Национальная Академия Наук

Кыргызской Республики

Институт Автоматики и Информационных Технологий

Лаборатория ИИС

**Листинг программных модулей системы анализа тональности текстов на основе LSTM-классификатора текстов SentiMe,**

**ПС SentiMe**

Автор:

м.н.с. Сороковая А.В.

Бишкек 2019

**Оглавление**

[**1. Листинг программного модуля библиотеки анализа тональности текстов на основе LSTM-классификатора текстов** 4](#_Toc16028575)

[1.1. Пакет corpus 4](#_Toc16028576)

[1.1.1. Модуль imdb.py 4](#_Toc16028577)

[1.2. Пакет dataio 4](#_Toc16028578)

[1.2.1. Модуль csv.py 4](#_Toc16028579)

[1.2.2. Модуль database.py 6](#_Toc16028580)

[1.2.3. Модуль result.py 6](#_Toc16028581)

[1.3. Пакет embedding\_methods\_plugins 6](#_Toc16028582)

[1.3.1. Модуль base\_embedding.py 6](#_Toc16028583)

[1.3.2. Модуль glove\_embedding.py 7](#_Toc16028584)

[1.3.3. Модуль lsa\_embedding.py 8](#_Toc16028585)

[1.3.4. Модуль word2vec\_cbow\_embedding.py 9](#_Toc16028586)

[1.3.5. Модуль word2vec\_skipgram\_embedding.py 9](#_Toc16028587)

[1.4. Пакет helpers 10](#_Toc16028588)

[1.4.1. Модуль helpers.py 10](#_Toc16028589)

[1.5. Пакет netword\_metrics 12](#_Toc16028590)

[1.5.1. Модуль confusion\_matrix.py 12](#_Toc16028591)

[1.6. Пакет network 12](#_Toc16028592)

[1.6.1. Модуль logging\_callback.py 12](#_Toc16028593)

[1.6.2. Модуль validation\_callback.py 13](#_Toc16028594)

[1.6.3. Модуль lstm\_categorical.py 13](#_Toc16028595)

[1.7. Пакет test\_rus\_comments 15](#_Toc16028596)

[1.7.1. Модуль step\_1\_restore\_tokens.py 15](#_Toc16028597)

[1.7.2. Модуль step\_2\_do\_experiments.py 15](#_Toc16028598)

[1.7.3. Модуль step\_3\_analysis.py 15](#_Toc16028599)

[1.8. Модуль tokenizing 15](#_Toc16028600)

[1.9. Модуль visualization 16](#_Toc16028601)

[**2. Листинг программного модуля web-интерфейса системы анализа тональности текстов на основе LSTM-классификатора текстов** 17](#_Toc16028602)

[2.1. Пакет alembic 17](#_Toc16028603)

[2.1.1. Модуль env.py 17](#_Toc16028604)

[2.1.2. Модуль script.py.mako 17](#_Toc16028605)

[2.2. Пакет validation 17](#_Toc16028606)

[2.2.1. Модуль error\_codes.py 17](#_Toc16028607)

[2.2.2. Модуль errorinfo.py 18](#_Toc16028608)

[2.2.3. Модуль validators.py 19](#_Toc16028609)

[2.3. Пакет models 23](#_Toc16028610)

[2.3.1. Модуль base\_model.py 23](#_Toc16028611)

[2.3.2. Модуль enums.py 29](#_Toc16028612)

[2.3.3. Модуль file.py 30](#_Toc16028613)

[2.3.4. Модуль vocabulary.py 31](#_Toc16028614)

[2.3.5. Модуль word.py 32](#_Toc16028615)

[2.3.6. Модуль embedding\_method.py 33](#_Toc16028616)

[2.3.7. Модуль embedding\_method\_param.py 33](#_Toc16028617)

[2.3.8. Модуль embedding\_model.py 34](#_Toc16028618)

[2.3.9. Модуль embedding\_method\_param\_value.py 35](#_Toc16028619)

[2.3.10. Модуль model\_word\_vector.py 36](#_Toc16028620)

[2.3.11. Модуль net\_module.py 37](#_Toc16028621)

[2.3.12. Модуль net\_module\_param.py 38](#_Toc16028622)

[2.3.13. Модуль net.py 39](#_Toc16028623)

[2.3.14. Модуль net\_module\_param\_value.py 40](#_Toc16028624)

[2.3.15. Модуль net\_epoch.py 40](#_Toc16028625)

[2.3.16. Модуль epoch\_confusion\_matrix.py 41](#_Toc16028626)

[2.3.17. Модуль link.py 42](#_Toc16028627)

[2.3.18. Модуль text\_set.py 43](#_Toc16028628)

[2.3.19. Модуль text.py 43](#_Toc16028629)

[2.3.20. Модуль text\_set\_file.py 45](#_Toc16028630)

[2.3.21. Модуль text\_set\_link.py 45](#_Toc16028631)

[2.3.22. Модуль assotiations.py 46](#_Toc16028632)

[2.4. Пакет logic 46](#_Toc16028633)

[2.4.1. Модуль file.py 46](#_Toc16028634)

[2.4.2. Модуль vocabulary.py 47](#_Toc16028635)

[2.4.3. Модуль embedding\_model.py 50](#_Toc16028636)

[2.4.4. Модуль net.py 53](#_Toc16028637)

[2.4.5. Модуль text\_set.py 56](#_Toc16028638)

[2.5. Пакет controllers 59](#_Toc16028639)

[2.5.1. Модуль enum\_data.py 59](#_Toc16028640)

[2.5.2. Модуль file.py 59](#_Toc16028641)

[2.5.3. Модуль vocabulary.py 60](#_Toc16028642)

[2.5.4. Модуль embedding\_method.py 61](#_Toc16028643)

[2.5.5. Модуль embedding\_model.py 62](#_Toc16028644)

[2.5.6. Модуль net.py 63](#_Toc16028645)

[2.5.7. Модуль text\_set.py 67](#_Toc16028646)

# **1. Листинг программного модуля библиотеки анализа тональности текстов на основе LSTM-классификатора текстов**

## 1.1. Пакет corpus

### 1.1.1. Модуль imdb.py

Не депонируется

## 1.2. Пакет dataio

### 1.2.1. Модуль csv.py

import numpy as np

def read\_texts\_from\_file(file, quoted\_fields=True, separator=",", text\_col=1):

with open(file, "r", encoding="utf-8", errors="ignore") as filestream:

return read\_texts\_from\_filestream(filestream, quoted\_fields, separator, text\_col);

def read\_texts\_from\_filestream(file, quoted\_fields=True, separator=",", text\_col=1):

texts = []

for line in file:

new\_line = line[1:-2] if quoted\_fields else line

values = new\_line.split('"' + separator + '"') if quoted\_fields else new\_line.split(separator)

text = values[text\_col-1]

texts.append(text)

return texts

def read\_samples(filename, delimeter, tag\_col, neg\_tag, pos\_tag, text\_col, pos\_count=20000, neg\_count=20000):

pos\_samples = []

neg\_samples = []

with open(filename, 'r', encoding='UTF-8', errors='ignore') as f:

i = 0

try:

for line in f:

if len(pos\_samples) >= pos\_count and len(neg\_samples) >= neg\_count:

break

values = line[1:-2].split('"' + delimeter + '"')

tag = {

neg\_tag: -1,

pos\_tag: 1

}[values[tag\_col]] if values[tag\_col] == neg\_tag or values[tag\_col] == pos\_tag else None

if tag is None: continue

text = values[text\_col]

if tag == -1 and len(neg\_samples) < neg\_count:

neg\_samples.append((text, tag))

if tag == 1 and len(pos\_samples) < pos\_count:

pos\_samples.append((text, tag))

i = i + 1

except:

a = 1

return neg\_samples, pos\_samples

def save\_embedding\_matrix(file\_name, embedding\_matrix, delimeter='\t'):

with open(file\_name, mode='w', encoding='utf8') as f:

for i, vector in enumerate(embedding\_matrix):

f.write(str(i) + delimeter + delimeter.join([str(val) for val in vector]) + '\n')

def read\_embedding\_matrix(file\_name, delimeter='\t'):

embedding\_matrix = []

with open(file\_name, mode='r', encoding='utf8') as f:

for line in f:

values = line.split(delimeter)

i = int(values[0])

coefs = np.asarray(values[1:], dtype='float32')

embedding\_matrix.append(coefs)

return np.asarray(embedding\_matrix), len(coefs)

def save\_embedding\_dictionary(file\_name, embedding\_dictionary, delimeter='\t'):

with open(file\_name, mode='w', encoding='utf8') as f:

for word, vector in embedding\_dictionary.items():

f.write(word + delimeter + delimeter.join([str(val) for val in vector]) + '\n')

def read\_embedding\_dictionary(file\_name, delimeter='\t'):

embedding\_dictionary = {}

with open(file\_name, mode='r', encoding='utf8') as f:

for line in f:

values = line.split(delimeter)

word = values[0]

coefs = np.asarray(values[1:], dtype='float32')

embedding\_dictionary[word] = coefs

return embedding\_dictionary, len(coefs)

def save\_word\_to\_index(file\_name, word\_to\_index, delimeter='\t'):

with open(file\_name, mode='w', encoding='utf8') as f:

for word, i in word\_to\_index.items():

f.write(str(i) + delimeter + word + '\n')

def read\_word\_to\_index(file\_name, delimeter='\t'):

word\_to\_index = {}

with open(file\_name, mode='r', encoding='utf8') as f:

for line in f:

values = line.split(delimeter)

i = int(values[0])

word = values[1][:-1]

word\_to\_index[word] = i

return word\_to\_index

### 1.2.2. Модуль database.py

Не депонируется

### 1.2.3. Модуль result.py

Не депонируется

## 1.3. Пакет embedding\_methods\_plugins

### 1.3.1. Модуль base\_embedding.py

import numpy as np

class BaseEmbedding:

pad\_word = "<<PAD>>"

unknown\_word = "<<UNKNOWN>>"

def \_\_init\_\_(self, embedding\_dictionary, embedding\_dim, word\_to\_index):

self.name = 'base'

self.embedding\_dim = embedding\_dim

self.embedding\_dictionary = embedding\_dictionary

self.word\_to\_index = word\_to\_index

self.complete\_dictionary\_with\_pad()

self.complete\_dictionary\_with\_unknown()

self.embedding\_matrix = self.get\_embedding\_matrix()

def get\_word\_dictionary(self):

word\_dictionary = {}

for word, index in self.word\_to\_index.items():

word\_dictionary[word] = (index, self.embedding\_matrix[index])

return word\_dictionary

def complete\_dictionary\_with\_pad(self, pad\_index=0):

if self.pad\_word in self.word\_to\_index.keys():

old\_pad\_index = self.word\_to\_index[self.pad\_word]

else:

old\_pad\_index = sorted(self.word\_to\_index.values())[-1] + 1

if old\_pad\_index > pad\_index:

for word, index in self.word\_to\_index.items():

if pad\_index <= index < old\_pad\_index:

self.word\_to\_index[word] = index + 1

elif old\_pad\_index > pad\_index:

for word, index in self.word\_to\_index.items():

if old\_pad\_index < index <= pad\_index:

self.word\_to\_index[word] = index - 1

self.word\_to\_index[self.pad\_word] = pad\_index

def complete\_dictionary\_with\_unknown(self, unknown\_index=1):

if self.unknown\_word in self.word\_to\_index.keys():

old\_unknown\_index = self.word\_to\_index[self.unknown\_word]

else:

old\_unknown\_index = sorted(self.word\_to\_index.values())[-1] + 1

if old\_unknown\_index > unknown\_index:

for word, index in self.word\_to\_index.items():

if unknown\_index <= index < old\_unknown\_index:

self.word\_to\_index[word] = index + 1

elif old\_unknown\_index > unknown\_index:

for word, index in self.word\_to\_index.items():

if old\_unknown\_index < index <= unknown\_index:

self.word\_to\_index[word] = index - 1

self.word\_to\_index[self.unknown\_word] = unknown\_index

def get\_embedding\_matrix(self):

embedding\_dictionary = self.embedding\_dictionary

word\_to\_index = self.word\_to\_index

word\_num = len(word\_to\_index)

embedding\_dim = self.embedding\_dim

embedding\_matrix = np.zeros((word\_num, embedding\_dim))

for word, i in word\_to\_index.items():

embedding\_vector = embedding\_dictionary.get(word, None)

if embedding\_vector is not None:

embedding\_matrix[i] = embedding\_vector

return embedding\_matrix

@staticmethod

def get\_from\_data(embedding\_dictionary, embedding\_dim, word\_to\_index, self=None):

if self is None:

return BaseEmbedding(embedding\_dictionary, embedding\_dim, word\_to\_index)

else:

BaseEmbedding.\_\_init\_\_(self, embedding\_dictionary, embedding\_dim, word\_to\_index)

### 1.3.2. Модуль glove\_embedding.py

import glove

from time import time

from embedding\_methods\_plugins.base\_embedding import BaseEmbedding

class EmbeddingModel(BaseEmbedding):

def \_\_init\_\_(self, docs\_tokens, emb\_dim, iters, window, learn\_rate):

self.time = 0.

self.time = time()

corpus\_model = glove.Corpus()

corpus\_model.fit(docs\_tokens, window=window)

glove\_model = glove.Glove(no\_components=emb\_dim, learning\_rate=learn\_rate)

glove\_model.fit(corpus\_model.matrix, epochs=iters, no\_threads=4)

glove\_model.add\_dictionary(corpus\_model.dictionary)

self.time = time() - self.time

word\_to\_index = glove\_model.dictionary

index\_word = glove\_model.inverse\_dictionary

embedding\_dictionary = {index\_word[i]: vector for i, vector in enumerate(glove\_model.word\_vectors)}

super(EmbeddingModel, self).get\_from\_data(embedding\_dictionary, emb\_dim, word\_to\_index, self)

self.name = 'glove'

### 1.3.3. Модуль lsa\_embedding.py

from gensim.corpora import Dictionary

from gensim.models import LsiModel, TfidfModel

from time import time

from embedding\_methods\_plugins.base\_embedding import BaseEmbedding

class EmbeddingModel(BaseEmbedding):

def \_\_init\_\_(self, docs\_tokens, emb\_dim, iters):

self.time = 0.

self.time = time()

word\_dictionary = Dictionary(docs\_tokens)

word\_to\_index = word\_dictionary.token2id

docs\_term\_matrix = [word\_dictionary.doc2bow(tokens) for tokens in docs\_tokens]

tfidfmodel = TfidfModel(docs\_term\_matrix, id2word=word\_dictionary)

corpus = [tfidfmodel[doc] for doc in docs\_term\_matrix]

lsamodel = LsiModel(corpus, num\_topics=emb\_dim, id2word=word\_dictionary, power\_iters=iters)

self.time = time() - self.time

embedding\_matrix = lsamodel.get\_topics().transpose()

embedding\_dictionary = {}

embedding\_dim = None

for word, i in word\_to\_index.items():

embedding\_dictionary[word] = embedding\_matrix[i]

if embedding\_dim is None:

embedding\_dim = len(embedding\_matrix[i])

super(EmbeddingModel, self).get\_from\_data(embedding\_dictionary, embedding\_dim, word\_to\_index, self)

self.name = 'lsa'

