic.com/2y1zl</a> - Awww, t... @switchfoot
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	matthaus	@Kenichan I dived many times for the

Выведем несколько примеров твитов, чтобы понимать, с чем мы имеем дело:

```

1 data["flag"].unique()

array(['NO_QUERY'], dtype=object)

1 data['emotion'].value_counts()

4      800000
0      800000
Name: emotion, dtype: int64

1 examples = data["text"].sample(10)
2 print("\n".join(examples))

@chrishasboobs AHHH I HOPE YOUR OK!!!
@misstoriblack cool , i have no tweet apps for my razr 2
@TiannaChaos i know just family drama. its lame.hey next time u hang out with kim n
School email won't open and I have geography stuff on there to revise! *Stupid Scho
upper airways problem
Going to miss Pastor's sermon on Faith...
on lunch....dj should come eat with me
@piginthepoke oh why are you feeling like that?
gahh noo!peyton needs to live!this is horrible
@mrstessyman thank you glad you like it! There is a product review bit on the site

```

Как видим, тексты твитов очень "грязные". Нужно предобработать датасет, прежде чем строить для него модель классификации.

Чтобы сравнивать различные методы обработки текста/модели/прочее, разделим датасет на dev(для обучения модели) и test(для получения качества модели).

```

1 print(data.shape[0])

1600000

1 indexes = np.arange(data.shape[0])
2 np.random.shuffle(indexes)
3 dev_size = math.ceil(data.shape[0] * 0.8)
4
5 dev_indexes = indexes[:dev_size]
6 test_indexes = indexes[dev_size:]
7
8 dev_data = data.iloc[dev_indexes]
9 test_data = data.iloc[test_indexes]
10
11 dev_data.reset_index(drop=True, inplace=True)
12 test_data.reset_index(drop=True, inplace=True)

```

## ▼ Обработка текста

Токенизируем текст, избавимся от знаков пунктуации и выкинем все слова, состоящие менее чем из 4 букв:

```
1 tokenizer = nltk.WordPunctTokenizer()
2 line = tokenizer.tokenize(dev_data["text"][0].lower())
3 print(" ".join(line))
```

```
@ claire_nelson i ' m on the north devon coast the next few weeks will be down in de
```



```
1 filtered_line = [w for w in line if all(c not in string.punctuation for c in w) and len(w) >= 4]
2 print(" ".join(filtered_line))
```

```
north devon coast next weeks will down devon again sometime hope though
```

Загрузим предобученную модель эмбедингов.

Нужно попробовать другую.


p.s. другую так и не попробовал, потому что решил все протестировать сначала с одной, а для другой времени не выделил:(

Полный список можно найти здесь: <https://github.com/RaRe-Technologies/gensim-data>.

Данная модель выдает эмбединги для **слов**.

```
1 word2vec = api.load("word2vec-google-news-300")
```

```
[=====] 100.0% 1662.8/1662.8MB download
```



```
1 emb_line = [word2vec.get_vector(w) for w in filtered_line if w in word2vec]
2 print(sum(emb_line).shape)
```

```
(300,)
```

Нормализуем эмбединги, прежде чем обучать на них сеть.

```
1 mean = np.mean(word2vec.vectors, 0)
2 std = np.std(word2vec.vectors, 0)
3 norm_emb_line = [(word2vec.get_vector(w) - mean) / std for w in filtered_line if w in word2vec]
4 print(sum(norm_emb_line).shape)
5 print([all(norm_emb_line[i] == emb_line[i]) for i in range(len(emb_line))])
```

```
(300,)
```

```
[False, False, False, False, False, False, False, False, False, False, False, False]
```



Сделаем датасет, который будет по запросу возвращать подготовленные данные.

```

1 from torch.utils.data import Dataset, random_split
2
3
4 class TwitterDataset(Dataset):
5     def __init__(self, data: pd.DataFrame, feature_column: str, target_column: str, wor
6         self.tokenizer = nltk.WordPunctTokenizer()
7
8         self.data = data
9
10        self.feature_column = feature_column
11        self.target_column = target_column
12
13        self.word2vec = word2vec
14
15        self.label2num = lambda label: 0 if label == 0 else 1
16        self.mean = np.mean(word2vec.vectors, axis=0)
17        self.std = np.std(word2vec.vectors, axis=0)
18
19    def __getitem__(self, item):
20        text = self.data[self.feature_column][item]
21        label = self.label2num(self.data[self.target_column][item])
22
23        tokens = self.get_tokens_(text)
24        embeddings = self.get_embeddings_(tokens)
25
26        return {"feature": embeddings, "target": label}
27
28    def get_tokens_(self, text):
29        # Получи все токены из текста и профильтруй их
30        tokens = self.tokenizer.tokenize(text.lower())
31        filtered_tokens = [t for t in tokens if all(c not in string.punctuation for c i
32        return filtered_tokens
33
34    def get_embeddings_(self, tokens):
35        embeddings = [(self.word2vec.get_vector(t) - self.mean) / self.std for t in tok
36
37        if len(embeddings) == 0:
38            embeddings = np.zeros((1, self.word2vec.vector_size))
39        else:
40            embeddings = np.array(embeddings)
41            if len(embeddings.shape) == 1:
42                embeddings = embeddings.reshape(-1, 1)
43
44        return embeddings
45
46    def __len__(self):
47        return self.data.shape[0]

```

```

1 dev = TwitterDataset(dev_data, "text", "emotion", word2vec)

```

Отлично, мы готовы с помощью эмбедингов слов превращать твиты в векторы и обучать нейронную сеть.

