Федеральное государственное образовательное бюджетное учреждение
высшего профессионального образования
«Финансовый университет при Правительстве Российской Федерации»

Департамент анализа данных, принятия решения и финансовых технологий

Курсовая работа

по дисциплине **"Технологии анализа данных и машинное обучение"** на тему:

"Разработка подходов и алгоритмов для сбора и разметки лингвистических и семантических наборов данных большого объема"

Вид исследуемых данных: Банковские транзакции

Выполнил:
студент группы ПМ17-2
Бахматов А. В.
Научный руководитель:
доцент
Макрушин Сергей Вячеславович

Москва 2020

Содержание

Введение

В данный момент технологии позволяют записывать и хранить массивные объемы данных о самых разных аспектах мира. Еженедельный приток информации на отдельных сайтах числится в терабайтах. Один из основных способов их структуризации и анализа - разметка, то есть разделение на категории. Размечать данные может либо человек, либо созданная для этой цели модель. Особенность больших данных состоит в том, что они по определению слишком массивны для того, чтобы процессом разметки полностью занимались люди. Поэтому вручную пытаться анализировать эту информацию просто невозможно (не считая частичный анализ), и необходимо для этого использовать модели машинного обучения.

В настоящее время машинное обучение стало флагманом IT-индустрии, о нем наслышаны все, его внедряют в самых разных сферах общества. Исключением не стала и банковская сфера: банки располагают большим количеством информации о своей деятельности и желают использовать ее для корректировок своей политики, принятия прибыльных решений.

Целью данной работы является классификация записей банковских транзакций (данные Газпромбанка) по текстовым данным и дополнительным признакам, используя модели машинного обучения. Достижение этой цели позволит банку автоматически группировать транзакции в группы, проводить над ними детальный анализ для дальнейшего принятия решений в управлении банком.

Работа будет реализована на языке программирования Python 3.7 и будет состоять из попытки разметки данных с учителем (то есть на основе небольшой размеченной выборки) при помощи нескольких моделей: метода опорных векторов и искусственных сверточных нейронных сетей.

1 Теоретическая часть

1.1 Задачи обработки естественного языка и ее проблематика

Natural language processing (NLP) - это набор подходов компьютерной обработки естественных языков. Его основные задачи[1]:

1) классификация текста: отнесение текста к одному из заранее определенных классов. Примеры: фильтр спама, анализ тональности, определение темы текста;

2) кластеризация текста: деление текстов на группы (кластеры). Примеры: агрегация новостей, рекомендательные системы;

3) машинный перевод;

4) текстовый анализ. Примеры: распознавание именованных сущностей (при имеющемся наборе сущностей, отнести каждое слово к одному из них), нахождение семантических связей между словами;

5) распознавание речи;

6) проверка правописания;

7) генерация текста.

Основные проблемы, возникающие при обработке языка:

1) Контекст слов: у слов зачастую имеется несколько значений, которые зависят от контекста слова в предложении, а так же от времени, в которое текст был написан, и от интенции пишущего (сарказм);

2) Опечатки, мусор в тексте (например, в html), множество форм у одного слова.

1.2 Этапы обработки текста

При создании алгоритмов обработки текста обычно используют следующие процедуры:

1) Токенизация: разбиение текста на составные части - токены (обычно слова, но в некоторых задачах имеет смысл выделять знаки препинания как отдельные токены, например, при обработке текста с форумов и социальных сетей);

2) Очистка незначащих слов (стоп-слов);

3) Лемматизация: приведение слов к их леммам - начальным формам;

4) Стемминг: приведение слов к их основам. Более простой аналог лемматизации (как пример: при лемматизации слово "сделали" будет преобразовано в "сделать", а при стемминге - в "сделал");

5) Исправление опечаток;

6) Представление текста (encoding) - перевод текста в более простую для обработки форму. Популярные виды представления текста:

а) мешок слов - каждое предложение выражается как вектор, показывающий, сколько раз каждое слово из корпуса встречается в предложении (т.е. показывается, какие слова находятся в данном предложении, а какие нет);