### 1.3.4. Модуль word2vec\_cbow\_embedding.py

from gensim.models import Word2Vec

from time import time

from embedding\_methods\_plugins.base\_embedding import BaseEmbedding

class EmbeddingModel(BaseEmbedding):

def \_\_init\_\_(self, docs\_tokens, emb\_dim, iters, window):

self.time = 0.

self.time = time()

word\_model = Word2Vec(docs\_tokens, size=emb\_dim, window=window, iter=iters, sg=0, cbow\_mean=1)

self.time = time() - self.time

word\_to\_index = {word: i for i, word in enumerate(word\_model.wv.index2word)}

embedding\_dictionary = {word: word\_model.wv.get\_vector(word) for word, i in word\_to\_index.items()}

super(EmbeddingModel, self).get\_from\_data(embedding\_dictionary, emb\_dim, word\_to\_index, self)

self.name = 'w2v\_cbow'

### 1.3.5. Модуль word2vec\_skipgram\_embedding.py

from gensim.models import Word2Vec

from time import time

from embedding\_methods\_plugins.base\_embedding import BaseEmbedding

class EmbeddingModel(BaseEmbedding):

def \_\_init\_\_(self, docs\_tokens, emb\_dim, iters, window):

self.time = 0.

self.time = time()

word\_model = Word2Vec(docs\_tokens, size=emb\_dim, window=window, iter=iters, sg=1, cbow\_mean=0)

self.time = time() - self.time

word\_to\_index = {word: i for i, word in enumerate(word\_model.wv.index2word)}

embedding\_dictionary = {word: word\_model.wv.get\_vector(word) for word, i in word\_to\_index.items()}

super(EmbeddingModel, self).get\_from\_data(embedding\_dictionary, emb\_dim, word\_to\_index, self)

self.name = 'w2v\_skipgram'

## 1.4. Пакет helpers

### 1.4.1. Модуль helpers.py

from random import shuffle

import numpy as np

def min\_max\_norm(x):

min\_x = min(x)

max\_x = max(x)

return [(xi-min\_x)/(max\_x-min\_x) for xi in x]

def split\_by\_groups(tagged\_list, get\_tag\_function, get\_item\_function=None):

grouped = {}

for item in tagged\_list:

item\_tag = get\_tag\_function(item)

item\_item = get\_item\_function(item)

if item\_tag not in grouped:

grouped[item\_tag] = [item\_item]

else:

grouped[item\_tag].append(item\_item)

return grouped

def shuffle\_and\_get\_parts(samples, tags, parts=None):

samples\_tags = [(sample, tag) for sample, tag in zip(samples, tags)]

shuffle(samples\_tags)

# half of pos and neg

if parts is None:

parts = [int(len(samples)/2)]

else:

parts = [int(part/2) for part in parts]

# for each part neg and pos sets with samples and tags

parts\_sets = [[[[], []], [[], []]] for part in parts]

neg\_part\_i = 0

pos\_part\_i = 0

for i, (sample, tag) in enumerate(samples\_tags):

if tag == 0 and neg\_part\_i < len(parts):

if len(parts\_sets[neg\_part\_i][0][0]) >= parts[neg\_part\_i]:

neg\_part\_i += 1

if neg\_part\_i >= len(parts):

continue

parts\_sets[neg\_part\_i][0][0].append(sample)

parts\_sets[neg\_part\_i][0][1].append(tag)

elif tag == 1 and pos\_part\_i < len(parts):

if len(parts\_sets[pos\_part\_i][1][0]) >= parts[pos\_part\_i]:

pos\_part\_i += 1

if pos\_part\_i >= len(parts):

continue

parts\_sets[pos\_part\_i][1][0].append(sample)

parts\_sets[pos\_part\_i][1][1].append(tag)

for i in range(0, len(parts\_sets)):

part\_set = parts\_sets[i]

min\_count = min(len(part\_set[0][0]), len(part\_set[1][0]))

tmp = (

part\_set[0][0][0:min\_count] + part\_set[1][0][0:min\_count],

part\_set[0][1][0:min\_count] + part\_set[1][1][0:min\_count]

)

samples\_tags = [(sample, tag) for sample, tag in zip(tmp[0], tmp[1])]

shuffle(samples\_tags)

parts\_sets[i] = [[sample for sample, tag in samples\_tags], [tag for sample, tag in samples\_tags]]

return parts\_sets

def get\_embedding\_matrix\_by\_dictionary(embedding\_dictionary, word\_to\_index):

embedding\_dim = len(embedding\_dictionary.values()[0])

embedding\_matrix = np.zeros((len(word\_to\_index) + 1, embedding\_dim))

for word, i in word\_to\_index.items():

embedding\_vector = embedding\_dictionary.get(word, None)

if embedding\_vector is not None:

# words not found in embedding index will be all-zeros.

embedding\_matrix[i] = embedding\_vector

return embedding\_matrix

def get\_emb\_dic\_filename(emb\_name, sample\_num, word\_len, emb\_epoch):

return '%s\_dic\_samples%s\_dim%s\_epo%s.txt' % (emb\_name, sample\_num, word\_len, emb\_epoch)

def get\_emb\_w2i\_filename(emb\_name, sample\_num, word\_len, emb\_epoch):

return '%s\_w2i\_samples%s\_dim%s\_epo%s.txt' % (emb\_name, sample\_num, word\_len, emb\_epoch)

## 1.5. Пакет netword\_metrics

### 1.5.1. Модуль confusion\_matrix.py

def get\_confusion\_matrix(true\_tags, predicted\_tags):

tags = set(true\_tags + predicted\_tags)

confusion\_matrix = {true\_tag: {pred\_tag: 0 for pred\_tag in tags} for true\_tag in tags }

for true\_tag, pred\_tag in zip(true\_tags, predicted\_tags):

confusion\_matrix[true\_tag][pred\_tag] += 1

return confusion\_matrix

def get\_class\_recalls(confusion\_matrix):

class\_recalls = {}

for true\_tag, predictions in confusion\_matrix.items():

class\_recalls[true\_tag] = predictions[true\_tag] / sum(predictions.values())

return class\_recalls

def get\_class\_precisions(confusion\_matrix):

class\_sum = {tag: 0 for tag, predictions in confusion\_matrix.items()}

for true\_tag, predictions in confusion\_matrix.items():

for pred\_tag, count in predictions.items():

class\_sum[pred\_tag] += count

class\_precisions = {}

for tag, count in class\_sum.items():

class\_precisions[tag] = confusion\_matrix[tag][tag] / count

return class\_precisions

def get\_average\_recall(confusion\_matrix):

recalls = get\_class\_recalls(confusion\_matrix)

return sum(recalls.values()) / len(recalls)

def get\_average\_precision(confusion\_matrix):

precisions = get\_class\_precisions(confusion\_matrix)

return sum(precisions.values()) / len(precisions)

## 1.6. Пакет network

### 1.6.1. Модуль logging\_callback.py

from keras.callbacks import Callback

import time

from datetime import datetime

class LoggingCallback(Callback):

def \_\_init\_\_(self):

Callback.\_\_init\_\_(self)

self.epoch\_acc\_dic = {}

self.epoch\_loss\_dic = {}

self.epoch\_val\_acc\_dic = {}

self.epoch\_val\_loss\_dic = {}

self.epoch\_time\_dic = {}

self.starttime = None

def on\_train\_begin(self, logs=None):

self.starttime = datetime.now()

def on\_epoch\_begin(self, epoch, logs=None):

self.epoch\_time\_dic[epoch] = time.time()

def on\_epoch\_end(self, epoch, logs={}):

self.epoch\_acc\_dic[epoch] = logs["acc"]

self.epoch\_loss\_dic[epoch] = logs["loss"]

self.epoch\_val\_acc\_dic[epoch] = logs["val\_acc"]

self.epoch\_val\_loss\_dic[epoch] = logs["val\_loss"]

self.epoch\_time\_dic[epoch] = int((time.time() - self.epoch\_time\_dic[epoch])\*1000000)

def save(self, filename):

with open(filename, mode='w', encoding='utf8') as file:

for epoch, logs in self.epoch\_log\_dic.items():

msg = "Epoch: %i, time: %ss, %s" % (epoch, self.epoch\_time\_dic[epoch] , ", ".join("%s: %f" % (k, v) for k, v in logs.items()))

file.write(msg + '\n')

### 1.6.2. Модуль validation\_callback.py

from keras.callbacks import Callback

class ValidationCallback(Callback):

def \_\_init\_\_(self):

super(ValidationCallback, self).\_\_init\_\_()

self.epoch\_targets = {}

self.epoch\_outputs = {}

def on\_epoch\_end(self, epoch, logs={}):

self.epoch\_targets[epoch] = self.validation\_data[1].argmax(axis=-1)

self.epoch\_outputs[epoch] = self.model.predict\_classes(self.validation\_data[0], verbose=0)

return

### 1.6.3. Модуль lstm\_categorical.py

from keras import Sequential

from keras.backend import clear\_session

from keras.layers import Embedding, LSTM, Dense, Dropout, Conv1D, MaxPooling1D

from keras.utils.np\_utils import to\_categorical

from network.logging\_callback import LoggingCallback

import numpy as np

from network.validation\_callback import ValidationCallback

class LSTMCategorical:

def \_\_init\_\_(self, lstm\_out, doc\_len, dropout, batch\_size, epoch\_num, word\_num=None, word\_len=None, embedding\_matrix=None):

clear\_session()

if embedding\_matrix is not None:

word\_num = embedding\_matrix.shape[0]

word\_len = embedding\_matrix.shape[1]

model = Sequential()

if embedding\_matrix is not None:

model.add(Embedding(word\_num, word\_len, weights=[embedding\_matrix], trainable=False, input\_length=doc\_len))

else:

model.add(Embedding(word\_num, word\_len, input\_length=doc\_len))

self.dropout = dropout/100

model.add(Dropout(self.dropout))

model.add(Conv1D(64, 5, activation='relu'))

model.add(MaxPooling1D(pool\_size=4))

model.add(LSTM(lstm\_out))

categories\_num = 2

model.add(Dense(categories\_num, activation='sigmoid'))

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

self.model = model

self.word\_num = word\_num

self.word\_len = word\_len

self.doc\_len = doc\_len

self.lstm\_out = lstm\_out

self.batch\_size = batch\_size

self.epoch\_num = epoch\_num

self.history = None

self.test\_count = 0

self.train\_count = 0

self.class\_tag = {}

self.name = 'lstm\_conv'

self.categories\_num = categories\_num

self.epoch\_logger = LoggingCallback()

self.valid\_callback = ValidationCallback()

def get\_epoch\_predictions(self):

epoch\_target\_tags = \

{e: [self.class\_tag[cls] for cls in clses] for e, clses in self.valid\_callback.epoch\_targets.items()}

epoch\_predicted\_tags = \

{e: [self.class\_tag[cls] for cls in clses] for e, clses in self.valid\_callback.epoch\_outputs.items()}

return epoch\_target\_tags, epoch\_predicted\_tags

def train\_by\_sequences(self, train\_sequences, train\_tags, test\_sequences, test\_tags):

# map model classes to valid tags

unique\_tags = set(train\_tags + test\_tags)

self.class\_tag = {}

for tag, i in enumerate(sorted(unique\_tags)):

self.class\_tag[i] = tag

train\_tags = to\_categorical(train\_tags)

test\_tags = to\_categorical(test\_tags)

train\_sequences = np.array(train\_sequences)

test\_sequences = np.array(test\_sequences)

train\_tags = np.array(train\_tags)

test\_tags = np.array(test\_tags)

self.model.fit(train\_sequences, train\_tags,

validation\_data=(test\_sequences, test\_tags), batch\_size=self.batch\_size, epochs=self.epoch\_num,

callbacks=[self.epoch\_logger, self.valid\_callback])

## 1.7. Пакет test\_rus\_comments

### 1.7.1. Модуль step\_1\_restore\_tokens.py

Не депонируется

### 1.7.2. Модуль step\_2\_do\_experiments.py

Не депонируется

### 1.7.3. Модуль step\_3\_analysis.py

Не депонируется

## 1.8. Модуль tokenizing

from nltk.tokenize import TweetTokenizer

from Stemmer import Stemmer

import re

stop\_words = {

'\*', '+', '^', '&', '.', ':', '!', '\_', '-', '`', '$', "'", '\\', ';', '~', '@',

'{', '/', '=', '?', '>', '%', '"', '|', '#', '<', '}', ','

}

stemmer\_ru = Stemmer('russian')

stemmer\_en = Stemmer('english')

stop\_words\_regexes = [

r'https?://.\*[\r\n]\*' # links have not semantic

]

tokenizer = TweetTokenizer(preserve\_case=False, reduce\_len=True, strip\_handles=True)

def tokenize(text):

return tokenizer.tokenize(text)

def remove\_stopwords\_from\_text(text):

for stop\_word\_regex in stop\_words\_regexes:

text = re.sub(stop\_word\_regex, '', text)

return text

def remove\_stop\_words(tokens):

return [token for token in tokens if token not in stop\_words]

def stem\_ru(tokens):

return stemmer\_ru.stemWords(tokens)

def stem\_en(tokens):

return stemmer\_en.stemWords(tokens)

## 1.9. Модуль visualization

import matplotlib.pyplot as plt

def draw\_experiment\_points\_on\_ax(ax, title, emb\_times, accs, color, marker, time\_lim):

ax.scatter(emb\_times, accs, color=color, marker=marker)

ax.set\_title(title)

ax.set\_xlabel('emb\_time (seconds)')

ax.set\_ylabel('max\_valid\_accurancy')

ax.set\_xlim(0, time\_lim)

ax.set\_ylim(0, 1)

def draw\_experiment\_points(lsa\_accs, lsa\_emb\_times, w2v\_cbow\_accs, w2v\_cbow\_emb\_times, w2v\_skipgram\_accs,

w2v\_skipgram\_emb\_times, glove\_cbow\_accs, glove\_cbow\_emb\_times):

fig, ((lsa\_ax, w2v\_cbow\_ax), (glove\_ax, w2v\_skipgram\_ax)) = plt.subplots(2, 2)

time\_lim = max(lsa\_emb\_times + glove\_cbow\_emb\_times + w2v\_skipgram\_emb\_times + w2v\_cbow\_emb\_times) + 10

draw\_experiment\_points\_on\_ax(lsa\_ax, 'LSA', lsa\_emb\_times, lsa\_accs, 'red', '.', time\_lim)

draw\_experiment\_points\_on\_ax(w2v\_cbow\_ax, 'Word2Vec CBOW', w2v\_cbow\_emb\_times, w2v\_cbow\_accs, 'green', '.',

time\_lim)

draw\_experiment\_points\_on\_ax(w2v\_skipgram\_ax, 'Word2Vec Skip-Gram', w2v\_skipgram\_emb\_times, w2v\_skipgram\_accs,

'blue', '.', time\_lim)

draw\_experiment\_points\_on\_ax(glove\_ax, 'GloVe', glove\_cbow\_emb\_times, glove\_cbow\_accs, 'black', '.', time\_lim)

fig.tight\_layout()

plt.show()

# **2. Листинг программного модуля web-интерфейса системы анализа тональности текстов на основе LSTM-классификатора текстов**

## 2.1. Пакет alembic

### 2.1.1. Модуль env.py

Автор модуля Майкл Байер (входит в состав python-библиотеки alembic). Изменены конфигурационные строки.