Преобразовывать твиты в векторы, используя эмбединги слов, можно несколькими способами. А именно такими:

## ▼ Average embedding (2 балла)

---

Это самый простой вариант, как получить вектор предложения, используя векторные представления слов в предложении. А именно: вектор предложения есть средний вектор всех слов в предложении (которые остались после токенизации и удаления коротких слов, конечно).

```
1 indexes = np.arange(len(dev))
2 np.random.shuffle(indexes)
3 example_indexes = indexes[::1000]
4 # Почему sum??????
5 # Если делать mean, то получается другая форма на графике
6 examples = {"features": [np.mean(dev[i]["feature"], axis=0) for i in example_indexes],
7              "targets": [dev[i]["target"] for i in example_indexes]}
8 print(len(examples["features"]))
```

1280

```
1 type(examples['features'])

list
```

Давайте сделаем визуализацию полученных векторов твитов тренировочного (dev) датасета. Так мы увидим, насколько хорошо твиты с разными target значениями отделяются друг от друга, т.е. насколько хорошо усреднение эмбедингов слов предложения передает информацию о предложении.

Для визуализации векторов надо получить их проекцию на плоскость. Сделаем это с помощью PCA. Если хотите, можете вместо PCA использовать TSNE: так у вас получится более точная проекция на плоскость (а значит, более информативная, т.е. отражающая реальное положение векторов твитов в пространстве). Но TSNE будет работать намного дольше.

```
1 from sklearn.decomposition import PCA
2
3
4 from sklearn.manifold import TSNE
5 examples["transformed_features"] = TSNE(n_components=2).fit_transform(X=examples['features'])
6
7 # pca = PCA(n_components=2)
8 # examples["transformed_features"] = pca.fit_transform(X=examples['features']) # Обучи
```

```

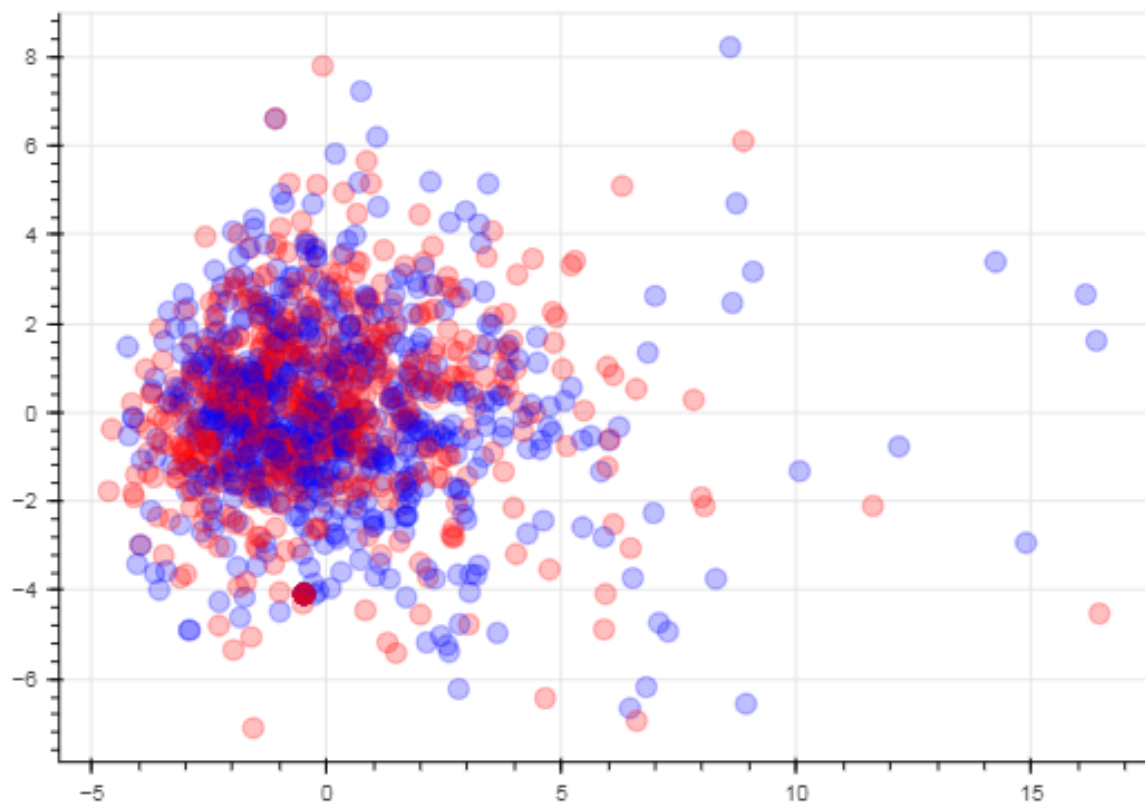
1 import bokeh.models as bm, bokeh.plotting as pl
2 from bokeh.io import output_notebook
3 output_notebook()
4
5 def draw_vectors(x, y, radius=10, alpha=0.25, color='blue',
6                 width=600, height=400, show=True, **kwargs):
7     """ draws an interactive plot for data points with auxiliary info on hover """
8     data_source = bm.ColumnDataSource({ 'x' : x, 'y' : y, 'color': color, **kwargs })
9
10    fig = pl.figure(active_scroll='wheel_zoom', width=width, height=height)
11    fig.scatter('x', 'y', size=radius, color='color', alpha=alpha, source=data_source)
12
13    fig.add_tools(bm.HoverTool(tooltips=[(key, "@" + key) for key in kwargs.keys()]))
14    if show: pl.show(fig)
15    return fig

```

```

1 # PCA
2 draw_vectors(
3     examples["transformed_features"][:, 0],
4     examples["transformed_features"][:, 1],
5     color=[["red", "blue"][t] for t in examples["targets"]]
6 )