б) TF-IDF[2] - похож на мешок слов, но вместо количества у каждого слова, которое есть в конкретном предложении, имеется величина, расчитываемая как TF*IDF, где TF (Term Frequency) - частота встречаемости слова в предложении, а IDF (Inverse Document Frequency) - обратная частота встречаемости слова во всем корпусе. Величина IDF позволяет игнорировать слова, которые встречаются очень часто, а следовательно, не несут информации. Таким образом, величина TF-IDF вычисляет "полезность слова";

в) представление метками (Label Encoding) - каждый токен представляется уникальной числом;

г) представление унитарным кодом (One-hot Encoding) - аналог Label Encoding, но вместо числа - вектор-индикатор;

д) векторное представление токенов (ембеддинги)[3]: каждый токен представляется как n-мерный вектор, чье значение отображает его связи с остальными токенами в корпусе (т.е. чем ближе векторы разных токенов, тем они более похожи по смыслу).

Среди различных видов представлений текста, мешок слов и TF-IDF не сохраняют порядок слов в предложении и смысла слов, но при этом довольно просто реализуются. Эмбеддинги имеют более сложную реализацию, но сохраняют порядок слов, и каждое слово имеет относительный смысл.

1.3 Модели классификации

Для классификации текста используется множество моделей:

1) Наивный байесовский классификатор - один из самых простых классификаторов, который для предсказания классов считает частоты слов и классов в корпусе. Считает вероятность появления каждого слова независимым от других;

2) Логистическая регрессия: простой вид искусственной нейронной сети, имеющей лишь входной и выходной слои;

3) Метод опорных векторов[4]: разделяет пространство векторов гиперплоскостью. С точки зрения качества может работать на уровне нейронных сетей со сложной архитектурой, и при этом не имеет большого количества гиперпараметров;

4) Нейронные сети прямого распространения: принцип работы тот же, что и у логистической регрессии, но имеет скрытые слои;

5) Сверточные нейронные сети: имеют блоки свертки, которые проводят

слои перед ней через фильтр (с точки зрения математики - умножают на матрицу). Обычно используются для работы с фото, но полезны также для текста (ибо при представлении текста ембеддингами входные данные имеют вид двухмерной матрицы);

6) Рекуррентные нейронные сети: главная особенность - передача данных между узлами на одном и том же слое, из-за чего возможно сохранять контекст. Из-за этого требуют больше ресурсов для обучения.

В данной работе для классификации будет использоваться метод опорных векторов и 1-мерная сверточная нейронная сеть (так как они показали себя лучше всего на имеющемся датасете).

2 Построение моделей

2.1 Постановка задачи и датасет

Датасет имеет следующие признаки: documentid, to_acc, name_to, from_acc, name_from, purp, sum, payer_tax_num, payee_tax_num, payer_bank_bic_cd, payee_bank_bic_cd, inn_same, bic_same, class, payer_ОКВЭД_main, payee_ОКВЭД_main. Из интересующих нас признаков имеются purp - текстовое описание транзакции (например "За электроснабжение (окончательный расчет, за январь), по договору xxxxxxxx от 01.01.2018. С/ф N 1111111111111/12/00000 от 31.01.2019Сумма 98898-65В т.ч. НДС 20% 11111-11"), to_acc и from_acc - на какой и с какого счета производится транзакция, payer_ОКВЭД_main и payee_ОКВЭД_main - счета ОКВЭД (Общероссийский классификатор видов экономической деятельности).

Транзакцию нам нужно отнести к одному из классов: Комиссии банку, Суды, Зарплата, Аренда, Валюта, Инкассирование, Эквайринг, Возврат, Штрафы прочие, Штрафы государство, Проценты по кредиту, Страховое возмещение, Налоги НДФЛ, Налоги прочие, Налоги прибыль, Налоги НДС, Депозит, Вексель, Кредит, Оплата ФЛ, Выручка, Пополнение счета, Перевод на ФЛ, Займ, Выплаты соцхарактера, Взыскание с ФЛ, Страховая премия, Обеспечение, Пожертвования и благотворительность, Дивиденды, Прочее.

Датасет включает в себя 3300 размеченных строк (20000 всего).