### 2.1.2. Модуль script.py.mako

Автор модуля Майкл Байер (входит в состав python-библиотеки alembic).

## 2.2. Пакет validation

### 2.2.1. Модуль error\_codes.py

from enum import Enum

class ErrorCode(Enum):

UnknownError = 0, 'Unknown error'

NotDefinedError = 1, 'Field is required'

NotUniqueError = 2, 'Value is not unique'

NotAllowedValueError = 3, 'Value is not allowed'

NotStringError = 100, 'Value is not string value'

WhiteSpaceStringError = 101, 'Non empty string is required'

MaxLengthStringError = 102, 'Max allowed length is {max\_length}'

MinlengthStringError = 103, 'Min allowed length is {min\_length}'

NotIntegerStringError = 104, 'String is not integer string'

NotFloatStringError = 105, 'String is not float string'

NotDatetimeStringError = 106, 'String is not datetime string'

NotEnumStringError = 107, 'String is not valid enum string'

NotBoolStringError = 108, 'String is not boolean string'

NotPostUrlStringError = 109, 'String is not post url string'

MaxValueError = 110, 'Max allowed value is {max\_value}'

MinValueError = 111, 'Min allowed value is {min\_value}'

NotIntegerError = 201, 'Value is not integer value'

NotDatetimeError = 301, 'Value is not integer value'

NotFloatError = 401, 'Value is not float value'

NotEnumError = 501, 'Value is not valid enum value'

NotBoolError = 601, 'Value is not valid enum value'

### 2.2.2. Модуль errorinfo.py

import json

from typing import List

from validation.error\_codes import ErrorCode

class ErrorInfo:

def \_\_init\_\_(self, error\_code: ErrorCode, \*\*kwargs):

self.error\_name = error\_code.name

self.error\_code = error\_code.value[0]

self.message = error\_code.value[1].format(\*\*kwargs)

self.kwargs = {key: value for key, value in kwargs.items()}

def to\_dict(self):

error\_info = {

'code': self.error\_code,

'name': self.error\_name,

'message': self.message

}

error\_info.update(self.kwargs)

return error\_info

class FieldError(Exception):

def \_\_init\_\_(self, field\_name, errors\_info):

self.field\_name = field\_name

self.errors\_info = errors\_info

def to\_dict(self):

return {

'field\_name': self.field\_name,

'errors': [error\_info.to\_dict() for error\_info in self.errors\_info]

}

def \_\_str\_\_(self):

return print(json.dumps(self.to\_dict(), indent=4))

class ObjectError(Exception):

def \_\_init\_\_(self, obj\_name, fields\_error: List[FieldError]):

self.obj\_name = obj\_name

self.fields\_errors = {}

for field\_error in fields\_error:

if field\_error.field\_name in self.fields\_errors:

self.fields\_errors[field\_error.field\_name].extend(field\_error.errors\_info)

else:

self.fields\_errors[field\_error.field\_name] = field\_error.errors\_info

def to\_dict(self):

return {

'obj\_name': self.obj\_name,

'errors': {

field: [error\_info.to\_dict()

for error\_info in errors\_info] for field, errors\_info in self.fields\_errors.items()

}

}

def \_\_str\_\_(self):

return json.dumps(self.to\_dict(), indent=4)

### 2.2.3. Модуль validators.py

import re

from datetime import datetime

from validation.errorinfo import ErrorInfo

from validation.error\_codes import ErrorCode

DATETIME\_FORMAT = '%d.%m.%Y %H:%M:%S'

VALID\_URLS\_REGEXS = [

r"^https://www.instagram.com/p/[^\s]+/$"

]

class Validator:

def validate(self, value):

raise NotImplementedError()

class IntegerValidator(Validator):

def validate(self, value):

errors = []

new\_value = value

if isinstance(value, int):

return errors, new\_value

if not isinstance(value, str):

errors.append(ErrorInfo(ErrorCode.NotIntegerError))

return errors, None

try:

new\_value = int(new\_value)

except Exception as ex:

errors.append(ErrorCode.NotIntegerStringError)

return errors, None

return errors, new\_value

class StringValidator(Validator):

def validate(self, value):

errors = []

new\_value = value

if not isinstance(value, str):

errors.append(ErrorInfo(ErrorCode.NotStringError))

return errors, None if errors else new\_value

class FloatValidator(Validator):

def validate(self, value):

errors = []

new\_value = value

if isinstance(value, float):

return errors, new\_value

if isinstance(value, int):

new\_value = float(new\_value)

return errors, new\_value

if not isinstance(value, str):

errors.append(ErrorInfo(ErrorCode.NotFloatError))

return errors, None

try:

new\_value = float(new\_value)

except Exception as ex:

errors.append(ErrorCode.NotFloatStringError)

return errors, None

return errors, new\_value

class EnumValidator(Validator):

def \_\_init\_\_(self, enum\_class):

self.enum\_class = enum\_class

def validate(self, value):

errors = []

new\_value = value

if isinstance(value, self.enum\_class):

return errors, new\_value

if not isinstance(value, str):

errors.append(ErrorInfo(ErrorCode.NotEnumError))

return errors, None

for i, enum\_value in enumerate(self.enum\_class):

if value == enum\_value.name:

new\_value = enum\_value

return errors, new\_value

errors.append(ErrorInfo(ErrorCode.NotDatetimeStringError))

return errors, None

class BooleanValidator(Validator):

def validate(self, value):

errors = []

new\_value = value

if isinstance(value, bool):

return errors, new\_value

if isinstance(value, int):

new\_value = bool(value)

return errors, new\_value

if isinstance(value, str):

lower\_value = value.lower().strip()

if lower\_value == 'true':

new\_value = True

elif lower\_value == 'false':

new\_value = False

else:

errors.append(ErrorInfo(ErrorCode.NotBoolStringError))

return errors, None if errors else new\_value

errors.append(ErrorInfo(ErrorCode.NotBoolError))

return errors, None

class DatetimeValidator(Validator):

def validate(self, value):

errors = []

new\_value = value

if isinstance(value, datetime):

return errors, new\_value

if not isinstance(value, str):

errors.append(ErrorInfo(ErrorCode.NotDatetimeError))

return errors, None

try:

new\_value = datetime.strptime(new\_value, DATETIME\_FORMAT)

except Exception as ex:

errors.append(ErrorInfo(ErrorCode.NotDatetimeStringError))

return errors, None

return errors, new\_value

class IntegerMaxValueValidator(Validator):

def \_\_init\_\_(self, max\_value):

self.max\_value = max\_value

self.integer\_validator = IntegerValidator()

def validate(self, value):

errors, new\_value = self.integer\_validator.validate(value)

if not errors and new\_value > self.max\_value:

errors.append(ErrorInfo(ErrorCode.MaxValueError, max\_value=self.max\_value))

return errors, None if errors else new\_value

class IntegerMinValueValidator(Validator):

def \_\_init\_\_(self, min\_value):

self.min\_value = min\_value

self.integer\_validator = IntegerValidator()

def validate(self, value):

errors, new\_value = self.integer\_validator.validate(value)

if not errors and new\_value < self.min\_value:

errors.append(ErrorInfo(ErrorCode.MinValueError, min\_value=self.min\_value))

return errors, None if errors else new\_value

class StringMaxLengthValidator(Validator):

def \_\_init\_\_(self, max\_length):

self.max\_length = max\_length

self.string\_validator = StringValidator()

def validate(self, value):

errors, new\_value = self.string\_validator.validate(value)

if not errors and len(value) > self.max\_length:

errors.append(ErrorInfo(ErrorCode.MaxLengthStringError, max\_length=self.max\_length))

return errors, None if errors else new\_value

class StringMinLengthValidator(Validator):

def \_\_init\_\_(self, min\_length):

self.min\_length = min\_length

self.string\_validator = StringValidator()

def validate(self, value):

errors, new\_value = self.string\_validator.validate(value)

if not errors and len(value) < self.min\_length:

errors.append(ErrorInfo(ErrorCode.MaxLengthStringError, min\_length=self.min\_length))

return errors, None if errors else new\_value

class StringUrlValidator(Validator):

def \_\_init\_\_(self, url\_regexs=None):

self.url\_regexs = url\_regexs if url\_regexs else VALID\_URLS\_REGEXS

self.string\_validator = StringValidator()

def validate(self, value):

errors, new\_value = self.string\_validator.validate(value)

if not errors and not any(regex for regex in self.url\_regexs if re.match(regex, value)):

errors.append(ErrorInfo(ErrorCode.NotPostUrlStringError))

return errors, None if errors else new\_value

class AllowedValuesValidator(Validator):

def \_\_init\_\_(self, allowed\_values):

self.allowed\_values = allowed\_values

def validate(self, value):

errors = []

new\_value = value

if value not in self.allowed\_values:

errors.append(ErrorInfo(ErrorCode.NotAllowedValueError))

return errors, None if errors else new\_value

## 2.3. Пакет models

### 2.3.1. Модуль base\_model.py

from flask import json

from flask\_sqlalchemy import SQLAlchemy

from sqlalchemy.event import listens\_for

from sqlalchemy.orm.attributes import QueryableAttribute

from datetime import datetime

from sqlalchemy.exc import IntegrityError, OperationalError

from enum import Enum

from datetime import time

from sqlalchemy.orm.base import NO\_VALUE, NEVER\_SET

from validation.errorinfo import FieldError, ObjectError

db = SQLAlchemy()

def validatable(cls):

cls.register\_validators()

return cls

class BaseModel(db.Model):

\_\_abstract\_\_ = True

@classmethod

def register\_validators(cls):

raise NotImplementedError()

@staticmethod

def \_\_register\_col\_validators\_\_(col, validators):

@listens\_for(col, 'set', retval=True)

def validate(instance, value, oldvalue, initiator):

errors = []

new\_value = value

for validator in validators:

validator\_errors, validator\_value = validator.validate(value)

errors.extend(validator\_errors)

new\_value = validator\_value

if errors:

raise FieldError(initiator.key, errors)

return new\_value

def to\_dict(self, show=None, \_hide=[], \_path=None):

show = show or []

hidden = self.\_hidden\_fields if hasattr(self, "\_hidden\_fields") else []

default = self.\_default\_fields if hasattr(self, "\_default\_fields") else []

default.extend(['id', 'modified\_at', 'created\_at'])

if not \_path:

\_path = self.\_\_tablename\_\_.lower()

def prepend\_path(item):

item = item.lower()

if item.split(".", 1)[0] == \_path:

return item

if len(item) == 0:

return item

if item[0] != ".":

item = ".%s" % item

item = "%s%s" % (\_path, item)

return item

\_hide[:] = [prepend\_path(x) for x in \_hide]

show[:] = [prepend\_path(x) for x in show]

columns = self.\_\_table\_\_.columns.keys()

relationships = self.\_\_mapper\_\_.relationships.keys()

properties = dir(self)

ret\_data = {}

for key in columns:

if key.startswith("\_"):

continue

check = "%s.%s" % (\_path, key)

if check in \_hide or key in hidden:

continue

if check in show or key in default:

column\_type = self.\_\_table\_\_.columns[key].type.python\_type

attr\_val = getattr(self, key)

if issubclass(column\_type, Enum):

attr\_val = attr\_val.name

elif column\_type is time and attr\_val is not None:

attr\_val = attr\_val.strftime("%d.%m.%Y %H:%M:%S")

elif column\_type is datetime and attr\_val is not None:

attr\_val = attr\_val.strftime("%d.%m.%Y %H:%M:%S")

ret\_data[key] = attr\_val

for key in relationships:

if key.startswith("\_"):

continue

check = "%s.%s" % (\_path, key)

if check in \_hide or key in hidden:

continue

if check in show or key in default:

\_hide.append(check)

is\_list = self.\_\_mapper\_\_.relationships[key].uselist

if is\_list:

items = getattr(self, key)

if self.\_\_mapper\_\_.relationships[key].query\_class is not None:

if hasattr(items, "all"):

items = items.all()

ret\_data[key] = []

for item in items:

ret\_data[key].append(

item.to\_dict(

show=list(show),

\_hide=list(\_hide),

\_path=("%s.%s" % (\_path, key.lower())),

)

)

else:

if (

self.\_\_mapper\_\_.relationships[key].query\_class is not None

or self.\_\_mapper\_\_.relationships[key].instrument\_class

is not None

):

item = getattr(self, key)

if item is not None:

ret\_data[key] = item.to\_dict(

show=list(show),

\_hide=list(\_hide),

\_path=("%s.%s" % (\_path, key.lower())),

)

else:

ret\_data[key] = None

else:

ret\_data[key] = getattr(self, key)

for key in list(set(properties) - set(columns) - set(relationships)):

if key.startswith("\_"):

continue

if not hasattr(self.\_\_class\_\_, key):

continue

attr = getattr(self.\_\_class\_\_, key)

if not (isinstance(attr, property) or isinstance(attr, QueryableAttribute)):

continue

check = "%s.%s" % (\_path, key)

if check in \_hide or key in hidden:

continue

if check in show or key in default:

val = getattr(self, key)

if hasattr(val, "to\_dict"):

ret\_data[key] = val.to\_dict(

show=list(show),

\_hide=list(\_hide),

\_path=('%s.%s' % (\_path, key.lower())),

)

else:

try:

ret\_data[key] = json.loads(json.dumps(val))

except:

pass

return ret\_data

def from\_dict(self, \*\*kwargs):

"""Update this model with a dictionary."""