```

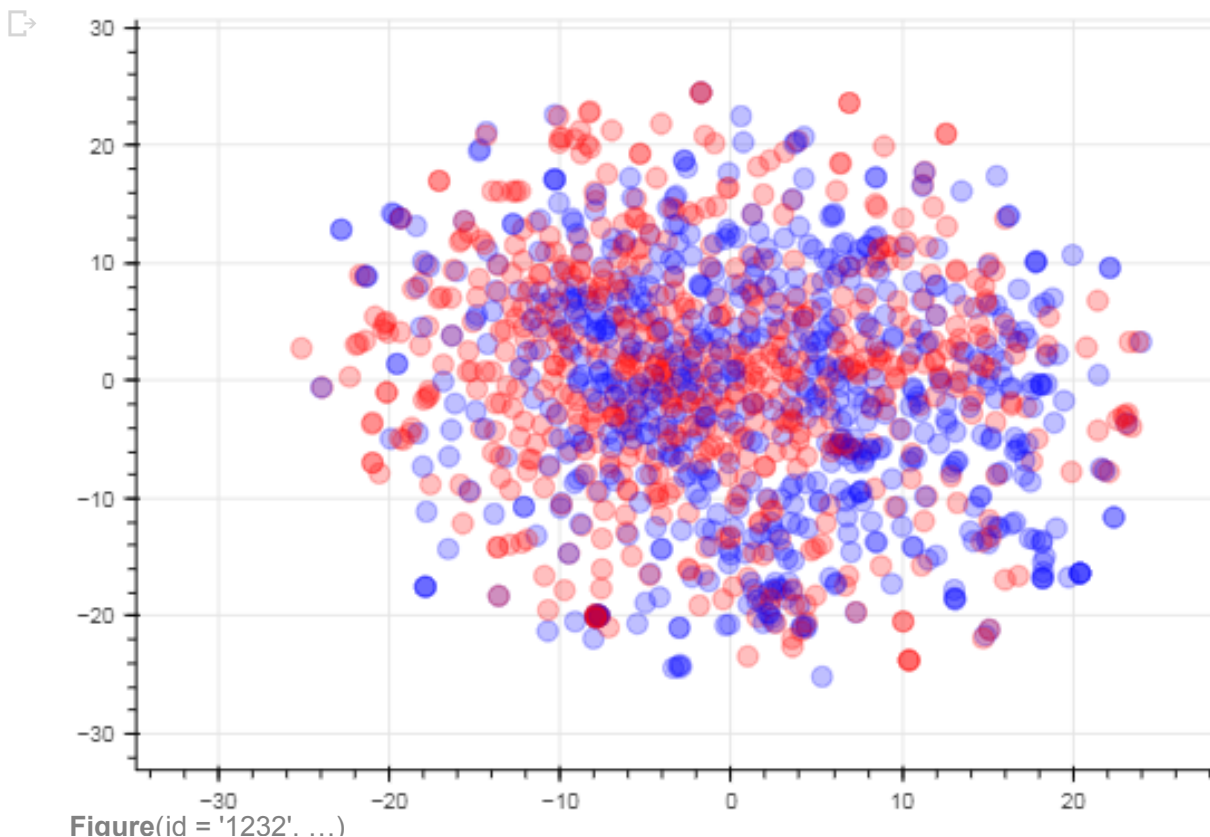


Figure(id = '1003', ...)

```

1 draw_vectors(
2     examples["transformed_features"][:, 0],
3     examples["transformed_features"][:, 1],
4     color=[["red", "blue"][t] for t in examples["targets"]]
5 )

```



Скорее всего, на визуализации нет четкого разделения твитов между классами. Это значит, что по полученным нами векторам твитов не так-то просто определить, к какому классу твит принадлежит. Значит, обычный линейный классификатор не очень хорошо справится с задачей. Надо будет делать глубокую (хотя бы два слоя) нейронную сеть.

Подготовим загрузчики данных. Усреднение векторов будем делать в "батчевалке" (`collate_fn`). Она используется для того, чтобы собирать из данных `torch.Tensor` батчи, которые можно отправлять в модель.

```
1 from torch.utils.data import DataLoader
2
3
4 batch_size = 1024
5 num_workers = 4
6
7 def average_emb(batch):
8     features = [np.mean(b["feature"], axis=0) for b in batch]
9     targets = [b["target"] for b in batch]
10
11     return {"features": torch.FloatTensor(features), "targets": torch.LongTensor(targets)}
12
13
14 train_size = math.ceil(len(dev) * 0.8)
15
16 train, valid = random_split(dev, [train_size, len(dev) - train_size])
17
18 train_loader = DataLoader(train, batch_size=batch_size, num_workers=num_workers, shuffle=True)
19 valid_loader = DataLoader(valid, batch_size=batch_size, num_workers=num_workers, shuffle=False)
```



```
cpuset_checked))
```