2.2 Сверточная нейронная сеть

2.2.1 Предобработка

Загрузим датасет:

```python
def load_data(path):
  data = pd.read_csv(path)
  data.rename({'Ответ 2 из 3:':'class'}, axis=1, inplace=True)
  return data

train_path = get_drive_path('data\\train_data_3300_win.txt')
test_path = get_drive_path('data\\test_data_3300_win.txt')

train_data = load_data(train_path)
test_data = load_data(test_path)
all_data = pd.concat((train_data, test_data))
```

Номер класса	Частота
26	824
28	297
32	283
8	221
37	158
38	143
36	107
6	104
39	89
9	78
24	67
13	66
27	38
12	37
19	31
22	28
29	27
14	21
30	19
33	17
17	15

Figure 1: Частоты классов (названия представлены числами)

Предобработка текста состоит из очистки от дат (get_clear_text), токенизации по словам, замены токенов из 20 цифр в "карта", очистки от чисел

и нелексикографических символов

```
def remove_noise(tokens, stop_words = []):

    cleaned_tokens = []

    for token in tokens:

        #номера карт (весь токен это 20 цифр)
        token = "карта" if re.search(r"^\d{20}$", token) else token
        #убираем числа
        token = re.sub(r'\d', '', token)
        #очистка нелексигографических символов
        token = re.sub(r'\W', '', token)

        #убираем токены состоящие из одного символа или являющиеся стоп-словом
        if len(token) > 1 and token.lower() not in stop_words:
            cleaned_tokens.append(token.lower())

    return cleaned_tokens

def get_cleaned_features(features, stop_words):
    sentences_tokens = [word_tokenize(get_clear_text(sentence)) for sentence in
    ↪  features]
    return [remove_noise(tokens, stop_words) for tokens in sentences_tokens]
```

Функция для предобработки данных (очистки, разбиения на подвыборки, приведения слов к числам)

```
def get_data_for_nn(data, no_other_features=False, max_len=None, all_data=None,
↪  test=False, tokenizer=None, encoder_dict=None, y_encoder=None,
↪  path_to_shortenings_file=None, fix_spelling=False):

  stop_words = stopwords.words('russian')

  X, y = data['purp'], data['class']
  X_text = X
  #чистка текста
  X = get_cleaned_features(X, stop_words, path_to_shortenings_file, fix_spelling)
```

```
if not test and not no_other_features:
  y, y_encoder = encode(y, OneHotEncoder, all_data['class'])
  other_features, encoder_dict = extract_features(data)
  other_features, X, y = dropna(other_features, X, y)
  #валидационная выборка
  X, X_val, y, y_val, other_features, other_features_val, X_text, X_text_val =
  ↪ train_test_split(X, y, other_features, X_text, test_size=0.2, stratify=y,
  ↪ random_state=42)
  #слова в числа
  tokenizer = Tokenizer(num_words=10000)
  tokenizer.fit_on_texts(X)
  X = tokenizer.texts_to_sequences(X)
  X_val = tokenizer.texts_to_sequences(X_val)
  max_len = len(max(X, key=len))
  #паддинг
  X = pad_sequences(X, padding='post', maxlen=max_len)
  X_val = pad_sequences(X_val, padding='post', maxlen=max_len)

  return X, X_val, y, y_val, X_text, X_text_val, other_features,
  ↪ other_features_val, tokenizer, encoder_dict, y_encoder

elif not test:
  y, y_encoder = encode(y, OneHotEncoder, all_data['class'])
  #валидационная выборка
  X, X_val, y, y_val, X_text, X_text_val = train_test_split(X, y, X_text,
  ↪ test_size=0.2, stratify=y, random_state=42)
  #слова в числа
  tokenizer = Tokenizer(num_words=10000)
  tokenizer.fit_on_texts(X)
  X = tokenizer.texts_to_sequences(X)
  X_val = tokenizer.texts_to_sequences(X_val)
  max_len = len(max(X, key=len))
  #паддинг
  X = pad_sequences(X, padding='post', maxlen=max_len)
  X_val = pad_sequences(X_val, padding='post', maxlen=max_len)

  return X, X_val, y, y_val, X_text, X_text_val, tokenizer, y_encoder

elif test and not no_other_features:
  y = encode(y, y_encoder)
  other_features = extract_features(data, encoder_dict)
```

```
    #слова в числа
    X = tokenizer.texts_to_sequences(X)
    #паддинг
    X = pad_sequences(X, padding='post', maxlen=max_len)

    return X, y, other_features, X_text

  else:
    y = encode(y, y_encoder)
    #слова в числа
    X = tokenizer.texts_to_sequences(X)
    #паддинг
    X = pad_sequences(X, padding='post', maxlen=max_len)

    return X, y, X_text
```