\_force = kwargs.pop("\_force", False)

readonly = self.\_readonly\_fields if hasattr(self, "\_readonly\_fields") else []

if hasattr(self, "\_hidden\_fields"):

readonly += self.\_hidden\_fields

readonly += ["id", "created\_at", "modified\_at"]

columns = self.\_\_table\_\_.columns.keys()

relationships = self.\_\_mapper\_\_.relationships.keys()

properties = dir(self)

changes = {}

errors = []

for key in columns:

if key.startswith("\_"):

continue

allowed = True if \_force or key not in readonly else False

exists = True if key in kwargs else False

if allowed and exists:

val = getattr(self, key)

if val != kwargs[key]:

try:

column\_type = self.\_\_table\_\_.columns[key].type.python\_type

except Exception:

continue

if issubclass(column\_type, Enum):

attr\_val = column\_type[kwargs[key]]

elif column\_type is time:

attr\_val = datetime.strptime(kwargs[key], "%d.%m.%Y %H:%M:%S")

elif column\_type is datetime:

attr\_val = datetime.strptime(kwargs[key], "%d.%m.%Y %H:%M:%S")

else:

attr\_val = kwargs[key]

changes[key] = {"old": val, "new": attr\_val}

try:

setattr(self, key, attr\_val)

except FieldError as e:

errors.append(e)

for rel in relationships:

if rel.startswith("\_"):

continue

allowed = True if \_force or rel not in readonly else False

exists = True if rel in kwargs else False

if allowed and exists:

is\_list = self.\_\_mapper\_\_.relationships[rel].uselist

if is\_list:

valid\_ids = []

rel\_list = getattr(self, rel)

# query = getattr(self, rel)

cls = self.\_\_mapper\_\_.relationships[rel].argument()

for item in kwargs[rel]:

if (

"id" in item

and any(x for x in rel\_list if x.id == item["id"])

# and query.filter\_by(id=item["id"]).limit(1).count() == 1

):

obj = cls.query.filter\_by(id=item["id"]).first()

col\_changes = obj.from\_dict(\*\*item)

if col\_changes:

col\_changes["id"] = str(item["id"])

if rel in changes:

changes[rel].append(col\_changes)

else:

changes.update({rel: [col\_changes]})

valid\_ids.append(str(item["id"]))

else:

col = cls()

col\_changes = col.from\_dict(\*\*item)

rel\_list.append(col)

# query.append(col)

db.session.flush()

if col\_changes:

col\_changes["id"] = str(col.id)

if rel in changes:

changes[rel].append(col\_changes)

else:

changes.update({rel: [col\_changes]})

if col.id is not None:

valid\_ids.append(str(col.id))

# delete rows from relationship that were not in kwargs[rel]

for item in [x for x in rel\_list if x.id is not None and x.id not in valid\_ids]:

# for item in query.filter(not\_(cls.id.in\_(valid\_ids))).all():

col\_changes = {"id": str(item.id), "deleted": True}

if rel in changes:

changes[rel].append(col\_changes)

else:

changes.update({rel: [col\_changes]})

db.session.delete(item)

else:

val = getattr(self, rel)

if self.\_\_mapper\_\_.relationships[rel].query\_class is not None:

if val is not None:

col\_changes = val.from\_dict(\*\*kwargs[rel])

if col\_changes:

changes.update({rel: col\_changes})

else:

if val != kwargs[rel]:

setattr(self, rel, kwargs[rel])

changes[rel] = {"old": val, "new": kwargs[rel]}

for key in list(set(properties) - set(columns) - set(relationships)):

if key.startswith("\_"):

continue

allowed = True if \_force or key not in readonly else False

exists = True if key in kwargs else False

if allowed and exists and getattr(self.\_\_class\_\_, key).fset is not None:

val = getattr(self, key)

if hasattr(val, "to\_dict"):

val = val.to\_dict()

changes[key] = {"old": val, "new": kwargs[key]}

setattr(self, key, kwargs[key])

if errors:

raise ObjectError(type(self).\_\_name\_\_, errors)

return changes

### 2.3.2. Модуль enums.py

from enum import Enum

class Language(Enum):

russian = 1

english = 2

class Normalization(Enum):

stemming = 1

lemmatisation = 2

class NetStatus(Enum):

created = 1

training = 2

trained = 3

error = 4

class EmbeddingModelStatus(Enum):

created = 1

training = 2

trained = 3

error = 4

class VocabularyStatus(Enum):

created = 1

extending = 2

ready = 3

error = 4

### 2.3.3. Модуль file.py

from sqlalchemy import Column, Integer, String, DateTime

from datetime import datetime

from validation.validators import IntegerValidator, StringMaxLengthValidator, DatetimeValidator

from models.base\_model import BaseModel, validatable

@validatable

class File(BaseModel):

\_\_tablename\_\_ = 'file'

id = Column(Integer, primary\_key=True)

name = Column(String(100), nullable=False)

mimetype = Column(String(50), nullable=False)

uploaded\_time = Column(DateTime, default=datetime.now)

path = Column(String(255), nullable=True)

separator = Column(String(10), nullable=True)

text\_column = Column(Integer)

tag\_column = Column(Integer)

positive\_tag = Column(String(50))

negative\_tag = Column(String(50))

@classmethod

def register\_validators(cls):

cls.\_\_register\_col\_validators\_\_(cls.id, [IntegerValidator()])

cls.\_\_register\_col\_validators\_\_(cls.name, [StringMaxLengthValidator(100)])

cls.\_\_register\_col\_validators\_\_(cls.mimetype, [StringMaxLengthValidator(50)])

cls.\_\_register\_col\_validators\_\_(cls.uploaded\_time, [DatetimeValidator()])

cls.\_\_register\_col\_validators\_\_(cls.path, [StringMaxLengthValidator(255)])

cls.\_\_register\_col\_validators\_\_(cls.separator, [StringMaxLengthValidator(10)])

cls.\_\_register\_col\_validators\_\_(cls.text\_column, [IntegerValidator()])

cls.\_\_register\_col\_validators\_\_(cls.tag\_column, [IntegerValidator()])

cls.\_\_register\_col\_validators\_\_(cls.positive\_tag, [StringMaxLengthValidator(50)])

cls.\_\_register\_col\_validators\_\_(cls.negative\_tag, [StringMaxLengthValidator(50)])

\_default\_fields = [

"name",

"mimetype",

"uploaded\_time",

"path",

"separator",

"text\_column",

"tag\_column",

"positive\_tag",

"negative\_tag"

]

### 2.3.4. Модуль vocabulary.py

from sqlalchemy import Column, Integer, String, Enum, Index, Boolean, DateTime

from sqlalchemy.orm import relationship

from datetime import datetime

from models.assotiations import vocabulary\_file

from models.enums import Language, Normalization, VocabularyStatus

from models.base\_model import BaseModel, validatable

from validation.validators import StringMaxLengthValidator, IntegerValidator, EnumValidator, BooleanValidator, \

DatetimeValidator

@validatable

class Vocabulary(BaseModel):

\_\_tablename\_\_ = 'vocabulary'

id = Column(Integer, primary\_key=True)

name = Column(String(100), nullable=False)

language = Column(Enum(Language), nullable=False)

info = Column(String(1000))

updated\_time = Column(DateTime, default=datetime.now)

lower\_cased = Column(Boolean, nullable=False)

normalization\_method = Column(Enum(Normalization), nullable=False)

punctuation\_removed = Column(Boolean, nullable=False)

words = relationship("Word", back\_populates="vocabulary")

files = relationship("File", secondary=vocabulary\_file)

status = Column(Enum(VocabularyStatus), default=VocabularyStatus.created, nullable=False)

\_\_table\_args\_\_ = (

Index('name\_and\_language\_unique', name, language, unique=True),

)

@classmethod

def register\_validators(cls):

cls.\_\_register\_col\_validators\_\_(cls.id, [IntegerValidator()])

cls.\_\_register\_col\_validators\_\_(cls.name, [StringMaxLengthValidator(50)])

cls.\_\_register\_col\_validators\_\_(cls.language, [EnumValidator(Language)])

cls.\_\_register\_col\_validators\_\_(cls.info, [StringMaxLengthValidator(1000)])

cls.\_\_register\_col\_validators\_\_(cls.updated\_time, [DatetimeValidator()])

cls.\_\_register\_col\_validators\_\_(cls.lower\_cased, [BooleanValidator()])

cls.\_\_register\_col\_validators\_\_(cls.normalization\_method, [EnumValidator(Normalization)])

cls.\_\_register\_col\_validators\_\_(cls.punctuation\_removed, [BooleanValidator()])

\_default\_fields = [

"name",

"language",

"info",

"updated\_time",

"lower\_cased",

"normalization\_method",

"punctuation\_removed"

]

### 2.3.5. Модуль word.py

from sqlalchemy import Column, Integer, String, ForeignKey, Index

from sqlalchemy.orm import relationship

from validation.validators import IntegerValidator, StringMaxLengthValidator, IntegerMinValueValidator

from models.base\_model import BaseModel, validatable

@validatable

class Word(BaseModel):

\_\_tablename\_\_ = 'word'

id = Column(Integer, primary\_key=True)

token = Column(String(100, collation="utf8mb4\_bin"), nullable=False, index=True)

token\_id = Column(Integer, nullable=False, index=True)

count = Column(Integer, nullable=False)

vocabulary\_id = Column(Integer, ForeignKey('vocabulary.id', ondelete='cascade'))

vocabulary = relationship("Vocabulary", back\_populates="words")

\_\_table\_args\_\_ = (

Index('vocabulary\_id\_and\_token\_unique', vocabulary\_id, token, unique=True),

)

@classmethod

def register\_validators(cls):

cls.\_\_register\_col\_validators\_\_(cls.id, [IntegerValidator()])

cls.\_\_register\_col\_validators\_\_(cls.token, [StringMaxLengthValidator(100)])

cls.\_\_register\_col\_validators\_\_(cls.token\_id, [IntegerMinValueValidator(0)])

cls.\_\_register\_col\_validators\_\_(cls.count, [IntegerMinValueValidator(0)])

\_default\_fields = [

"token",

"count"

]

### 2.3.6. Модуль embedding\_method.py

from sqlalchemy import Column, Integer, String, DateTime

from sqlalchemy.orm import relationship

from datetime import datetime

from models.base\_model import BaseModel, validatable

from validation.validators import IntegerValidator, StringMaxLengthValidator, DatetimeValidator

@validatable

class EmbeddingMethod(BaseModel):

\_\_tablename\_\_ = 'embedding\_method'

id = Column(Integer, primary\_key=True)

name = Column(String(100), nullable=False, unique=True)

info = Column(String(1000))

updated\_time = Column(DateTime, default=datetime.now)

plugin\_path = Column(String(255), nullable=False, unique=True)

params = relationship("EmbeddingMethodParam")

embedding\_models = relationship("EmbeddingModel", back\_populates="embedding\_method")

@classmethod

def register\_validators(cls):

cls.\_\_register\_col\_validators\_\_(cls.id, [IntegerValidator()])

cls.\_\_register\_col\_validators\_\_(cls.name, [StringMaxLengthValidator(100)])

cls.\_\_register\_col\_validators\_\_(cls.info, [StringMaxLengthValidator(1000)])

cls.\_\_register\_col\_validators\_\_(cls.updated\_time, [DatetimeValidator()])

cls.\_\_register\_col\_validators\_\_(cls.plugin\_path, [StringMaxLengthValidator(255)])

\_default\_fields = [

"name",

"info",

"updated\_time",

"plugin\_path",

"params"

]

### 2.3.7. Модуль embedding\_method\_param.py

from sqlalchemy import Column, Integer, String, Index, ForeignKey

from models.base\_model import BaseModel, validatable

from validation.validators import IntegerValidator, StringMaxLengthValidator, AllowedValuesValidator

@validatable

class EmbeddingMethodParam(BaseModel):

\_\_tablename\_\_ = 'embedding\_method\_param'

id = Column(Integer, primary\_key=True)

name = Column(String(50), nullable=False)

type = Column(String(10), nullable=False)

display\_name = Column(String(100), nullable=False)

embedding\_method\_id = Column(Integer, ForeignKey("embedding\_method.id"))

\_\_table\_args\_\_ = (

Index('embedding\_method\_id\_and\_name\_unique', embedding\_method\_id, name, unique=True),

)

@classmethod

def register\_validators(cls):

cls.\_\_register\_col\_validators\_\_(cls.id, [IntegerValidator()])

cls.\_\_register\_col\_validators\_\_(cls.name, [StringMaxLengthValidator(50)])

cls.\_\_register\_col\_validators\_\_(cls.type, [AllowedValuesValidator(['str', 'int', 'float'])])

cls.\_\_register\_col\_validators\_\_(cls.display\_name, [StringMaxLengthValidator(100)])

\_default\_fields = [

"name",

"type",

"display\_name"

]

### 2.3.8. Модуль embedding\_model.py

from sqlalchemy import Column, Integer, String, ForeignKey, Enum, DateTime

from sqlalchemy.orm import relationship

from datetime import datetime

from validation.validators import IntegerValidator, StringMaxLengthValidator, DatetimeValidator, EnumValidator

from models.enums import EmbeddingModelStatus

from models.assotiations import embedding\_model\_file

from models.base\_model import BaseModel, validatable

@validatable

class EmbeddingModel(BaseModel):

\_\_tablename\_\_ = 'embedding\_model'

id = Column(Integer, primary\_key=True)

name = Column(String(100), nullable=False, unique=True)

info = Column(String(1000))

updated\_time = Column(DateTime, default=datetime.now)

embedding\_dim = Column(Integer)

status = Column(Enum(EmbeddingModelStatus), default=EmbeddingModelStatus.created, nullable=False)

embedding\_method\_id = Column(Integer, ForeignKey("embedding\_method.id"))

embedding\_method = relationship("EmbeddingMethod", back\_populates="embedding\_models")

embedding\_params = relationship("EmbeddingMethodParamValue")

vocabulary\_id = Column(Integer, ForeignKey("vocabulary.id"))

vocabulary = relationship("Vocabulary")

model\_word\_vectors = relationship("ModelWordVector")

files = relationship("File", secondary=embedding\_model\_file)

@classmethod

def register\_validators(cls):

cls.\_\_register\_col\_validators\_\_(cls.id, [IntegerValidator()])

cls.\_\_register\_col\_validators\_\_(cls.name, [StringMaxLengthValidator(100)])

cls.\_\_register\_col\_validators\_\_(cls.info, [StringMaxLengthValidator(1000)])

cls.\_\_register\_col\_validators\_\_(cls.updated\_time, [DatetimeValidator()])

cls.\_\_register\_col\_validators\_\_(cls.embedding\_dim, [IntegerValidator()])

cls.\_\_register\_col\_validators\_\_(cls.status, [EnumValidator(EmbeddingModelStatus)])

\_default\_fields = [

"name",

"info",

"updated\_time",

"embedding\_dim",

"embedding\_method",

"embedding\_params",

"vocabulary",

"files",

"status"

]

### 2.3.9. Модуль embedding\_method\_param\_value.py

from sqlalchemy import Column, Integer, Float, ForeignKey

from sqlalchemy.orm import relationship

from models.base\_model import BaseModel, validatable

from validation.validators import IntegerValidator, FloatValidator

@validatable

class EmbeddingMethodParamValue(BaseModel):

\_\_tablename\_\_ = 'embedding\_method\_param\_value'

id = Column(Integer, primary\_key=True)

value = Column(Float(precision=5), nullable=False)

embedding\_method\_param\_id = Column(Integer, ForeignKey("embedding\_method\_param.id"))

embedding\_method\_param = relationship("EmbeddingMethodParam")

embedding\_model\_id = Column(Integer, ForeignKey("embedding\_model.id"))

@classmethod

def register\_validators(cls):

cls.\_\_register\_col\_validators\_\_(cls.id, [IntegerValidator()])

cls.\_\_register\_col\_validators\_\_(cls.value, [FloatValidator()])

\_default\_fields = [

"value",

"embedding\_method\_param"

]

### 2.3.10. Модуль model\_word\_vector.py

from sqlalchemy import Column, Integer, Float, ForeignKey, Text

from sqlalchemy.orm import relationship

from validation.validators import IntegerMinValueValidator, StringValidator, FloatValidator

from models.base\_model import BaseModel, validatable

@validatable

class ModelWordVector(BaseModel):

\_\_tablename\_\_ = 'model\_word\_vector'

id = Column(Integer, primary\_key=True)

vector = Column(Text, nullable=False)

index = Column(Integer, nullable=False)

x = Column(Float, nullable=True)

y = Column(Float, nullable=True)

word\_id = Column(Integer, ForeignKey("word.id"))

word = relationship("Word")

embedding\_model\_id = Column(Integer, ForeignKey("embedding\_model.id"))