Определим функции для тренировки и теста модели:

```
1 from tqdm.notebook import tqdm
2
3
4 def training(model, optimizer, criterion, train_loader, epoch, device="cpu"):
5     pbar = tqdm(train_loader, desc=f"Epoch {e + 1}. Train Loss: {0}")
6     model.train()
7     for batch in pbar:
8         features = batch["features"].to(device)
9         targets = batch["targets"].to(device)
10
11         # Обнуляем градиенты
12         optimizer.zero_grad()
13
14         # Прямой ход
15         outputs = model(features)
16
17         # лосс
18         # targets = targets.unsqueeze(1)
19         # targets = targets.float()
20         loss = criterion(outputs, targets)
21         # обратный ход
22         loss.backward()
23         # обновление весов
24         optimizer.step()
25
26         pbar.set_description(f"Epoch {e + 1}. Train Loss: {loss:.4}")
27
28
29 def testing(model, criterion, test_loader, device="cpu"):
30     pbar = tqdm(test_loader, desc=f"Test Loss: {0}, Test Acc: {0}")
31     mean_loss = 0
32     mean_acc = 0
33     model.eval()
34     with torch.no_grad():
35         for batch in pbar:
36             features = batch["features"].to(device)
37             targets = batch["targets"].to(device)
38             # считаем выход модели
39             outputs = model(features)
40
41
42             loss = criterion(outputs, targets) # Посчитай лосс
43             # Получаем предсказания
44             _, preds = torch.max(outputs, 1)
45             acc = (preds == targets).to(torch.float).mean() # Посчитай точность модели
46
47             mean_loss += loss.item()
48             mean_acc += acc.item()
49
```

```

50         pbar.set_description(f"Test Loss: {loss:.4}, Test Acc: {acc:.4}")
51
52     pbar.set_description(f"Test Loss: {mean_loss / len(test_loader):.4}, Test Acc: {mean_acc:.4}")
53
54     return {"Test Loss": mean_loss / len(test_loader), "Test Acc": mean_acc / len(test_loader)}

```

Создадим модель, оптимизатор и целевую функцию. Вы можете сами выбрать количество слоев в нейронной сети, ваш любимый оптимизатор и целевую функцию.

```

1 import torch.nn as nn
2 from torch.optim import Adam
3
4
5 # Не забудь поиграться с параметрами ;)
6 vector_size = dev.word2vec.vector_size
7 # в случае BCELoss num_classes = 1
8 num_classes = 2
9 lr = 1e-2
10 num_epochs = 2
11 # изменение модели значимого качество не давало
12 model = nn.Sequential(
13     nn.Linear(vector_size, 300),
14     nn.ReLU(),
15
16     nn.Linear(300, 400),
17     nn.ReLU(),
18
19     nn.Linear(400, 100),
20     nn.ReLU(),
21
22     nn.Linear(100, num_classes),
23 ) # Твоя модель
24 model = model.cuda()
25 criterion = nn.CrossEntropyLoss() # Твой лосс
26 optimizer = torch.optim.Adam(model.parameters(), lr=lr) # Твой оптимайзер

```

Наконец, обучим модель и протестируем её.

После каждой эпохи будем проверять качество модели на валидационной части датасета. Если метрика стала лучше, будем сохранять модель. **Подумайте, какая метрика (точность или лосс) будет лучше работать в этой задаче?**

В данном случае можно использовать ассигасу, т.к. классы сбалансированны. Но, возможно я что-то не понимаю, мы когда используем другую метрику, мы так или иначе улучшаем точность. Да, там у каждой свои преимущества, но все же. Когда писал это, понял, что у нас всего два класса и можно использовать бинарную кросс энтропию в качестве лосса

```

1 best_metric = np.inf

```

```

2 for e in range(num_epochs):
3     training(model, optimizer, criterion, train_loader, e, device)
4     log = testing(model, criterion, valid_loader, device)
5     print(log)
6     if log["Test Loss"] < best_metric:
7         torch.save(model.state_dict(), "model.pt")
8         best_metric = log["Test Loss"]

Epoch 1. Train Loss: 0.5214:                                1000/1000 [03:02<00:00,
100%                                                         8.45it/s]
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarning:
  cpuset_checked))
Test Loss: 0.5051, Test Acc: 0.7471:                        250/250 [00:47<00:00,
100%                                                         7.66it/s]
{'Test Loss': 0.5193222169876098, 'Test Acc': 0.74221484375}
Epoch 2. Train Loss: 0.5265:                                1000/1000 [02:58<00:00,
100%                                                         6.34it/s]

1 test_loader = DataLoader(
2     TwitterDataset(test_data, "text", "emotion", word2vec),
3     batch_size=batch_size,
4     num_workers=num_workers,
5     shuffle=False,
6     drop_last=False,
7     collate_fn=average_emb)
8
9 model.load_state_dict(torch.load("model.pt", map_location=device))
10
11 print(testing(model, criterion, test_loader, device=device))

/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarning:
  cpuset_checked))
Test Loss: 0.5507, Test Acc: 0.7285:                        313/313 [00:58<00:00,
100%                                                         5.55it/s]
{'Test Loss': 0.5120721242298334, 'Test Acc': 0.7454822284345048}

```

Double-click (or enter) to edit

## Embeddings for unknown words (8 баллов)

Пока что использовалась не вся информация из текста. Часть информации фильтровалось – если слова не было в словаре эмбеддингов, то мы просто превращали слово в нулевой вектор. Хочется использовать информацию по-максимуму. Поэтому рассмотрим другие способы обработки слов, которых нет в словаре. А именно:

- Для каждого незнакомого слова будем запоминать его контекст(слова слева и справа от этого слова). Эмбеддингом нашего незнакомого слова будет сумма

эмбеддингов всех слов из его контекста. (4 балла)

- Для каждого слова текста получим его эмбеддинг из Tfidf с помощью TfidfVectorizer из [sklearn](https://scikit-learn.org/). Итоговым эмбеддингом для каждого слова будет сумма двух эмбеддингов: предобученного и Tfidf-ного. Для слов, которых нет в словаре предобученных эмбеддингов, результирующий эмбеддинг будет просто полученный из Tfidf. (4 балла)

Реализуйте оба варианта **ниже**. Напишите, какой способ сработал лучше и ваши мысли, почему так получилось.