Функция для загрузки ембеддингов с помощью предобученной модели fasttext

```
def load_fast_text_pretrained_oov(corpus, output_file_path, path_to_model,
 ↪  tokenizer, number_of_dimensions):
    """
    Загрузить embedding_matrix из предобученных векторов fasttext
    """
    try:
        embedding_matrix = load_embedding(output_file_path, tokenizer)
    except FileNotFoundError:
        ft_model = FastTextKeyedVectors.load(path_to_model)
        model_to_vec_file(corpus, output_file_path, ft_model.__getitem__,
         ↪  number_of_dimensions)
        embedding_matrix = model_to_matrix(ft_model.__getitem__, tokenizer,
         ↪  number_of_dimensions)

    return embedding_matrix
```

Создадим тренировочную и валидационную выборку, сформируем дополнительные признаки как one-hot матрицу нграмм, загрузим предобученную fasttext модель out-of-vocabulary

```
X_train, X_val, y_train, y_val, X_text_train, X_text_val, tokenizer, y_encoder =
↪   get_data_for_nn(train_data, no_other_features=True, all_data=all_data)
cleaned_train_data =
↪   train_data[['purp','class']].copy().rename({'class':'class_number'}, axis=1)

cleaned_train_data['purp'] = [' '.join(sentence) for sentence in
↪   get_cleaned_features(train_data['purp'], stop_words)]
ngrams = utils.NGramsCounter(cleaned_train_data).choose_significant_ngrams(2, 9,
↪   0.7).columns
other_features_train = np.array([[int(ngram in ' '.join(sentence)) for ngram in
↪   ngrams] for sentence in get_cleaned_features(X_text_train, stop_words)])
other_features_val = np.array([[int(ngram in ' '.join(sentence)) for ngram in
↪   ngrams] for sentence in get_cleaned_features(X_text_val, stop_words)])

vocab_size = len(tokenizer.word_index)+1
max_len = X_train.shape[1]
output_number = y_train.shape[1]

path_to_model =
↪   get_drive_path("data\\fasttext\\geowac_tokens_none_fasttextskipgram_300\\model.model")
output_file_path =
↪   get_drive_path(f"data\\fasttext\\ft_geowac_oov_300_{cleaned_corpus_version}.vec")

embedding_matrix = eb.load_fast_text_pretrained_oov(cleaned_corpus,
↪   output_file_path, path_to_model, tokenizer, number_of_dimensions)
```

2.2.2 Построение и обучение модели

Построение и обучение будет реализовано при помощи библиотеки keras. Определим функции f1 micro и f1 weighted для оценки качества модели

```
import keras.backend as K
from sklearn.metrics import f1_score

def f1(true, pred, average='micro', loss=False): #shapes (batch, output_number)
    if not loss:
        predLabels = K.argmax(pred, axis=-1)
        pred = K.one_hot(predLabels, output_number)

    ground_positives = K.sum(true, axis=0)       # = TP + FN
```

```
    pred_positives = K.sum(pred, axis=0)         # = TP + FP
    true_positives = K.sum(true * pred, axis=0)  # = TP

    if average == 'micro':
        true_positives = K.sum(true_positives)
        pred_positives = K.sum(pred_positives)
        ground_positives = K.sum(ground_positives)

    precision = (true_positives + K.epsilon()) / (pred_positives + K.epsilon())
    recall = (true_positives + K.epsilon()) / (ground_positives + K.epsilon())
        #both = 1 if ground_positives == 0 or pred_positives == 0
        #shape (output_number,)

    f1 = 2 * (precision * recall) / (precision + recall + K.epsilon())
        #not sure if this last epsilon is necessary
        #mathematically not, but maybe to avoid computational instability
        #still with shape (output_number,)

    if average == 'weighted':
        f1 = f1 * ground_positives / K.sum(ground_positives)
        f1 = K.sum(f1)

    return f1 if not loss else 1-f1

def f1_micro(true, pred):
    return f1(true, pred, average='micro', loss=False)

def f1_weighted(true, pred):
    return f1(true, pred, average='weighted', loss=False)
```