@classmethod

def register\_validators(cls):

cls.\_\_register\_col\_validators\_\_(cls.id, [IntegerMinValueValidator(0)])

cls.\_\_register\_col\_validators\_\_(cls.vector, [StringValidator()])

cls.\_\_register\_col\_validators\_\_(cls.index, [IntegerMinValueValidator(0)])

cls.\_\_register\_col\_validators\_\_(cls.x, [FloatValidator()])

cls.\_\_register\_col\_validators\_\_(cls.y, [FloatValidator()])

\_default\_fields = [

"vector",

"index",

"x",

"y",

"word"

]

### 2.3.11. Модуль net\_module.py

from sqlalchemy import Column, Integer, String, DateTime

from sqlalchemy.orm import relationship

from datetime import datetime

from validation.validators import IntegerMinValueValidator, StringMaxLengthValidator, DatetimeValidator

from models.base\_model import BaseModel, validatable

@validatable

class NetModule(BaseModel):

\_\_tablename\_\_ = 'net\_module'

id = Column(Integer, primary\_key=True)

name = Column(String(100), nullable=False, unique=True)

info = Column(String(1000))

updated\_time = Column(DateTime, default=datetime.now)

plugin\_path = Column(String(255), nullable=False, unique=True)

net\_module\_params = relationship("NetModuleParam")

nets = relationship("Net", back\_populates="net\_module")

@classmethod

def register\_validators(cls):

cls.\_\_register\_col\_validators\_\_(cls.id, [IntegerMinValueValidator(0)])

cls.\_\_register\_col\_validators\_\_(cls.name, [StringMaxLengthValidator(100)])

cls.\_\_register\_col\_validators\_\_(cls.info, [StringMaxLengthValidator(1000)])

cls.\_\_register\_col\_validators\_\_(cls.updated\_time, [DatetimeValidator()])

cls.\_\_register\_col\_validators\_\_(cls.plugin\_path, [StringMaxLengthValidator(255)])

\_default\_fields = [

"name",

"info",

"updated\_time",

"plugin\_path",

"net\_module\_params"

]

### 2.3.12. Модуль net\_module\_param.py

from sqlalchemy import Column, Integer, String, Index, ForeignKey

from validation.validators import IntegerMinValueValidator, StringMaxLengthValidator, AllowedValuesValidator

from models.base\_model import BaseModel, validatable

@validatable

class NetModuleParam(BaseModel):

\_\_tablename\_\_ = 'net\_module\_param'

id = Column(Integer, primary\_key=True)

name = Column(String(50), nullable=False)

type = Column(String(10), nullable=False)

display\_name = Column(String(100), nullable=False)

net\_module\_id = Column(Integer, ForeignKey("net\_module.id"))

\_\_table\_args\_\_ = (

Index('net\_module\_id\_and\_name\_unique', net\_module\_id, name, unique=True),

)

@classmethod

def register\_validators(cls):

cls.\_\_register\_col\_validators\_\_(cls.id, [IntegerMinValueValidator(0)])

cls.\_\_register\_col\_validators\_\_(cls.name, [StringMaxLengthValidator(50)])

cls.\_\_register\_col\_validators\_\_(cls.type, [AllowedValuesValidator(['float', 'str', 'int'])])

cls.\_\_register\_col\_validators\_\_(cls.display\_name, [StringMaxLengthValidator(100)])

\_default\_fields = [

"name",

"type",

"display\_name"

]

### 2.3.13. Модуль net.py

from sqlalchemy import Column, Integer, String, ForeignKey, Enum, DateTime

from sqlalchemy.orm import relationship

from datetime import datetime

from models.assotiations import net\_file

from models.enums import NetStatus

from validation.validators import IntegerMinValueValidator, StringMaxLengthValidator, DatetimeValidator, EnumValidator

from models.base\_model import BaseModel, validatable

@validatable

class Net(BaseModel):

\_\_tablename\_\_ = 'net'

id = Column(Integer, primary\_key=True)

name = Column(String(100), nullable=False, unique=True)

info = Column(String(1000))

updated\_time = Column(DateTime, default=datetime.now)

net\_file\_path = Column(String(255), nullable=True, unique=True)

status = Column(Enum(NetStatus), default=NetStatus.created, nullable=False)

net\_module\_id = Column(Integer, ForeignKey("net\_module.id"))

net\_module = relationship("NetModule", back\_populates="nets")

net\_module\_param\_values = relationship("NetModuleParamValue")

embedding\_model\_id = Column(Integer, ForeignKey("embedding\_model.id"))

embedding\_model = relationship("EmbeddingModel")

files = relationship("File", secondary=net\_file)

epochs = relationship("Epoch")

@classmethod

def register\_validators(cls):

cls.\_\_register\_col\_validators\_\_(cls.id, [IntegerMinValueValidator(0)])

cls.\_\_register\_col\_validators\_\_(cls.name, [StringMaxLengthValidator(100)])

cls.\_\_register\_col\_validators\_\_(cls.info, [StringMaxLengthValidator(100)])

cls.\_\_register\_col\_validators\_\_(cls.updated\_time, [DatetimeValidator()])

cls.\_\_register\_col\_validators\_\_(cls.net\_file\_path, [StringMaxLengthValidator(255)])

cls.\_\_register\_col\_validators\_\_(cls.status, [EnumValidator(NetStatus)])

\_default\_fields = [

"name",

"info",

"updated\_time",

"net\_module",

"net\_module\_param\_values",

"embedding\_model",

"net\_file\_path",

"files",

"status",

"epochs"

]

### 2.3.14. Модуль net\_module\_param\_value.py

from sqlalchemy import Column, Integer, String, Index, Float, ForeignKey

from sqlalchemy.orm import relationship

from validation.validators import IntegerMinValueValidator, FloatValidator

from models.base\_model import BaseModel, validatable

@validatable

class NetModuleParamValue(BaseModel):

\_\_tablename\_\_ = 'net\_module\_param\_value'

id = Column(Integer, primary\_key=True)

value = Column(Float(precision=5), nullable=False)

net\_module\_param\_id = Column(Integer, ForeignKey("net\_module\_param.id"))

net\_module\_param = relationship("NetModuleParam")

net\_id = Column(Integer, ForeignKey("net.id"))

@classmethod

def register\_validators(cls):

cls.\_\_register\_col\_validators\_\_(cls.id, [IntegerMinValueValidator(0)])

cls.\_\_register\_col\_validators\_\_(cls.value, [FloatValidator()])

\_default\_fields = [

"value",

"net\_module\_param"

]

### 2.3.15. Модуль net\_epoch.py

from sqlalchemy import Column, Integer, String, ForeignKey, Float

from sqlalchemy.orm import relationship

from validation.validators import IntegerMinValueValidator, StringMaxLengthValidator, FloatValidator

from models.base\_model import BaseModel, validatable

@validatable

class Epoch(BaseModel):

\_\_tablename\_\_ = 'epoch'

id = Column(Integer, primary\_key=True)

order\_number = Column(Integer, nullable=False)

epoch\_file\_path = Column(String(255), nullable=True, unique=True)

training\_time = Column(Integer, nullable=True)

train\_accurancy = Column(Float, nullable=True)

train\_loss = Column(Float, nullable=True)

test\_accurancy = Column(Float, nullable=True)

test\_loss = Column(Float, nullable=True)

net\_id = Column(Integer, ForeignKey("net.id"))

epoch\_confusion\_matrix = relationship("EpochConfusionMatrix")

@classmethod

def register\_validators(cls):

cls.\_\_register\_col\_validators\_\_(cls.id, [IntegerMinValueValidator(0)])

cls.\_\_register\_col\_validators\_\_(cls.order\_number, [IntegerMinValueValidator(0)])

cls.\_\_register\_col\_validators\_\_(cls.epoch\_file\_path, [StringMaxLengthValidator(255)])

cls.\_\_register\_col\_validators\_\_(cls.training\_time, [IntegerMinValueValidator(0)])

cls.\_\_register\_col\_validators\_\_(cls.train\_accurancy, [FloatValidator()])

cls.\_\_register\_col\_validators\_\_(cls.train\_loss, [FloatValidator()])

cls.\_\_register\_col\_validators\_\_(cls.test\_accurancy, [FloatValidator()])

cls.\_\_register\_col\_validators\_\_(cls.test\_loss, [FloatValidator()])

\_default\_fields = [

"order\_number",

"epoch\_file\_path",

"epoch\_confusion\_matrix",

"training\_time",

"train\_accurancy",

"train\_loss",

"test\_accurancy",

"test\_loss"

]

### 2.3.16. Модуль epoch\_confusion\_matrix.py

from sqlalchemy import Column, Integer, String, ForeignKey

from models.base\_model import BaseModel, validatable

from validation.validators import StringMaxLengthValidator, IntegerMinValueValidator

@validatable

class EpochConfusionMatrix(BaseModel):

\_\_tablename\_\_ = 'epoch\_confusion\_matrix'

id = Column(Integer, primary\_key=True)

epoch\_id = Column(Integer, ForeignKey("epoch.id"))

true\_tag = Column(String(50), nullable=False)

predicted\_tag = Column(String(50), nullable=False)

count = Column(Integer, nullable=False)

@classmethod

def register\_validators(cls):

cls.\_\_register\_col\_validators\_\_(cls.id, [IntegerMinValueValidator(0)])

cls.\_\_register\_col\_validators\_\_(cls.true\_tag, [StringMaxLengthValidator(50)])

cls.\_\_register\_col\_validators\_\_(cls.predicted\_tag, [StringMaxLengthValidator(50)])

cls.\_\_register\_col\_validators\_\_(cls.count, [IntegerMinValueValidator(0)])

\_default\_fields = [

"true\_tag",

"predicted\_tag",

"count"

]

### 2.3.17. Модуль link.py

from sqlalchemy import Column, Integer, Text

from models.base\_model import BaseModel, validatable

from validation.validators import IntegerMinValueValidator, StringUrlValidator

@validatable

class Link(BaseModel):

\_\_tablename\_\_ = 'link'

id = Column(Integer, primary\_key=True)

url = Column(Text, nullable=False)

@classmethod

def register\_validators(cls):

cls.\_\_register\_col\_validators\_\_(cls.id, [IntegerMinValueValidator(0)])

cls.\_\_register\_col\_validators\_\_(cls.url, [StringUrlValidator()])

\_default\_fields = [

"url"

]

### 2.3.18. Модуль text\_set.py

from sqlalchemy import Column, Integer, String, ForeignKey

from sqlalchemy.orm import relationship

from models.base\_model import BaseModel, validatable

from validation.validators import IntegerMinValueValidator, StringMaxLengthValidator

@validatable

class TextSet(BaseModel):

\_\_tablename\_\_ = 'text\_set'

id = Column(Integer, primary\_key=True)

name = Column(String(256), nullable=False)

files = relationship("TextSetFile")

links = relationship("TextSetLink")

net = relationship("Net")

net\_id = Column(Integer, ForeignKey("net.id"))

@classmethod

def register\_validators(cls):

cls.\_\_register\_col\_validators\_\_(cls.id, [IntegerMinValueValidator(0)])

cls.\_\_register\_col\_validators\_\_(cls.name, [StringMaxLengthValidator(256)])

\_default\_fields = [

"name",

"files",

"links",

"net"

]

### 2.3.19. Модуль text.py

from sqlalchemy import Column, Integer, String, ForeignKey, Float, Text

from sqlalchemy.orm import relationship

from validation.validators import IntegerMinValueValidator, StringValidator, FloatValidator

from models.base\_model import BaseModel, validatable

@validatable

class Text(BaseModel):

\_\_tablename\_\_ = 'text'

id = Column(Integer, primary\_key=True)

text\_type = Column(String(50))

text = Column(Text, nullable=False)

negative\_probability = Column(Float, nullable=False)

positive\_probability = Column(Float, nullable=False)

\_\_mapper\_args\_\_ = {

'polymorphic\_identity': 'text',

'polymorphic\_on': text\_type

}

@classmethod

def register\_validators(cls):

cls.\_\_register\_col\_validators\_\_(cls.id, [IntegerMinValueValidator(0)])

cls.\_\_register\_col\_validators\_\_(cls.text, [StringValidator()])

cls.\_\_register\_col\_validators\_\_(cls.negative\_probability, [FloatValidator()])

cls.\_\_register\_col\_validators\_\_(cls.positive\_probability, [FloatValidator()])

\_default\_fields = [

"text",

"negative\_probability",

"positive\_probability"

]

@validatable

class FileText(Text):

\_\_tablename\_\_ = 'file\_text'

id = Column(Integer, ForeignKey('text.id'), primary\_key=True)

text\_set\_file\_id = Column(Integer, ForeignKey('text\_set\_file.id'), nullable=False)

text\_set\_file = relationship("TextSetFile", back\_populates='texts')

\_\_mapper\_args\_\_ = {

'polymorphic\_identity': 'file\_text',

}

@validatable

class LinkText(Text):

\_\_tablename\_\_ = 'link\_text'

id = Column(Integer, ForeignKey('text.id'), primary\_key=True)

text\_set\_link\_id = Column(Integer, ForeignKey('text\_set\_link.id'), nullable=False)

text\_set\_link = relationship("TextSetLink", back\_populates='texts')

id\_from\_link = Column(Integer, nullable=False)

\_\_mapper\_args\_\_ = {

'polymorphic\_identity': 'link\_text',

}

### 2.3.20. Модуль text\_set\_file.py

from sqlalchemy import Column, Integer, ForeignKey

from sqlalchemy.orm import relationship

from models.base\_model import BaseModel

class TextSetFile(BaseModel):

\_\_tablename\_\_ = 'text\_set\_file'

id = Column(Integer, primary\_key=True)

text\_set\_id = Column(Integer, ForeignKey('text\_set.id'), nullable=False)

file\_id = Column(Integer, ForeignKey('file.id'), nullable=False)

texts = relationship("FileText", back\_populates='text\_set\_file')

file = relationship("File")

\_default\_fields = [

"texts",

"file"

]

### 2.3.21. Модуль text\_set\_link.py

from sqlalchemy import Column, Integer, ForeignKey

from sqlalchemy.orm import relationship

from models.base\_model import BaseModel

class TextSetLink(BaseModel):

\_\_tablename\_\_ = 'text\_set\_link'

id = Column(Integer, primary\_key=True)

text\_set\_id = Column(Integer, ForeignKey('text\_set.id'), nullable=False)

link\_id = Column(Integer, ForeignKey('link.id'), nullable=False)

texts = relationship("LinkText", back\_populates='text\_set\_link')

link = relationship("Link")

\_default\_fields = [

"texts",

"link"

]

### 2.3.22. Модуль assotiations.py

from sqlalchemy import Column, Integer, Table, ForeignKey

from models.base\_model import BaseModel

vocabulary\_file = Table('vocabulary\_file', BaseModel.metadata,

Column('vocabulary\_id', Integer, ForeignKey('vocabulary.id')),

Column('file\_id', Integer, ForeignKey('file.id'))

)

embedding\_model\_file = Table('embedding\_model\_file', BaseModel.metadata,

Column('embedding\_model\_id', Integer, ForeignKey('embedding\_model.id')),

Column('file\_id', Integer, ForeignKey('file.id'))