## ▼ 1 Способ - сумма соседних эмбеддингов

Пробегаемся по всем токенам, если он не содержится в word2vec, то заменяем на , и добавляем нулевой embedding, чтобы размер списка embeddings был одинаковый с размером списка tokens

```
1 class TwitterDatasetContextEmbeddings(TwitterDataset):
2
3     def get_embeddings_(self, tokens):
4
5         embeddings = []
6         for i, t in enumerate(tokens):
7
8             # Если токен уже есть, то просто нормализуем и добавляем в список эмбеддинг
9             if t in self.word2vec:
10
11                 emb = (self.word2vec.get_vector(t) - self.mean) / self.std
12                 embeddings.append(emb)
13
14             # если же нет, то заменяем, вместо него нулевой вектор ставим
15             else:
16                 tokens[i] = '<UNK>'
17                 embeddings.append([0 for _ in range(self.word2vec.vector_size)])
18
19             # теперь заменяем на среднее соседей
20             for i, t in enumerate(tokens):
21
22                 if t == '<UNK>':
23                     if i == 0:
24                         try:
25                             emb = embeddings[i+1]
26                             # чтобы избежать случая только одного токена
27                         except IndexError:
28                             # если он один, то оставляем нулями
29                             emb = [0 for _ in range(self.word2vec.vector_size)]
30                     elif i == len(tokens)-1:
31                         # если последний, то приравниваем значение предыдущего
32                         emb = embeddings[i-1]
33                     else:
```

```

33         else:
34             # иначе считаем среднее
35             emb = np.mean([embeddings[i-1], embeddings[i+1]], axis=0)
36
37             # делаем нормализацию, добавляем значение
38             emb = (emb - self.mean) / self.std
39             embeddings[i] = emb
40         else:
41             pass
42
43     if len(embeddings) == 0:
44         embeddings = np.zeros((1, self.word2vec.vector_size))
45     else:
46         embeddings = np.array(embeddings)
47         if len(embeddings.shape) == 1:
48             embeddings = embeddings.reshape(-1, 1)
49
50     return embeddings

```

```

1 dev = TwitterDatasetContextEmbeddings(dev_data, "text", "emotion", word2vec)

```

```

1 from tqdm.notebook import tqdm
2
3
4 def training(model, optimizer, criterion, train_loader, epoch, device="cpu"):
5     pbar = tqdm(train_loader, desc=f"Epoch {e + 1}. Train Loss: {0}")
6     model.train()
7     for batch in pbar:
8         features = batch["features"].to(device)
9         targets = batch["targets"].to(device)
10
11         # Обнуляем градиенты
12         optimizer.zero_grad()
13
14         # Прямой ход
15         outputs = model(features)
16
17         # Получи предсказания модели
18         loss = criterion(outputs, targets) # Посчитай лосс
19         # Обнови параметры модели
20         loss.backward()
21         optimizer.step()
22
23     pbar.set_description(f"Epoch {e + 1}. Train Loss: {loss:.4}")
24
25
26 def testing(model, criterion, test_loader, device="cpu"):
27     pbar = tqdm(test_loader, desc=f"Test Loss: {0}, Test Acc: {0}")
28     mean_loss = 0
29     mean_acc = 0
30     model.eval()
31     with torch.no_grad():
32         for batch in pbar:
33             features = batch["features"].to(device)

```

```

34         targets = batch["targets"].to(device)
35
36         outputs = model(features)
37
38         # Получи предсказания модели
39         loss = criterion(outputs, targets) # Посчитай лосс
40
41         _, preds = torch.max(outputs, 1)
42         acc = (preds == targets).to(torch.float).mean() # Посчитай точность модели
43
44         mean_loss += loss.item()
45         mean_acc += acc.item()
46
47         pbar.set_description(f"Test Loss: {loss:.4}, Test Acc: {acc:.4}")
48
49     pbar.set_description(f"Test Loss: {mean_loss / len(test_loader):.4}, Test Acc: {mean_acc / len(test_loader):.4}")
50
51     return {"Test Loss": mean_loss / len(test_loader), "Test Acc": mean_acc / len(test_loader)}

```

```

1 # Не забудь поиграться с параметрами ;)
2 vector_size = dev.word2vec.vector_size
3 num_classes = 2
4 lr = 1e-2
5 num_epochs = 2
6
7 model = nn.Sequential(
8     nn.Linear(vector_size, 300),
9     nn.ReLU(),
10
11     nn.Linear(300, 400),
12     nn.ReLU(),
13
14     nn.Linear(400, 100),
15     nn.ReLU(),
16
17     nn.Linear(100, num_classes),
18 ) # Твоя модель
19 model = model.cuda()
20 criterion = nn.CrossEntropyLoss() # Твой лосс
21 optimizer = torch.optim.Adam(model.parameters(), lr=lr) # Твой оптимайзер

```

```

1 best_metric = np.inf
2 for e in range(num_epochs):
3     training(model, optimizer, criterion, train_loader, e, device)
4     log = testing(model, criterion, valid_loader, device)
5     print(log)
6     if log["Test Loss"] < best_metric:
7         torch.save(model.state_dict(), "model.pt")
8         best_metric = log["Test Loss"]

```

```
Epoch 1. Train Loss: 0.5101: 1000/1000 [02:59<00:00,
100% 6.77it/s]
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarni
cpuset_checked))
Test Loss: 0.4998, Test Acc: 0.7363: 250/250 [00:47<00:00,
100% 5.13it/s]
```

```
1 test_loader = DataLoader(
2     TwitterDatasetContextEmbeddings(test_data, "text", "emotion", word2vec),
3     batch_size=batch_size,
4     num_workers=num_workers,
5     shuffle=False,
6     drop_last=False,
7     collate_fn=average_emb)
8
9 model.load_state_dict(torch.load("model.pt", map_location=device))
10
11 print(testing(model, criterion, test_loader, device=device))

/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarni
cpuset_checked))
Test Loss: 0.81, Test Acc: 0.6992: 313/313 [01:06<00:00,
100% 7.24it/s]
{'Test Loss': 11535.61929712604, 'Test Acc': 0.7192429612619808}
```