Построим модель: текстовые данные в виде числовых последовательностей поступают преобразуются в ембеддинги, получившаяся матрица поступает в сверточный слой (с дропаутом для регуляризации), проходит через слои макс пулинга для уменьшения размерности, соединяется с дополнительными признаками и получившийся слой соединяется с выходным. Дополнительные признаки - one-hot матрица нахождения нграмм, коррелирующих с классами на 0.7 или более.

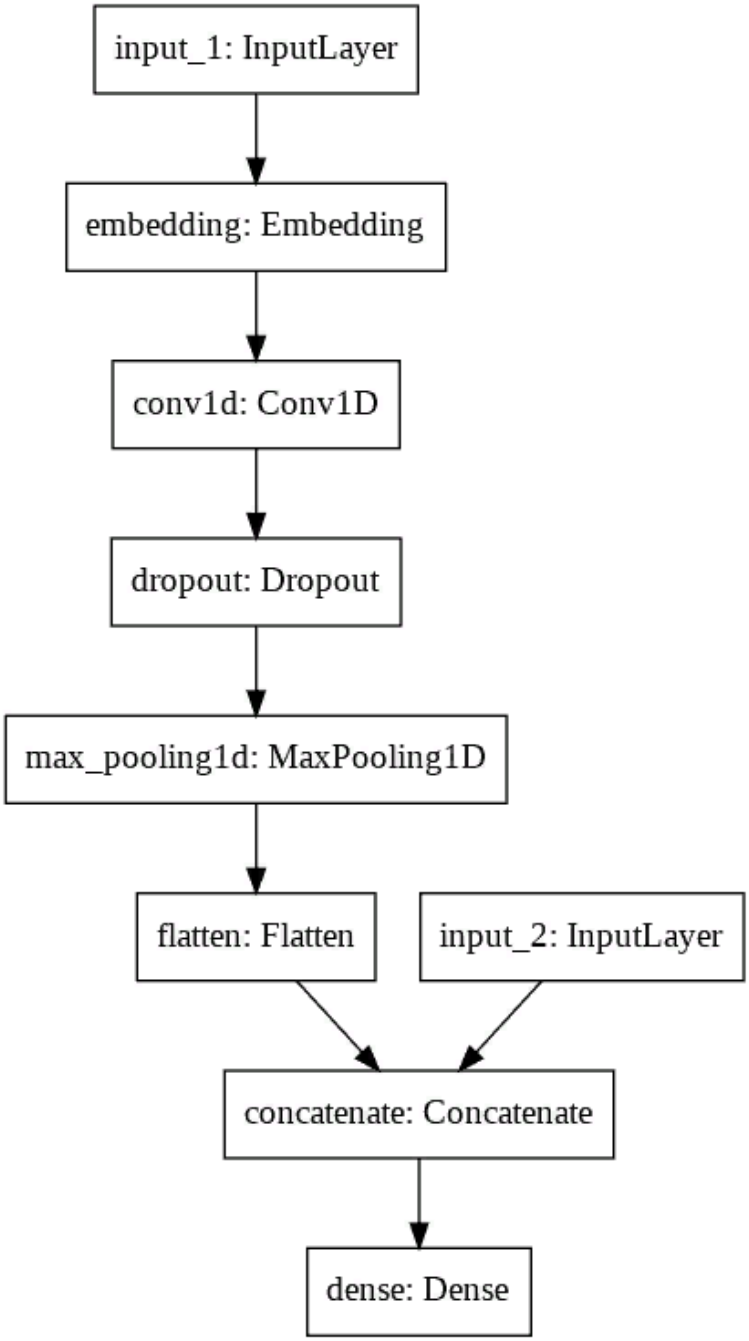

Figure 2: Одномерная сверточная нейронная сеть с двумя входными слоями

```
from keras.layers import Input, Concatenate, Flatten, Dense
from keras.models import Model

loss = 'categorical_crossentropy'
activation = 'softmax'
filters, kernel_size = 10, 5

text_input = Input(shape=(max_len,))
```

```
vector_input = Input(shape=(other_features_train.shape[1],))

embedding_layer = Embedding(input_dim=vocab_size,
                            output_dim=number_of_dimensions,
                            weights=[embedding_matrix],
                            input_length=max_len,
                            trainable=True,
                            embeddings_regularizer=reg.l1(0.0005))(text_input)
conv_layer = Conv1D(filters, kernel_size, activation='relu',
 ↪  kernel_regularizer=reg.l2(0.005))(embedding_layer)
conv_layer = Dropout(0.2)(conv_layer)
conv_layer = MaxPooling1D(10)(conv_layer)
conv_layer = GlobalMaxPooling1D()(conv_layer)

concat_layer = Concatenate()([vector_input, conv_layer])
output = Dense(output_number, activation=activation)(concat_layer)

model = Model(inputs=[text_input, vector_input], outputs=output)
model.compile(optimizer=Adam(learning_rate=0.01), loss=loss,
 ↪  metrics=['acc',f1_micro,f1_weighted])
model.summary()
```

Обучим модель.

```
filepath = get_drive_path("saved models\\saved-model-conv.hdf5")
checkpoint = ModelCheckpoint(filepath, monitor='val_f1_micro', verbose=0,
 ↪  save_best_only=True, mode='max', save_weights_only=True)
early_stop = EarlyStopping(monitor='val_f1_micro', mode='max', verbose=1,
 ↪  patience=20)

callbacks = [checkpoint, early_stop]
batch_size = 128
epochs = 50

history = model.fit([X_train,other_features_train],
                    y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    callbacks=callbacks,
                    validation_data=([X_val,other_features_val], y_val)
                   )
```