)

net\_file = Table('net\_file', BaseModel.metadata,

Column('net\_id', Integer, ForeignKey('net.id')),

Column('file\_id', Integer, ForeignKey('file.id'))

)

## 2.4. Пакет logic

### 2.4.1. Модуль file.py

import os

from werkzeug.utils import secure\_filename

from models import db, File

filepath = "/home/nastya/ftp\_files/{id}{ext}"

def get(file\_id):

file = db.session.query(File).get(file\_id) # type: File

if file is None:

raise ValueError("No file found by id {0}".format(file\_id))

return file

def save(file, filestream):

full\_filename = secure\_filename(filestream.filename)

file.name = full\_filename

file.mimetype = filestream.content\_type

db.session.add(file)

db.session.flush()

filename, file\_extension = os.path.splitext(full\_filename)

full\_path = filepath.format(id=file.id, ext=file\_extension)

filestream.save(full\_path)

file.path = full\_path

db.session.commit()

return file

### 2.4.2. Модуль vocabulary.py

from threading import Thread

import flask

from sqlalchemy.orm import sessionmaker, scoped\_session

from dataio.csv import read\_texts\_from\_file

from models import Vocabulary, Word, Normalization, Language, db, File, VocabularyStatus

from tokenizing import remove\_stopwords\_from\_text, remove\_stop\_words, tokenize, stem\_en, stem\_ru

import logic.file as file\_logic

def get\_token\_count\_dict(tokens):

token\_count\_dic = {}

for token in tokens:

db\_token = token.replace

token\_count\_dic[token] = token\_count\_dic.get(token, 0) + 1

return token\_count\_dic

def get\_next\_token\_id(vocabulary):

if vocabulary.words:

return max(vocabulary.words, key=lambda x: x.token\_id).token\_id + 1

else:

return 1

def add\_tokens\_to\_vocabulary(vocab: Vocabulary, tokens=None):

if not tokens:

return vocab

token\_count = get\_token\_count\_dict(tokens)

next\_token\_id = get\_next\_token\_id(vocab)

words = db.session.query(Word).filter\_by(vocabulary\_id=vocab.id)

# increase count for already existing

for word in words:

if word.token in token\_count:

word.count += token\_count[word.token]

token\_count[word.token] = 0

# add new for not existing

for token, count in token\_count.items():

if count == 0:

continue

try:

word = Word(next\_token\_id, token, count)

except ValueError:

continue

word.vocabulary = vocab

next\_token\_id += 1

return vocab

def get\_tokens\_by\_vocabulary(vocab: Vocabulary, texts):

if not texts:

return None

texts\_tokens = []

vocab\_word\_id = {word.token: word.id for word in vocab.words}

for text in texts:

text = remove\_stopwords\_from\_text(text)

vocab.lower\_cased = True

text\_tokens = tokenize(text)

if vocab.punctuation\_removed:

text\_tokens = remove\_stop\_words(text\_tokens)

if vocab.normalization\_method == Normalization.stemming:

if vocab.language == Language.russian:

text\_tokens = stem\_ru(text\_tokens)

if vocab.language == Language.english:

text\_tokens = stem\_en(text\_tokens)

texts\_tokens.append([token if token in vocab\_word\_id else "<<UNKNOWN>>" for token in text\_tokens])

return texts\_tokens

def add\_texts\_to\_vocabulary(vocab: Vocabulary, texts):

if not texts:

return None

tokens = []

for text in texts:

text = remove\_stopwords\_from\_text(text)

vocab.lower\_cased = True

text\_tokens = tokenize(text)

if vocab.punctuation\_removed:

text\_tokens = remove\_stop\_words(text\_tokens)

if vocab.normalization\_method == Normalization.stemming:

if vocab.language == Language.russian:

text\_tokens = stem\_ru(text\_tokens)

if vocab.language == Language.english:

text\_tokens = stem\_en(text\_tokens)

tokens.extend(text\_tokens)

tokens = list(set(tokens))

return add\_tokens\_to\_vocabulary(vocab, tokens)

def extend(vocabulary\_id, app):

with app.app\_context():

change\_status(vocabulary\_id, VocabularyStatus.extending)

vocabulary = get(vocabulary\_id)

for file in vocabulary.files:

# text do not need to be quoted if file is text file (text per line), without separators

texts = read\_texts\_from\_file(

file=file.file.path,

quoted\_fields=file.file.separator,

separator=file.file.separator,

text\_col=file.file.text\_column

)

add\_texts\_to\_vocabulary(vocabulary, texts)

change\_status(vocabulary\_id, VocabularyStatus.ready)

def init\_extending(vocabulary\_id, file\_ids):

vocabulary = get(vocabulary\_id)

for file\_id in file\_ids:

if any(voc\_file for voc\_file in vocabulary.files if voc\_file.id == file\_id):

continue

file = file\_logic.get(file\_id)

vocabulary.files.append(file)

db.session.commit()

Thread(target=extend, args=(vocabulary\_id, flask.current\_app.\_get\_current\_object(),)).start()

return vocabulary

def get(vocabulary\_id):

vocabulary = db.session.query(Vocabulary).get(vocabulary\_id)

if vocabulary is None:

raise ValueError("No vocabulary found by id {}".format(vocabulary\_id))

return vocabulary

def change\_status(vocabulary\_id, status: VocabularyStatus):

vocabulary = get(vocabulary\_id)

vocabulary.status = status

db.session.commit()

return vocabulary

### 2.4.3. Модуль embedding\_model.py

from typing import List, Tuple, Dict

from time import sleep

import numpy as np

from threading import Thread

import flask

from keras\_preprocessing.sequence import pad\_sequences

from dataio.csv import read\_samples

from helpers.helpers import shuffle\_and\_get\_parts

from models import EmbeddingModel, EmbeddingModelStatus, db, File, EmbeddingMethod, EmbeddingMethodParamValue, \

ModelWordVector, VocabularyStatus

import logic.vocabulary as vocabulary\_logic

import logic.file as file\_logic

def get(model\_id):

model = db.session.query(EmbeddingModel).get(model\_id) # type: EmbeddingModel

if model is None:

raise ValueError("No embedding model found by id {0}".format(model\_id))

return model

def change\_status(model\_id, new\_status: EmbeddingModelStatus):

model = get(model\_id)

model.status = new\_status

db.session.commit()

return model

def add\_word\_vectors(model\_id, word\_dictionary: Dict[str, Tuple[int, List[float]]]):

model = get(model\_id)

vocabulary\_token\_word\_dict = {word.token: word for word in model.vocabulary.words}

for token, (index, vector) in word\_dictionary.items():

word = vocabulary\_token\_word\_dict.get(token, None)

if word is None:

continue

model\_word\_vector = ModelWordVector()

model\_word\_vector.word = word

model\_word\_vector.index = index

model\_word\_vector.vector = '\t'.join([str(val) for val in vector])

model.model\_word\_vectors.append(model\_word\_vector)

db.session.commit()

def tsne(model\_id):

from sklearn.manifold import TSNE

model = get(model\_id)

tsne = TSNE(n\_components=2)

embedding\_matrix = get\_embedding\_matrix(model\_id)

tsne\_matrix = tsne.fit\_transform(embedding\_matrix)

for word\_vector in model.model\_word\_vectors: # type: ModelWordVector

word\_vector.x = tsne\_matrix[word\_vector.index][0].item()

word\_vector.y = tsne\_matrix[word\_vector.index][1].item()

db.session.commit()

def train(model\_id, app):

with app.app\_context():

model = get(model\_id)

if model.vocabulary.status == VocabularyStatus.created:

vocabulary\_logic.extend(model.vocabulary\_id, app)

elif model.vocabulary.status == VocabularyStatus.extending:

while True:

db.session.refresh(model)

if model.vocabulary.status == VocabularyStatus.ready:

break

sleep(30)

# if model.vocabulary.status != VocabularyStatus.ready:

# raise ValueError('Vocabulary has {} status (not ready)'.format(model.vocabulary.status))

model = change\_status(model\_id, EmbeddingModelStatus.training)

samples = []

tags = []

for file in model.files:

n, p = read\_samples(file.path, file.separator, file.tag\_column - 1, file.negative\_tag, file.positive\_tag,

file.text\_column - 1)

samples.extend([sample for sample, tag in n])

tags.extend([0 for sample, tag in n])

samples.extend([sample for sample, tag in p])

tags.extend([1 for sample, tag in p])

vocabulary = model.vocabulary

samples = vocabulary\_logic.get\_tokens\_by\_vocabulary(vocabulary, samples)

random\_emd\_samples\_part = shuffle\_and\_get\_parts(samples, tags)[0]

emb\_samples\_tokens = random\_emd\_samples\_part[0]

method = model.embedding\_method # type: EmbeddingMethod

import importlib.util

method\_module\_spec = importlib.util.spec\_from\_file\_location(method.name, method.plugin\_path)

method\_module = importlib.util.module\_from\_spec(method\_module\_spec)

method\_module\_spec.loader.exec\_module(method\_module)

max\_doc\_len = max([len(tokens) for tokens in emb\_samples\_tokens])

from pydoc import locate

additional\_params = {}

for param\_value in model.embedding\_params:

name = param\_value.embedding\_method\_param.name

param\_type = locate(param\_value.embedding\_method\_param.type)

value = param\_type(param\_value.value)

additional\_params[name] = value

embedding\_model = method\_module.EmbeddingModel(

emb\_samples\_tokens,

emb\_dim=model.embedding\_dim,

\*\*additional\_params

)

word\_dictionary = embedding\_model.get\_word\_dictionary()

add\_word\_vectors(model\_id, word\_dictionary)

tsne(model\_id)

change\_status(model\_id, EmbeddingModelStatus.trained)

def init\_training(model\_id, file\_ids, extend\_vocabulary):

model = change\_status(model\_id, EmbeddingModelStatus.created)

for file\_id in file\_ids:

file = file\_logic.get(file\_id)

if not any(model\_file for model\_file in model.files if model\_file.id == file\_id):

model.files.append(file)

if extend\_vocabulary and not any(voc\_file for voc\_file in model.vocabulary.files if voc\_file.id == file\_id):

model.vocabulary.files.append(file)

db.session.commit()

Thread(target=train, args=(model\_id, flask.current\_app.\_get\_current\_object(),)).start()

return model

def get\_sequences(model\_id, text\_tokens, seq\_len):

model = get(model\_id)

token\_index = {word\_vector.word.token: word\_vector.index for word\_vector in model.model\_word\_vectors}

sequences = []

for tokens in text\_tokens:

sequence = pad\_sequences([[token\_index.get(token, 1) for token in tokens]], maxlen=seq\_len)[0]

sequences.append(sequence)

return sequences

def get\_vector\_from\_str(vec\_string):

return [float(i) for i in vec\_string.split('\t')]

def get\_embedding\_matrix(model\_id):

model = get(model\_id)

word\_dic = {word\_vector.word.token: (word\_vector.index, get\_vector\_from\_str(word\_vector.vector))

for word\_vector in model.model\_word\_vectors}

word\_num = len(word\_dic) + 2

embedding\_dim = model.embedding\_dim

embedding\_matrix = np.zeros((word\_num, embedding\_dim))

for word, (i, vector) in word\_dic.items():

embedding\_matrix[i] = vector

return embedding\_matrix

def delete(model\_id):

model = get(model\_id)

for param in model.embedding\_params:

db.session.delete(param)

for word\_vector in model.model\_word\_vectors:

db.session.delete(word\_vector)

db.session.delete(model)

### 2.4.4. Модуль net.py

from threading import Thread

from time import sleep

import flask

from dataio.csv import read\_samples

from helpers.helpers import shuffle\_and\_get\_parts

from models import Net, NetStatus, db, File, EmbeddingModelStatus, Epoch, EpochConfusionMatrix, TextSet

import logic.vocabulary as vocabulary\_logic

import logic.embedding\_model as embedding\_model\_logic

import logic.file as file\_logic

import netword\_metrics.confusion\_matrix as net\_metrics

from network.lstm\_categorical import LSTMCategorical

def get(net\_id):

net = db.session.query(Net).get(net\_id) # type: Net

if net is None:

raise ValueError("No net found by id {0}".format(net\_id))

return net

def change\_status(net\_id, new\_status: NetStatus):

net = get(net\_id)

net.status = new\_status

db.session.commit()

return net

def train(net\_id, app):

with app.app\_context():

train\_count = 1700

test\_count = 300

lstm\_out = 100

batch\_size = 32

net\_epochs = 10

dropout = 20

net = get(net\_id)

if net.embedding\_model.status == EmbeddingModelStatus.created:

embedding\_model\_logic.train(net.embedding\_model\_id, app)

elif net.embedding\_model.status == EmbeddingModelStatus.training:

while True:

db.session.refresh(net)

if net.embedding\_model.status == EmbeddingModelStatus.trained:

break

sleep(30)

net = change\_status(net\_id, NetStatus.training)

samples = []

tags = []

for file in net.files:

n, p = read\_samples(file.path, file.separator, file.tag\_column-1, file.negative\_tag, file.positive\_tag, file.text\_column-1)

samples.extend([sample for sample, tag in n])

tags.extend([0 for sample, tag in n])

samples.extend([sample for sample, tag in p])

tags.extend([1 for sample, tag in p])

vocabulary = net.embedding\_model.vocabulary

samples = vocabulary\_logic.get\_tokens\_by\_vocabulary(vocabulary, samples)

[train\_set, test\_set] = shuffle\_and\_get\_parts(samples, tags, [train\_count, test\_count])

train\_samples\_tokens = train\_set[0]

train\_samples\_tags = train\_set[1]

test\_samples\_tokens = test\_set[0]

test\_samples\_tags = test\_set[1]

max\_doc\_len = max([len(tokens) for tokens in train\_samples\_tokens + test\_samples\_tokens])

train\_sequences = embedding\_model\_logic.get\_sequences(net.embedding\_model\_id, train\_samples\_tokens, max\_doc\_len)

test\_sequences = embedding\_model\_logic.get\_sequences(net.embedding\_model\_id, test\_samples\_tokens, max\_doc\_len)

embedding\_matrix = embedding\_model\_logic.get\_embedding\_matrix(net.embedding\_model.id)

doc\_len = max([len(tokens) for tokens in train\_samples\_tokens+test\_samples\_tokens])

net\_model = LSTMCategorical(lstm\_out, doc\_len, dropout=dropout, batch\_size=batch\_size, epoch\_num=net\_epochs,

embedding\_matrix=embedding\_matrix)

net\_model.train\_by\_sequences(train\_sequences, train\_samples\_tags, test\_sequences, test\_samples\_tags)

epoch\_true\_tags, epoch\_pred\_tags = net\_model.get\_epoch\_predictions()

for epoch\_num, true\_tags in epoch\_true\_tags.items():

pred\_tags = epoch\_pred\_tags[epoch\_num]

epoch = Epoch()

epoch.order\_number = epoch\_num

epoch.training\_time = net\_model.epoch\_logger.epoch\_time\_dic[epoch\_num]

epoch.train\_accurancy = net\_model.epoch\_logger.epoch\_acc\_dic[epoch\_num].item()

epoch.train\_loss = net\_model.epoch\_logger.epoch\_loss\_dic[epoch\_num].item()

epoch.test\_accurancy = net\_model.epoch\_logger.epoch\_val\_acc\_dic[epoch\_num].item()

epoch.test\_loss = net\_model.epoch\_logger.epoch\_val\_loss\_dic[epoch\_num].item()

confusion\_matrix = net\_metrics.get\_confusion\_matrix(true\_tags=true\_tags, predicted\_tags=pred\_tags)

for true\_tag, pred\_tags in confusion\_matrix.items():

for pred\_tag, count in pred\_tags.items():

matrix = EpochConfusionMatrix()

matrix.true\_tag = true\_tag

matrix.predicted\_tag = pred\_tag

matrix.count = count

epoch.epoch\_confusion\_matrix.append(matrix)

net.epochs.append(epoch)

net\_filename = '/home/nastya/ftp\_files/{0}.h5'.format(net.name)

net\_model.model.save('/home/nastya/ftp\_files/{0}.h5'.format(net.name))

net.net\_file\_path = net\_filename

db.session.commit()

net = change\_status(net\_id, NetStatus.trained)

db.session.commit()

def init\_training(net\_id, file\_ids, extend\_vocabulary):

net = change\_status(net\_id, NetStatus.created) # type: Net

for file\_id in file\_ids:

file = file\_logic.get(file\_id)

if not any(net\_file for net\_file in net.files if net\_file.id == file\_id):

net.files.append(file)

if not any(model\_file for model\_file in net.embedding\_model.files if model\_file.id not in file\_ids):

net.embedding\_model.files.append(file)

if extend\_vocabulary and not any(voc\_file for voc\_file in net.embedding\_model.vocabulary.files if voc\_file.id == file\_id):

net.embedding\_model.vocabulary.files.append(file)

db.session.commit()