## ▼ 2 способ - эмбединги из TFIDF

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2 from sklearn.utils.extmath import randomized_svd

1 def fun(doc):
2     return doc
3
4 tfidf = TfidfVectorizer(
5     analyzer='word',
6     tokenizer=fun,
7     preprocessor=fun,
8     token_pattern=None)

1 def my_get_tokens_(text, my_tokenizer=nlTK.WordPunctTokenizer()):
2     # Получи все токены из текста и профильтруй их
3     tokens = my_tokenizer.tokenize(text.lower())
4     filtered_tokens = [t for t in tokens if all(c not in string.punctuation for c in t)]
5     return filtered_tokens

1 examples = data["text"].sample(1000)
```

## 1 examples

```
541200          @chrishasboobs AHHH I HOPE YOUR OK!!!
750             @misstoriblack cool , i have no tweet apps fo...
766711          @TiannaChaos i know just family drama. its la...
285055          School email won't open and I have geography ...
705995                                     upper airways problem

                                     ...
338333          @girrlonthewing Ha, well you'd be surprised at...
109574          Some dark clouds in #indiavotes #indiavotes09 ...
1349309         @wolfgnards awesome. Thanks for letting me kno...
671510          I left my heart @holdenbeach Hoping to go bac...
385755                                     I wish I had the Sims 3
Name: text, Length: 1000, dtype: object
```

```
1 docs = [my_get_tokens_(text) for text in examples]
2 docs[::100]
```

```
[['chrishasboobs', 'ahhh', 'hope', 'your'],
 ['powersurf',
  'case',
  'designed',
  'someone',
  'idiot',
  'stole',
  'prove',
  'they',
  'want',
  'play',
  'hardball',
  'game'],
 ['missing'],
 ['laydeedelish', 'definatly', 'strong'],
 ['victoriastevens', 'yeah', 'started'],
 ['pollypocket3', 'hate'],
 ['finally',
  'that',
  'know',
  'gets',
  'gold',
  'logie',
  'disappointed',
  'wasn',
  'rove',
  'though'],
 ['just', 'stepped', 'chicharrone', 'crumbs'],
 ['congrats',
  'ashleytisdale',
  'miley Cyrus',
  'youll',
  'last',
  'song',
  'eventually',
  'girl',
  'dont',
  'worry'],
 ['someone',
  'needs',
  'bring',
```



```
'drugs',
'wish',
'have',
'sore',
'throat',
'longer',
'advil',
'tylenol',
'seem',
'work',
'anymore']]
```

```
1 tfidf_matrix = tfidf.fit_transform(docs)
2 print(tfidf.vocabulary_)
3
4 tfidf_matrix.todense().shape
```

```
{'chrishasboobs': 530, 'ahhh': 94, 'hope': 1290, 'your': 3176, 'misstoriblack': 1765
(1000, 3202)}
```



```
1 tfidf_matrix.todense()[0].shape

(1, 3202)
```

```
1 V, Sigma, UT = randomized_svd(tfidf_matrix,
2                               n_components=300,
3                               n_iter=5,
4                               random_state=None)
```

```
1 exampleword2tfidf = {}
2 for v in tfidf.vocabulary_:
3     exampleword2tfidf[v] = UT[:, tfidf.vocabulary_[v]]
```

```
1 exampleword2tfidf['make'].shape

(300,)
```

```
1 indexes = np.arange(data.shape[0])
2 np.random.shuffle(indexes)
3 dev_size = math.ceil(data.shape[0] * 0.8)
4
5 dev_indexes = indexes[:dev_size]
6 test_indexes = indexes[dev_size:]
7
8 dev_data = data.iloc[dev_indexes]
9 test_data = data.iloc[test_indexes]
10
11 dev_data.reset_index(drop=True, inplace=True)
12 test_data.reset_index(drop=True, inplace=True)
```

```
1 dev_examples = dev_data['text']
2 dev_docs = [mv.get_tokens(text) for text in dev_examples]
```

```
2 dev_docs = [my_get_tokens(text) for text in dev_examples]
```

```
1 # X = V Sigma U^T
2 UT = randomized_svd(tfidf.fit_transform(dev_docs),
3                     n_components=200,
4                     )[2]
```

```
1 dev_word2tfidf = {}
2 for v in tfidf.vocabulary_:
3     dev_word2tfidf[v] = UT[:, tfidf.vocabulary_[v]]
```

```
1 import pickle
2
3 f = open("dev_word2tfidf.pkl", "wb")
4 pickle.dump(dev_word2tfidf, f)
5 f.close()
```

```
1 len(dev_word2tfidf['make'])
```

```
200
```

```
1 test_examples = test_data['text']
2 test_docs = [my_get_tokens(text) for text in test_examples]
```

```
1 # X = V Sigma U^T
2
3 UT = randomized_svd(tfidf.transform(test_docs),
4                     n_components=200,
5                     )[2]
```

```
1 test_word2tfidf = {}
2 for v in tfidf.vocabulary_:
3     test_word2tfidf[v] = UT[:, tfidf.vocabulary_[v]]
```

```
1 import pickle
2
3 f = open("test_word2tfidf.pkl", "wb")
4 pickle.dump(test_word2tfidf, f)
5 f.close()
```

```
1 len(test_word2tfidf['make'])
```

```
200
```

```
1 # check: they should have different representation because the context and documents ir
2 # are different
3 any(test_word2tfidf['make'] == dev_word2tfidf['make'])
```