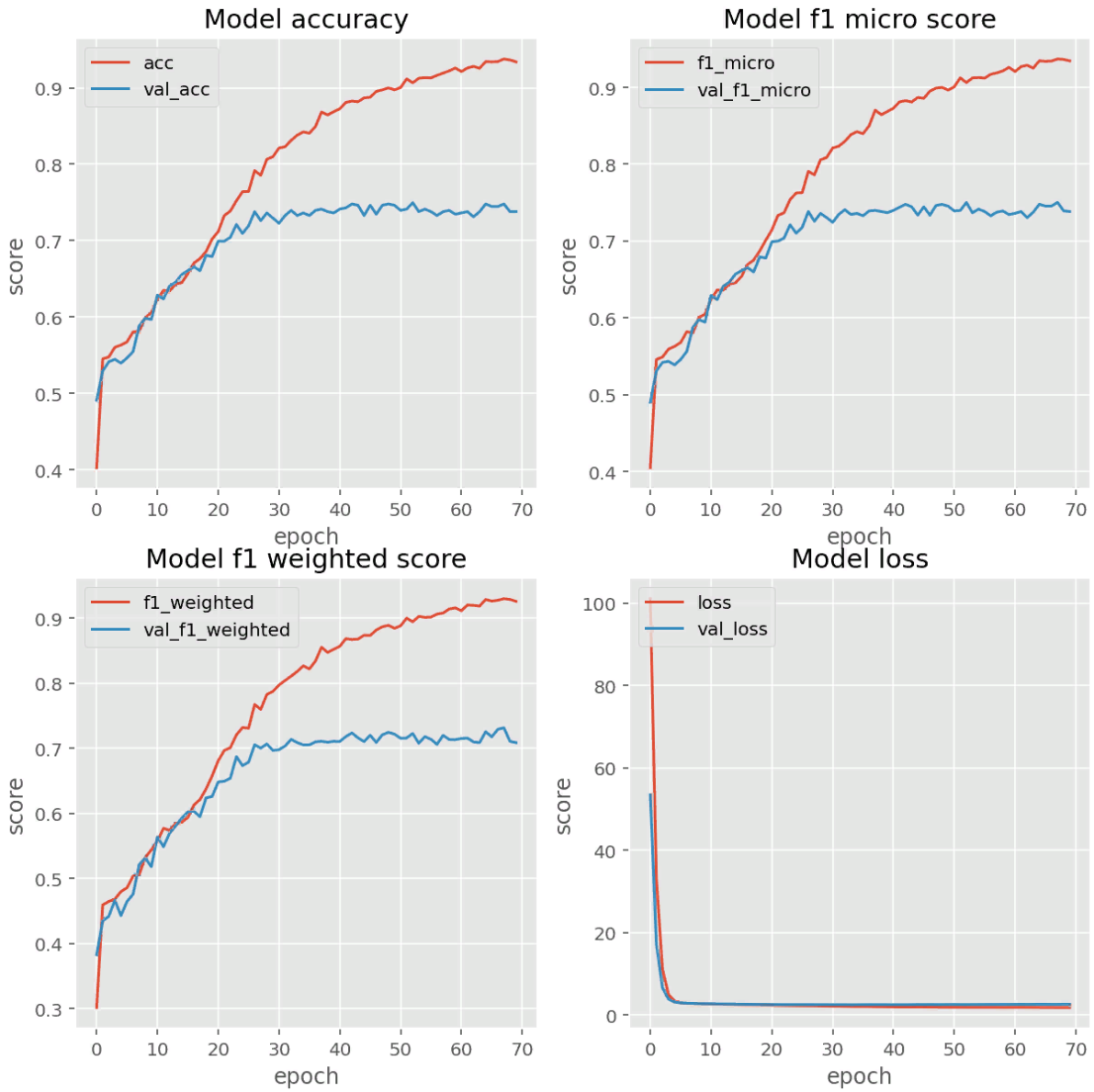

Figure 3: Оценка модели на эпохах обучения

2.3 Метод опорных векторов

2.3.1 Предобработка, построение и обучение модели

На вход методу опорных векторов будет подаваться связка TF-IDF текста и one-hot нграмм, коррелирующих с классами на 0.7 или более.

Для TF-IDF предобработка текста производиться не будет, так как TF-IDF автоматически определяет мусор.

Для матрицы нахождения нграмм почистим данные теми же методами, что и ранее, и сформируем one-hot матрицу.

```
cleaned_train_data = train_data[['purp',target_name]].copy()
stop_words = stopwords.words('russian')

cleaned_train_data['purp'] = [' '.join(sentence) for sentence in
↪  get_cleaned_features(train_data['purp'], stop_words)]
train_ngrams =
↪  utils.NGramsCounter(cleaned_train_data).choose_significant_ngrams(2, 9, 0.7)

cleaned_test_data = test_data[['purp',target_name]].copy()
cleaned_test_data['purp'] = [' '.join(sentence) for sentence in
↪  get_cleaned_features(test_data['purp'], stop_words)]
test_ngrams = np.array([[int(ngram in sentence) for ngram in train_ngrams.columns]
↪  for sentence in cleaned_test_data['purp']])
```

Сформируем TF-IDF матрицу, склеим ее с one-hot матрицей, и обучим SVM линейным ядром.

```
vectorizer = TfidfVectorizer()
train_vectors = vectorizer.fit_transform(train_data['purp'])
train_vectors = np.hstack((train_vectors.toarray(),train_ngrams))
test_vectors = vectorizer.transform(test_data['purp'])
test_vectors = np.hstack((test_vectors.toarray(),test_ngrams))

classifier_svm = LinearSVC()
classifier_svm.fit(train_vectors, train_data[target_name])
```

3 Оценка моделей

3.1 Выбор метрики

Существует множество метрик для оценки качества прогнозов моделей классификации, самая простая - это точность, однако она может быть обманчива (чаще всего это происходит в датасетах с несбалансированным количеством представителей классов). Метрики, использующиеся для вычисления большинства остальных классификационных метрик, это precision и recall, которые показывают, насколько модель подвержена ошибкам первого и второго рода (false negatives and false positives), считаются они по формулам true positivies / (true positivies + false positivies) и true positivies / (true positivies + false negatives) соответственно. В нашей задаче нельзя выделить, что важнее: не совершить ошибку первого или второго рода, поэтому логично