Thread(target=train, args=(net\_id, flask.current\_app.\_get\_current\_object(),)).start()

return net

def get\_confusion\_matrix(epoch\_id):

epoch = db.session.query(Epoch).get(epoch\_id) # type: Epoch

matrix = {}

for element in epoch.epoch\_confusion\_matrix: # type: EpochConfusionMatrix

if element.true\_tag not in matrix:

matrix[element.true\_tag] = {}

matrix[element.true\_tag][element.predicted\_tag] = element.count

return matrix

def delete(net\_id):

net = get(net\_id)

for text\_set in db.session.query(TextSet).filter(TextSet.net\_id == net\_id):

text\_set.net\_id = None

for epoch in net.epochs:

for confusion\_matrix in epoch.epoch\_confusion\_matrix:

db.session.delete(confusion\_matrix)

db.session.delete(epoch)

db.session.delete(net)

db.session.commit()

### 2.4.5. Модуль text\_set.py

from typing import List

from keras.backend import clear\_session

from keras.engine.saving import load\_model

from sqlalchemy import or\_

from sqlalchemy.util import symbol

from dataio.csv import read\_texts\_from\_file

from models import db, TextSet, Link, TextSetLink, TextSetFile, Net, LinkText, FileText, Text

import logic.file as file\_logic

import logic.vocabulary as vocabulary\_logic

import insta\_api

from network.lstm\_categorical import LSTMCategorical

import logic.embedding\_model as embedding\_model\_logic

import numpy as np

def get(text\_set\_id):

text\_set = db.session.query(TextSet).get(text\_set\_id) # type: TextSet

if text\_set is None:

raise ValueError("No text set found by id {0}".format(text\_set\_id))

return text\_set

def add\_urls\_to\_text\_set(text\_set\_id, urls: List[str]):

text\_set = get(text\_set\_id)

for url in urls:

link = Link()

link.url = url

text\_set\_link = TextSetLink()

text\_set\_link.link = link

text\_set.links.append(text\_set\_link)

db.session.commit()

def add\_files\_to\_text\_set(text\_set\_id, file\_ids):

text\_set = get(text\_set\_id) # type: TextSet

for file\_id in file\_ids:

file = file\_logic.get(file\_id)

if any(text\_set\_file for text\_set\_file in text\_set.files if text\_set\_file.file.id == file.id):

continue

text\_set\_file = TextSetFile()

text\_set\_file.file = file

text\_set.files.append(text\_set\_file)

db.session.commit()

def extend(text\_set\_id, app):

with app.app\_context():

text\_set = get(text\_set\_id)

net = text\_set.net # type: Net

vocabulary = net.embedding\_model.vocabulary

clear\_session()

classifier\_model = load\_model(net.net\_file\_path)

doc\_len = classifier\_model.get\_layer(index=0).input\_length

for text\_set\_link in text\_set.links: # type: TextSetLink

comments = insta\_api.get\_all\_comments(text\_set\_link.link.url)

comments = [comment['text'] for comment in comments]

comment\_tokens = vocabulary\_logic.get\_tokens\_by\_vocabulary(vocabulary, comments)

comment\_sequences = np.array(embedding\_model\_logic.get\_sequences(net.embedding\_model\_id, comment\_tokens, doc\_len))

predictions = classifier\_model.predict(comment\_sequences)

for comment, comment\_sequence, prediction in zip(comments, comment\_sequences, predictions):

link\_text = LinkText()

link\_text.id\_from\_link = 1

link\_text.negative\_probability = prediction[0].item()

link\_text.positive\_probability = prediction[1].item()

link\_text.text = comment

text\_set\_link.texts.append(link\_text)

for text\_set\_file in text\_set.files: # type: TextSetFile

# text do not need to be quoted if file is text file (text per line), without separators

comments = read\_texts\_from\_file(

file=text\_set\_file.file.path,

quoted\_fields=text\_set\_file.file.separator,

separator=text\_set\_file.file.separator,

text\_col=text\_set\_file.file.text\_column

)

comment\_tokens = vocabulary\_logic.get\_tokens\_by\_vocabulary(vocabulary, comments)

comment\_sequences = np.array(embedding\_model\_logic.get\_sequences(net.embedding\_model\_id, comment\_tokens, doc\_len))

predictions = classifier\_model.predict(comment\_sequences)

for comment, comment\_sequence, prediction in zip(comments, comment\_sequences, predictions):

file\_text = FileText()

file\_text.negative\_probability = prediction[0].item()

file\_text.positive\_probability = prediction[1].item()

file\_text.text = comment

text\_set\_file.texts.append(file\_text)

db.session.commit()

def wwwinit\_add\_urls\_to\_text\_set(text\_set\_id, urls):

text\_set = get(text\_set\_id)

for url in urls:

link = Link()

link.url = url

text\_set\_link = TextSetLink()

text\_set\_link.link = link

text\_set.links.append(text\_set\_link)

db.session.commit()

def delete(text\_set\_id):

texts = db.session.query(Text).with\_polymorphic([FileText, LinkText])\

.outerjoin(TextSetFile, FileText.text\_set\_file).outerjoin(TextSetLink, LinkText.text\_set\_link)\

.filter(or\_(TextSetFile.text\_set\_id == text\_set\_id, TextSetLink.text\_set\_id == text\_set\_id))

for text in texts:

db.session.delete(text)

text\_set = get(text\_set\_id)

for file in text\_set.files: # type: TextSetFile

db.session.delete(file)

for link in text\_set.links: # type: TextSetLink

db.session.delete(link)

db.session.delete(text\_set)

db.session.commit()

## 2.5. Пакет controllers

### 2.5.1. Модуль enum\_data.py

from flask import Blueprint, request, jsonify

from models import Language, Normalization

enumdata\_bp = Blueprint('enum', \_\_name\_\_)

@enumdata\_bp.route("/languages", methods=["GET"])

def get\_languages():

return jsonify([language.name for language in Language])

@enumdata\_bp.route("/normalizations", methods=["GET"])

def get\_normalizations():

return jsonify([normalization.name for normalization in Normalization])

### 2.5.2. Модуль file.py

from flask import Blueprint, jsonify, send\_file

import logic.file as file\_logic

file\_bp = Blueprint('file', \_\_name\_\_)

@file\_bp.route('/<file\_id>', methods=["GET"])

def get(file\_id):

file\_id = int(file\_id)

file = file\_logic.get(file\_id)

return jsonify(file.to\_dict())

@file\_bp.route('/download/<file\_id>', methods=["GET"])

def download(file\_id):

file\_id = int(file\_id)

file = file\_logic.get(file\_id)

return send\_file(file.path, file.mimetype, True, file.name)

### 2.5.3. Модуль vocabulary.py

from flask import Blueprint, request, jsonify

from models import Vocabulary, db, File

import logic.file as file\_logic

import logic.vocabulary as vocabulary\_logic

vocabulary\_bp = Blueprint('vocabulary', \_\_name\_\_)

@vocabulary\_bp.route("/<vocabulary\_id>", methods=["GET"])

def get\_by\_id(vocabulary\_id):

vocabulary\_id = int(vocabulary\_id)

vocabulary = vocabulary\_logic.get(vocabulary\_id)

return jsonify(vocabulary.to\_dict(\_hide=[]))

@vocabulary\_bp.route("/", methods=["GET"])

def get\_all():

try:

from\_index = request.args.get('from', 0, type=int)

to\_index = request.args.get('to', 9, type=int)

if from\_index > to\_index:

tmp\_index = from\_index

from\_index = to\_index

to\_index = tmp\_index

if to\_index - from\_index > 50:

raise ValueError('Can not return more than 50 net records')

direction = request.args.get('sort\_direction', 'asc')

if direction != 'asc' and direction != 'desc':

raise ValueError('Direction {} is not allowed'.format(direction))

sort\_field\_name = request.args.get('sort\_field', 'updated\_time')

if sort\_field\_name not in ('name', 'updated\_time'):

raise ValueError('Sort field {} is not allowed'.format(sort\_field\_name))

name = request.args.get('name', default=None)

text\_sets = db.session.query(Vocabulary)

if name is not None and name:

text\_sets = text\_sets.filter(Vocabulary.name.contains(name))

text\_sets = {

('name', 'desc'): text\_sets.order\_by(Vocabulary.name.desc()),

('name', 'asc'): text\_sets.order\_by(Vocabulary.name.asc()),

('updated\_time', 'desc'): text\_sets.order\_by(Vocabulary.updated\_time.desc()),

('updated\_time', 'asc'): text\_sets.order\_by(Vocabulary.updated\_time.asc()),

}[(sort\_field\_name, direction)]

all\_count = text\_sets.count()

text\_sets = text\_sets.limit(to\_index-from\_index).offset(from\_index)

return jsonify({'count': all\_count, 'vocabularies': [text\_set.to\_dict(\_hide=["words"]) for text\_set in text\_sets]})

except Exception as ex:

return jsonify({'success': False, 'errors': [str(ex)]})

@vocabulary\_bp.route("/", methods=["PUT"])

def create():

vocabulary = Vocabulary()

vocabulary.from\_dict(\*\*request.json)

db.session.add(vocabulary)

db.session.flush()

db.session.commit()

return jsonify(vocabulary.to\_dict())

@vocabulary\_bp.route("/extend", methods=["POST"])

def extend():

vocabulary\_id = request.form["vocabulary\_id"]

separator = request.form["file"]["separator"]

text\_column = request.form["file"]["text\_column"]

tag\_column = request.form["file"]["tag\_column"]

positive\_tag = request.form["file"]["positive\_tag"]

negative\_tag = request.form["file"]["negative\_tag"]

uploaded\_files = []

for filename, file\_stream in request.files.items():

file = File()

file.separator = separator

file.text\_column = text\_column

file.tag\_column = tag\_column

file.positive\_tag = positive\_tag

file.negative\_tag = negative\_tag

uploaded\_files.append(file\_logic.save(file, file\_stream))

vocabulary = vocabulary\_logic.init\_extending(vocabulary\_id, [file.id for file in uploaded\_files])

return jsonify(vocabulary.to\_dict())

### 2.5.4. Модуль embedding\_method.py

from flask import Blueprint, request, jsonify

from models import EmbeddingMethod, db

embedding\_method\_bp = Blueprint('embedding-method', \_\_name\_\_)

@embedding\_method\_bp.route("/", methods=["GET", "POST"])

def get\_all():

methods = db.session.query(EmbeddingMethod).all()

return jsonify([method.to\_dict(\_hide=[]) for method in methods])

### 2.5.5. Модуль embedding\_model.py

from flask import Blueprint, request, jsonify

import logic.file as file\_logic

import logic.embedding\_model as emb\_model\_logic

from models import EmbeddingModel, db, File, ModelWordVector

embedding\_model\_bp = Blueprint('embedding-model', \_\_name\_\_)

@embedding\_model\_bp.route("/", methods=["GET"])

def get\_all():

methods = db.session.query(EmbeddingModel).all()

data = [method.to\_dict(\_hide=[]) for method in methods]

response = jsonify(data)

return response

@embedding\_model\_bp.route('/<model\_id>', methods=["GET"])

def get(model\_id):

model\_id = int(model\_id)

model = emb\_model\_logic.get(model\_id)

return jsonify(model.to\_dict(\_hide=[]))

@embedding\_model\_bp.route("/", methods=["PUT"])

def create():

model = EmbeddingModel()

json = request.json

for param in json['embedding\_params']:

param['value'] = float(param['value'])

model.from\_dict(\*\*request.json)

db.session.add(model)

db.session.commit()

return jsonify(model.to\_dict())

@embedding\_model\_bp.route("/train", methods=["POST"])

def train():

model\_id = int(request.form["model\_id"])

model = db.session.query(EmbeddingModel).get(model\_id) # type: EmbeddingModel

if model is None:

raise ValueError("No embedding model found by id {0}".format(model\_id))

separator = request.form["separator"]

text\_column = int(request.form["text\_column"])

tag\_column = int(request.form["tag\_column"])

positive\_tag = request.form["positive\_tag"]

negative\_tag = request.form["negative\_tag"]

uploaded\_files = []

for filename, file\_stream in request.files.items():

file = File()

file.separator = separator

file.text\_column = text\_column

file.tag\_column = tag\_column

file.positive\_tag = positive\_tag

file.negative\_tag = negative\_tag

uploaded\_files.append(file\_logic.save(file, file\_stream))

extend\_vocabulary = True if request.form["extend\_vocabulary"] == "true" else False

model = emb\_model\_logic.init\_training(model\_id, [file.id for file in uploaded\_files], extend\_vocabulary)

return jsonify(model.to\_dict())

@embedding\_model\_bp.route("/t-sne", methods=["GET"])

def vector2d\_all():

for model in db.session.query(EmbeddingModel):

emb\_model\_logic.tsne(model.id)

@embedding\_model\_bp.route("/vectors/<model\_id>", methods=["GET"])

def get\_vectors(model\_id):

model\_id = int(model\_id)

word\_vectors = db.session.query(ModelWordVector).filter(ModelWordVector.embedding\_model\_id == model\_id)

response = []

for word\_vector in word\_vectors:

response.append((word\_vector.word.token, word\_vector.x, word\_vector.y))

return jsonify(response)