```
False
```

```

1 from torch.utils.data import Dataset, random_split
2
3
4 class TwitterDataset(Dataset):
5     def __init__(self, data: pd.DataFrame, feature_column: str, target_column: str, word2vec):
6         self.tokenizer = nltk.WordPunctTokenizer()
7
8         self.data = data
9
10        self.feature_column = feature_column
11        self.target_column = target_column
12
13        self.word2vec = word2vec
14
15        self.label2num = lambda label: 0 if label == 0 else 1
16        self.mean = np.mean(word2vec.vectors, axis=0)
17        self.std = np.std(word2vec.vectors, axis=0)
18
19    def __getitem__(self, item):
20        text = self.data[self.feature_column][item]
21        label = self.label2num(self.data[self.target_column][item])
22
23        tokens = self.get_tokens_(text)
24        embeddings = self.get_embeddings_(tokens)
25
26        return {"feature": embeddings, "target": label}
27
28    def get_tokens_(self, text):
29        # Получи все токены из текста и профильтруй их
30        tokens = self.tokenizer.tokenize(text.lower())
31        filtered_tokens = [t for t in tokens if all(c not in string.punctuation for c in t)]
32        return filtered_tokens
33
34    def get_embeddings_(self, tokens):
35        embeddings = [(self.word2vec.get_vector(t) - self.mean) / self.std for t in tokens]
36
37        if len(embeddings) == 0:
38            embeddings = np.zeros((1, self.word2vec.vector_size))
39        else:
40            embeddings = np.array(embeddings)
41            if len(embeddings.shape) == 1:
42                embeddings = embeddings.reshape(-1, 1)
43
44        return embeddings
45
46    def __len__(self):
47        return self.data.shape[0]
48
49
50 class TwitterDatasetTFIDFEmbeddings(TwitterDataset):
51     def __init__(self, data: pd.DataFrame, feature_column: str, target_column: str, word2tfidf):
52         super().__init__(data, feature_column, target_column, word2tfidf)
53         self.word2tfidf = word2tfidf

```

```

6
7
8 def get_embeddings_(self, tokens):
9     # Получи эмбединги слов и усредни их
10
11     embeddings_pretrained = np.zeros((len(tokens), self.word2vec.vector_size))
12     embeddings_tfidf = np.zeros((len(tokens), 200)) # 300 did not fit into RAM
13
14     for i, t in enumerate(tokens):
15
16         if t in self.word2tfidf:
17             embeddings_tfidf[i] = self.word2tfidf.get(t)
18         if t in self.word2vec:
19             embeddings_pretrained[i] = (self.word2vec.get_vector(t) - self.mean) /
20
21     embeddings = np.hstack([embeddings_pretrained, embeddings_tfidf])
22
23     assert embeddings.shape[1] == 500
24
25     if len(embeddings) == 0:
26         embeddings = np.zeros((1, 500))
27     else:
28         embeddings = np.array(embeddings)
29         if len(embeddings.shape) == 1:
30             embeddings = embeddings.reshape(-1, 1)
31
32     return embeddings

```

```

1 class TwitterDatasetTFIDFEmbeddings(TwitterDataset):
2     def __init__(self, data: pd.DataFrame, feature_column: str, target_column: str, wor
3         word2tfidf):
4         super().__init__(data, feature_column, target_column, word2vec)
5         self.word2tfidf = word2tfidf
6
7
8     def get_embeddings_(self, tokens):
9         # Получи эмбединги слов и усредни их
10
11         embeddings_pretrained = np.zeros((len(tokens), self.word2vec.vector_size))
12         embeddings_tfidf = np.zeros((len(tokens), 200)) # 300 did not fit into RAM
13
14         for i, t in enumerate(tokens):
15
16             if t in self.word2tfidf:
17                 embeddings_tfidf[i] = self.word2tfidf.get(t)
18             if t in self.word2vec:
19                 embeddings_pretrained[i] = (self.word2vec.get_vector(t) - self.mean) /
20
21         embeddings = np.hstack([embeddings_pretrained, embeddings_tfidf])
22
23         assert embeddings.shape[1] == 500
24
25         if len(embeddings) == 0:
26             embeddings = np.zeros((1, 500))
27

```

```

27         else:
28             embeddings = np.array(embeddings)
29             if len(embeddings.shape) == 1:
30                 embeddings = embeddings.reshape(-1, 1)
31
32     return embeddings

```

```

1 import pickle
2
3 with open('dev_word2tfidf.pkl', 'rb') as f:
4     word2tfidf = pickle.load(f)

```

```

1 word2vec = api.load("word2vec-google-news-300")

```

```

1 dev = TwitterDatasetTFIDFEmbeddings(
2     data=dev_data,
3     feature_column="text", target_column="emotion",
4     word2vec=word2vec,
5     word2tfidf=word2tfidf)

```

```

1 len(dev)

```

```

1 indexes = np.arange(len(dev))
2 np.random.shuffle(indexes)
3 example_indexes = indexes[:1000]
4
5 # changed np.sum to np.mean
6 examples = {"features": [np.mean(dev[i]["feature"], axis=0) for i in example_indexes],
7                     "targets": [dev[i]["target"] for i in example_indexes]}
8
9
10 print(len(examples["features"]), len(examples["features"][0]))

```

```

1280 500

```

```

1 from torch.utils.data import DataLoader
2
3 batch_size = 1024
4 num_workers = 4
5
6 def average_emb(batch):
7     features = [np.mean(b["feature"], axis=0) for b in batch]
8     targets = [b["target"] for b in batch]
9
10    return {"features": torch.FloatTensor(features), "targets": torch.LongTensor(targets)}
11
12
13 train_size = math.ceil(len(dev) * 0.8)
14
15 train, valid = random_split(dev, [train_size, len(dev) - train_size])
16
17 train_loader = DataLoader(train, batch_size=batch_size, num_workers=num_workers, shuffle=True)

```