использовать усредненную величину. В качестве такой величины мы будем использовать F1, являющейся сглаженной средней между precision и recall (2 * (precision * recall) / (precision + recall)). Стоит отметить, что эти метрики изначально расчитывают ошибки в бинарной классификации, а для обобщения на классификацию произвольного числа классов используется чаще всего метод один-против-остальных (one-vs-rest): один класс рассматривается как положительный, и все остальные - как отрицательный. Таким образом, у каждого класса будет свой набор recall, precision и f1. Для нахождения среднего f1 мы будем использовать два метода: взвешенный и микро[5]. Взвешенный метод берет среднее взвешеннее значение всех f1 (весом каждого класса является количество его представителей в датасете). Микро глобально считает общее число true positives, false negatives, false positives и на их основе считает общее f1.

3.2 Оценка моделей на тестовой выборке

Для оценки нейронной сети выберем ту обученную модель, у которой наибольшая валидационная f1 micro.

```
conv1d_model_other_features.load_weights(get_drive_path('saved
↪ models\\saved-model-conv-other-features.hdf5'))

X_test, y_test, X_text_test = get_data_for_nn(test_data, no_other_features=True,
↪ all_data=all_data, test=True, tokenizer=tokenizer, y_encoder=y_encoder)
other_features_test = np.array([[int(ngram in ' '.join(sentence)) for ngram in
↪ ngrams] for sentence in get_cleaned_features(X_text_test, stop_words)])

pred = np.round(conv1d_model_other_features.predict([X_test,
↪ other_features_test]))

print('f1 micro', np.round(f1_score(y_test, pred, average='micro'), 4))
print('f1 weighted', np.round(f1_score(y_test, pred, average='weighted'), 4))
```

Получаем f1 micro 0.7672, f1 weighted 0.7296.

Оценка метода опорных векторов:

```
prediction = classifier_svm.predict(test_vectors)

print('f1 micro:', f1_score(test_data[target_name], prediction, average='micro'))
print('f1 weighted:', f1_score(test_data[target_name], prediction,
↪ average='weighted'))
```

f1 micro 0.8217, f1 weighted 0.8098.

Заключение

Для задачи разметки лингвистических данных большого объема было выбрано использовать подходы NLP и машинного обучения: на размеченной выборке малого размера обучаются модели. Была проведена чистка текстовых данных, перебор разных конфигураций:

ембеддингов: word2vec, fasttext, которые были либо обучены на датасете, либо предобучены;

дополнительных признаков: из столбцов датасета или по нграммам;

архитектур нейронной сети и их гиперпараметров.

В итоге на тестовой выборке лучше всего себя показал метод опорных векторов. Несмотря на это, нейронные сети все равно остаются более перспективным инструментом для прогноза из-за их гибкости в настройке архитектуры и гиперпараметров.

Список использованных источников

[1] Applied Text Analysis with Python by Benjamin Bengfort, Rebecca Bilbro, and Tony Ojeda (O'Reilly). ISBN 978–1–4919630-4-3

[2] Кластеризация текста с помощью K-means и TF-IDF [Электронный ресурс]. URL: https://lambda-it.ru/post/klasterizatsiia-teksta-s-pomoshchiu-k-means-i-tf-i (дата обращения 25.11.2020)

[3] GloVe: Global Vectors for Word Representation [Электронный ресурс]. URL: https://nlp.stanford.edu/projects/glove/ (дата обращения 25.11.2020)

[4] Акбархужаев, С. А. Сравнительный анализ методов Наивного Байеса и SVM алгоритмов при классификации текстовых документов / С. А. Акбархужаев, Н. Н. Абдурахманова. — Текст : непосредственный, электронный // Молодой ученый. — 2019. — № 29 (267). — С. 810. — URL: https://moluch.ru/archive/267/61568/ (дата обращения: 25.11.2020).

[5] sklearn.metrics.f1_score [Электронный ресурс]. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html (дата обращения: 25.11.2020).