### 2.5.6. Модуль net.py

from flask import Blueprint, request, jsonify

from datetime import datetime, timedelta

from sqlalchemy import case

import logic.file as file\_logic

import logic.net as net\_logic

from models import EmbeddingModel, db, File, NetModule, Net, NetStatus

from netword\_metrics.confusion\_matrix import get\_class\_precisions, get\_class\_recalls, get\_average\_precision, \

get\_average\_recall

net\_bp = Blueprint('net', \_\_name\_\_)

def to\_datetime(queryDateString):

return datetime.strptime(queryDateString, "%d.%m.%Y")

def to\_net\_status(status):

for index, value in enumerate(NetStatus):

if status == value.name:

return value

return None

def to\_int(int\_str):

return int(int\_str)

@net\_bp.route("/net-modules", methods=["GET"])

def get\_all\_net\_modules():

modules = db.session.query(NetModule).all()

return jsonify([module.to\_dict(\_hide=[]) for module in modules])

@net\_bp.route("/trained", methods=["GET"])

def get\_trained\_only():

nets = db.session.query(Net).filter(Net.status == NetStatus.trained)

return jsonify([net.to\_dict(\_hide=[]) for net in nets])

@net\_bp.route("/", methods=["GET"])

def get\_all():

from\_index = request.args.get('from', 0, type=int)

to\_index = request.args.get('to', 9, type=int)

if from\_index > to\_index:

tmp\_index = from\_index

from\_index = to\_index

to\_index = tmp\_index

if to\_index - from\_index > 50:

raise ValueError('Can not return more than 50 net records')

direction = request.args.get('sort\_direction', 'asc')

if direction != 'asc' and direction != 'desc':

raise ValueError('Direction {} is not allowed'.format(direction))

sort\_field\_name = request.args.get('sort\_field', 'updated\_time')

if sort\_field\_name not in ('updated\_time', 'name', 'status', 'net\_module\_name', 'embedding\_model\_name'):

raise ValueError('Sort field {} is not allowed'.format(sort\_field\_name))

default\_from\_updated\_time = datetime.today() - timedelta(days=7)

from\_updated\_time = request.args.get('from\_updated\_time', default=default\_from\_updated\_time, type=to\_datetime)

default\_to\_updated\_time = datetime.today()

to\_updated\_time = request.args.get('to\_updated\_time', default=default\_to\_updated\_time, type=to\_datetime) + timedelta(days=1)

name = request.args.get('name', default=None)

status = request.args.get('status', default=None, type=to\_net\_status)

net\_module\_id = request.args.get('net\_module\_id', default=None, type=int)

embedding\_model\_name = request.args.get('embedding\_model\_name', default=None)

embedding\_method\_id = request.args.get('embedding\_method\_id', default=None, type=int)

nets = db.session.query(Net)

nets = nets.filter(Net.updated\_time.between(from\_updated\_time, to\_updated\_time))

if name is not None and name:

nets = nets.filter(Net.name.contains(name))

if status is not None:

nets = nets.filter(Net.status == status)

if net\_module\_id is not None:

nets = nets.filter(Net.net\_module\_id == net\_module\_id)

if embedding\_model\_name is not None and embedding\_model\_name:

nets = nets.filter(Net.embedding\_model.name.contains(embedding\_model\_name))

if embedding\_method\_id is not None:

nets = nets.filter(Net.embedding\_model.embedding\_method\_id == embedding\_method\_id)

status\_indexes = {status: index for index, status in enumerate(NetStatus)}

status\_sort\_logic = case(value=Net.status, whens=status\_indexes).label("status")

nets = {

('updated\_time', 'desc'): nets.order\_by(Net.updated\_time.desc()),

('updated\_time', 'asc'): nets.order\_by(Net.updated\_time.asc()),

('name', 'desc'): nets.order\_by(Net.name.desc()),

('name', 'asc'): nets.order\_by(Net.name.asc()),

('status', 'desc'): nets.order\_by(status\_sort\_logic),

('status', 'asc'): nets.order\_by(status\_sort\_logic),

('net\_module\_name', 'desc'): nets.join(NetModule, Net.net\_module).order\_by(NetModule.name.desc()),

('net\_module\_name', 'asc'): nets.join(NetModule, Net.net\_module).order\_by(NetModule.name.asc()),

('embedding\_model\_name', 'desc'): nets.join(EmbeddingModel, Net.embedding\_model).order\_by(EmbeddingModel.name.desc()),

('embedding\_model\_name', 'asc'): nets.join(EmbeddingModel, Net.embedding\_model).order\_by(EmbeddingModel.name.asc()),

}[(sort\_field\_name, direction)]

all\_count = nets.count()

nets = nets.limit(to\_index-from\_index).offset(from\_index)

return jsonify({'count': all\_count, 'nets': [net.to\_dict(\_hide=[]) for net in nets]})

@net\_bp.route('/<net\_id>', methods=["GET"])

def get(net\_id):

net\_id = int(net\_id)

net = net\_logic.get(net\_id)

net\_info = net.to\_dict(\_hide=["epochs.epoch\_confusion\_matrix"])

for epoch in net.epochs:

epoch\_info = next(epoch\_info for epoch\_info in net\_info["epochs"] if epoch\_info["id"] == epoch.id)

epoch\_info["confusion\_matrix"] = net\_logic.get\_confusion\_matrix(epoch.id)

epoch\_info["class\_precisions"] = get\_class\_precisions(epoch\_info["confusion\_matrix"])

epoch\_info["class\_recalls"] = get\_class\_recalls(epoch\_info["confusion\_matrix"])

epoch\_info["average\_precision"] = get\_average\_precision(epoch\_info["confusion\_matrix"])

epoch\_info["average\_recall"] = get\_average\_recall(epoch\_info["confusion\_matrix"])

return jsonify(net\_info)

@net\_bp.route('/<net\_id>', methods=["DELETE"])

def delete(net\_id):

net\_id = int(net\_id)

try:

net = net\_logic.delete(net\_id)

except:

return jsonify({'success': False})

return jsonify({'success': True})

@net\_bp.route("/", methods=["PUT"])

def create():

net = Net()

net.from\_dict(\*\*request.json)

db.session.add(net)

db.session.commit()

return jsonify(net.to\_dict(\_hide=[]))

@net\_bp.route("/train", methods=["POST"])

def train():

net\_id = int(request.form["net\_id"])

net = net\_logic.get(net\_id) # type: Net

separator = request.form["separator"]

text\_column = int(request.form["text\_column"])

tag\_column = int(request.form["tag\_column"])

positive\_tag = request.form["positive\_tag"]

negative\_tag = request.form["negative\_tag"]

uploaded\_files = []

for filename, file\_stream in request.files.items():

file = File()

file.separator = separator

file.text\_column = text\_column

file.tag\_column = tag\_column

file.positive\_tag = positive\_tag

file.negative\_tag = negative\_tag

uploaded\_files.append(file\_logic.save(file, file\_stream))

extend\_vocabulary = True if request.form["extend\_vocabulary"] == "true" else False

net = net\_logic.init\_training(net.id, [file.id for file in uploaded\_files], extend\_vocabulary)

net\_info = net.to\_dict(\_hide=[])

return jsonify(net\_info)

### 2.5.7. Модуль text\_set.py

import re

from threading import Thread

import flask

from flask import Blueprint, request, jsonify

from sqlalchemy import or\_

import logic.file as file\_logic

import logic.text\_set as text\_set\_logic

from models import db, File, TextSet, Text, LinkText, FileText, TextSetFile, TextSetLink

text\_set\_bp = Blueprint('text-set', \_\_name\_\_)

@text\_set\_bp.route("/", methods=["GET"])

def get\_all():

from\_index = request.args.get('from', 0, type=int)

to\_index = request.args.get('to', 9, type=int)

if from\_index > to\_index:

tmp\_index = from\_index

from\_index = to\_index

to\_index = tmp\_index

if to\_index - from\_index > 50:

raise ValueError('Can not return more than 50 net records')

direction = request.args.get('sort\_direction', 'asc')

if direction != 'asc' and direction != 'desc':

raise ValueError('Direction {} is not allowed'.format(direction))

sort\_field\_name = request.args.get('sort\_field', 'id')

if sort\_field\_name not in ('name', 'id'):

raise ValueError('Sort field {} is not allowed'.format(sort\_field\_name))

name = request.args.get('name', default=None)

text\_sets = db.session.query(TextSet)

if name is not None and name:

text\_sets = text\_sets.filter(TextSet.name.contains(name))

text\_sets = {

('name', 'desc'): text\_sets.order\_by(TextSet.name.desc()),

('name', 'asc'): text\_sets.order\_by(TextSet.name.asc()),

('id', 'desc'): text\_sets.order\_by(TextSet.id.desc()),

('id', 'asc'): text\_sets.order\_by(TextSet.id.asc()),

}[(sort\_field\_name, direction)]

all\_count = text\_sets.count()

text\_sets = text\_sets.limit(to\_index-from\_index).offset(from\_index)

return jsonify({'count': all\_count,

'text\_sets': [text\_set.to\_dict(\_hide=["files", "links"]) for text\_set in text\_sets]})

@text\_set\_bp.route('/<text\_set\_id>', methods=["GET"])

def get(text\_set\_id):

text\_set\_id = int(text\_set\_id)

text\_set = text\_set\_logic.get(text\_set\_id)

text\_set\_info = text\_set.to\_dict(\_hide=["files", "links"])

return jsonify(text\_set\_info)

@text\_set\_bp.route('/texts/<text\_set\_id>', methods=["GET"])

def get\_texts(text\_set\_id):

text\_set\_id = int(text\_set\_id)

from\_index = request.args.get("from", 0, type=int)

count = request.args.get("count", 50, type=int)

direction = request.args.get('sort\_direction', 'asc')

direction = direction if direction else 'asc'

if direction != 'asc' and direction != 'desc':

raise ValueError('Direction {} is not allowed'.format(direction))

sort\_field\_name = request.args.get('sort\_field', 'source')

sort\_field\_name = sort\_field\_name if sort\_field\_name else 'source'

if sort\_field\_name not in ('text', 'tag', 'source'):

raise ValueError('Sort field {} is not allowed'.format(sort\_field\_name))

tag\_key = request.args.get('tag', default=-1, type=int)

if tag\_key not in (-1, 0, 1, 2):

raise ValueError('Tag {} is not allowed'.format(tag\_key))

source\_key = request.args.get('source', default='')

texts = db.session.query(Text).with\_polymorphic([FileText, LinkText])\

.outerjoin(TextSetFile, FileText.text\_set\_file).outerjoin(TextSetLink, LinkText.text\_set\_link)\

.filter(or\_(TextSetFile.text\_set\_id == text\_set\_id, TextSetLink.text\_set\_id == text\_set\_id))

all\_count = texts.count()

positive\_count = texts.filter(Text.negative\_probability < Text.positive\_probability).count()

negative\_count = texts.filter(Text.negative\_probability > Text.positive\_probability).count()

if tag\_key == 0:

texts = texts.filter(Text.negative\_probability > Text.positive\_probability)

if tag\_key == 1:

texts = texts.filter(Text.negative\_probability < Text.positive\_probability)

if tag\_key == 2:

texts = texts.filter(Text.negative\_probability == Text.positive\_probability)

if source\_key:

texts = texts.filter(or\_(

TextSetFile.file.name.contains(source\_key),

TextSetLink.link.url.contains(source\_key)

))

if sort\_field\_name == 'text' and direction == 'asc':

texts = texts.order\_by(Text.text.asc())

if sort\_field\_name == 'text' and direction == 'desc':

texts = texts.order\_by(Text.text.desc())

filtered\_count = texts.count()

texts = texts.limit(count).offset(from\_index)

texts\_data = []

for text in texts:

text\_data = {

'id': text.id,

'text': text.text,

'positive\_probability': text.positive\_probability,

'negative\_probability': text.negative\_probability

}

if isinstance(text, FileText):

text\_data['file'] = text.text\_set\_file.file.to\_dict(\_hide=[])

if isinstance(text, LinkText):

text\_data['link'] = text.text\_set\_link.link.to\_dict(\_hide=[])

texts\_data.append(text\_data)

response = {

'all\_count': all\_count,

'positive\_count': positive\_count,

'negative\_count': negative\_count,

'filtered\_count': filtered\_count,

'filtered\_texts': texts\_data

}

return jsonify(response)

@text\_set\_bp.route("/create", methods=["PUT"])

def create():

text\_set = TextSet()

text\_set.from\_dict(\*\*request.json)

db.session.add(text\_set)

db.session.commit()

return jsonify(text\_set.to\_dict(\_hide=[]))

@text\_set\_bp.route('/<text\_set\_id>', methods=["DELETE"])

def delete(text\_set\_id):

text\_set\_id = int(text\_set\_id)

try:

net = text\_set\_logic.delete(text\_set\_id)

except:

return jsonify({'success': False})

return jsonify({'success': True})

@text\_set\_bp.route("/add-urls/<text\_set\_id>", methods=["POST"])

def add\_urls(text\_set\_id):

text\_set\_id = int(text\_set\_id)

text\_set = text\_set\_logic.get(text\_set\_id)

urls = request.json.get("urls", None)

if urls is None:

raise ValueError("No URLs provided")

text\_set\_logic.add\_urls\_to\_text\_set(text\_set\_id, urls)

return jsonify(text\_set.to\_dict(\_hide=[]))

@text\_set\_bp.route("/add-files/<text\_set\_id>", methods=["POST"])

def add\_files(text\_set\_id):

text\_set\_id = int(text\_set\_id)

text\_set = text\_set\_logic.get(text\_set\_id)

uploaded\_files = []

for filename, file\_stream in request.files.items():

fileindex = re.search("file\\[(?P<index>\d+)\\]", filename).group('index')

text\_column = request.form.get("text\_column[{}]".format(fileindex), '')

text\_column = int(text\_column) if text\_column else 1

separator = request.form.get("separator[{}]".format(fileindex), '')

separator = separator if separator else ','

file = File()

file.separator = separator

file.text\_column = text\_column

uploaded\_files.append(file\_logic.save(file, file\_stream))

text\_set\_logic.add\_files\_to\_text\_set(text\_set\_id, [file.id for file in uploaded\_files])

return jsonify(text\_set.to\_dict(\_hide=[]))

@text\_set\_bp.route("/analyze/<text\_set\_id>", methods=["POST"])

def extend(text\_set\_id):

text\_set\_id = int(text\_set\_id)

text\_set = text\_set\_logic.get(text\_set\_id)

Thread(target=text\_set\_logic.extend, args=(text\_set\_id, flask.current\_app.\_get\_current\_object(),)).start()

return jsonify(text\_set.to\_dict(\_hide=[]))

2.6. Исполняемый модуль server.py

from flask import Flask

from flask\_cors import CORS

SQLALCHEMY\_DATABASE\_URI = ‘’ # there is confidential info

def create\_app\_models\_only():

import models

app = Flask(\_\_name\_\_)

app.config['SQLALCHEMY\_DATABASE\_URI'] = SQLALCHEMY\_DATABASE\_URI

models.init\_app(app)

return app

def create\_app():

import models

import controllers

app = Flask(\_\_name\_\_)

app.config['SQLALCHEMY\_DATABASE\_URI'] = SQLALCHEMY\_DATABASE\_URI

CORS(app)

models.init\_app(app)

controllers.init\_app(app)

return app

if \_\_name\_\_ == '\_\_main\_\_':

server\_app = create\_app()

server\_app.run(host='localhost', port=5000, debug=True)