```
17 train_loader = DataLoader(train, batch_size=batch_size, num_workers=num_workers, shuffle=True)
18 valid_loader = DataLoader(valid, batch_size=batch_size, num_workers=num_workers, shuffle=False)
```

/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarning: The following gradient is not backpropagated because torch.autograd.set\_detect\_anomaly was set to True and the operation was detected as non-differentiable (autograd will raise an error if detect\_anomaly was False):

cpuset\_checked))

```
1 import torch.nn as nn
2 from torch.optim import Adam
3
4
5 # TODO: try different parameters
6
7 # 300 from word2vec
8 # 200 from tfidf
9 vector_size = 500
10
11 num_classes = 2
12 lr = 1e-3
13 # после 3-х эпох переобучение начинается?
14 # построить графики
15 num_epochs = 3
16
17 # TODO: define the model, loss, and optimiser
18 model = nn.Sequential(
19     nn.Linear(vector_size, 200),
20     nn.ReLU(),
21
22     nn.Linear(200, 100),
23     nn.ReLU(),
24
25     nn.Linear(100, num_classes),
26 )
27 model = model.cuda()
28
29 criterion = nn.CrossEntropyLoss()
30 optimizer = torch.optim.Adam(model.parameters(), lr=lr)
```

```
1 from tqdm.notebook import tqdm
2
3
4 def training(model, optimizer, criterion, train_loader, epoch, device="cpu"):
5     pbar = tqdm(train_loader, desc=f"Epoch {e + 1}. Train Loss: {0}")
6     model.train()
7     for batch in pbar:
8         features = batch["features"].to(device)
9         targets = batch["targets"].to(device)
10
11         # Обнуляем градиенты
12         optimizer.zero_grad()
13
14         # Прямой ход
15         outputs = model(features)
16
17         # Получим предсказание модели
```

```

17         # Получи предсказания модели
18         loss = criterion(outputs, targets) # Посчитай лосс
19         # Обнови параметры модели
20         loss.backward()
21         optimizer.step()
22
23         pbar.set_description(f"Epoch {e + 1}. Train Loss: {loss:.4}")
24
25
26 def testing(model, criterion, test_loader, device="cpu"):
27     pbar = tqdm(test_loader, desc=f"Test Loss: {0}, Test Acc: {0}")
28     mean_loss = 0
29     mean_acc = 0
30     model.eval()
31     with torch.no_grad():
32         for batch in pbar:
33             features = batch["features"].to(device)
34             targets = batch["targets"].to(device)
35
36             outputs = model(features)
37
38             # Получи предсказания модели
39             loss = criterion(outputs, targets) # Посчитай лосс
40
41             _, preds = torch.max(outputs, 1)
42             acc = (preds == targets).to(torch.float).mean() # Посчитай точность модели
43
44             mean_loss += loss.item()
45             mean_acc += acc.item()
46
47             pbar.set_description(f"Test Loss: {loss:.4}, Test Acc: {acc:.4}")
48
49     pbar.set_description(f"Test Loss: {mean_loss / len(test_loader):.4}, Test Acc: {mean_acc / len(test_loader):.4}")
50
51     return {"Test Loss": mean_loss / len(test_loader), "Test Acc": mean_acc / len(test_loader)}

```

```

1 best_metric = np.inf
2 for e in range(num_epochs):
3     training(model, optimizer, criterion, train_loader, e, device)
4     log = testing(model, criterion, valid_loader, device)
5     print(log)
6     if log["Test Loss"] < best_metric:
7         torch.save(model.state_dict(), "model_tfidf.pt")
8         best_metric = log["Test Loss"]

```

```

Epoch 1. Train Loss: 0.4998: 1000/1000 [04:01<00:00,
100% 5.25it/s]
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarni
cpuset_checked))
Test Loss: 0.5144, Test Acc: 0.7373: 250/250 [01:03<00:00,
100% 5.98it/s]
{'Test Loss': 0.4990757074356079, 'Test Acc': 0.75308203125}
Epoch 2. Train Loss: 0.4941: 1000/1000 [04:13<00:00,
100% 5.04it/s]
Test Loss: 0.5055, Test Acc: 0.7402: 250/250 [01:05<00:00,
100% 5.86it/s]

1 import pickle
2 from torch.utils.data import DataLoader
3
4
5 # rewrite word2tfidf to save RAM
6 # with open('test_word2tfidf.pkl', 'rb') as f:
7 #     word2tfidf = pickle.load(f)
8
9 # create test loader
10 # note, we create the dataset inside
11 test_loader = DataLoader(
12     TwitterDatasetTFIDFEmbeddings(test_data, "text", "emotion", word2vec,
13                                     word2tfidf=word2tfidf),
14     batch_size=batch_size,
15     num_workers=num_workers,
16     shuffle=False,
17     drop_last=False,
18     collate_fn=average_emb)
19
20 model.load_state_dict(torch.load("model_tfidf.pt", map_location=device))
21
22 print(testing(model, criterion, test_loader, device=device))

/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarni
cpuset_checked))
Test Loss: 0.4461, Test Acc: 0.7812: 313/313 [01:22<00:00,
100% 5.00it/s]
{'Test Loss': 0.4854061859674728, 'Test Acc': 0.7633661142172524}

```

Вывод:

Метод	score
default	0.7454822284345048
Модель с контекстным эмбедингом	0.7115927016773163
Модель с TFidf	0.763366114217252



Модель, в которой мы для неизвестных эмбедингов работает хуже всех. Мне кажется, это происходит из-за случаев, когда рядом стоят несколько неизвестных токенов. Скорее всего, нужно было усреднять не по соседним, а по целому предложению, либо по окну.

Модель с TFidf показала лучшее качество. В принципе, это логично - мы использовали для каждого слова его эмбединг из word2vec + информацию о вероятности встретить его в конкретно наших текстах.