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:

"Классификация банковских транзакций методами машинного обучения"

Вид исследуемых данных: Банковские транзакции

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

Москва 2020

Содержание

Введение

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 предобработки текстовых данных, выделения релевантных дополнительных признаков, перевода текстовых данных в векторный вид, выбора модели для классификации, оценки результатов обучения.

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) 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 - 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): 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: разделяет пространство векторов гиперплоскостью. Метод может работать на уровне нейронных сетей со сложной архитектурой, и при этом не имеет большого количество параметров;

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

5) Сверточные нейронные сети: обычно используются для работы с фото, но полезны также для текста (ибо при представлении текста ембеддингами

на входе имеем двухмерную матрицу);

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.

Мы для классификации будем использовать 1-мерную сверточную нейронную сеть.

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 N КП-13-Н/309 от 01.01.2018. С/ф N 3010119120000385/12/00000 от 31.01.2019Сумма 98898-65В т.ч. НДС 20% 16483-11"), 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 и уберем строки в которых есть nan:

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

train_data = pd.read_csv(train_path)
```

```
train_data.rename({'Ответ 2 из 3:':'class'}, axis=1, inplace=True)
train_data = train_data[~np.isnan(train_data['payee_ОКВЭД_main'])]
#класс -1 переименовывается в максимальный класс+1
train_data['class'].replace(-1,train_data['class'].max()+1, inplace=True)

test_data = pd.read_csv(test_path)
test_data.rename({'Ответ 2 из 3:':'class'}, axis=1, inplace=True)
test_data = test_data[~np.isnan(test_data['payee_ОКВЭД_main'])]
test_data['class'].replace(-1,test_data['class'].max()+1, inplace=True)
train_data.shape, test_data.shape
```

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 нелексикографических 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]
```

Почистим текстовые данные, преобразуем таргет в унитарный код

```
X_train, X_test, y_train, y_test = train_data['purp'], test_data['purp'],
↪  train_data['class'], test_data['class']

X_train, X_test = get_cleaned_features(X_train,stop_words),
↪  get_cleaned_features(X_test,stop_words)

y_train, y_test = to_categorical(y_train), to_categorical(y_test)
output_number = y_train.shape[1]
```

Извлечем дополнительные признаки (енкодированная пара to_acc и from_acc, payee_ОКВЭД_main), отберем валидационную выборку для отслеживания переобучения.

```
def extract_features(data, encoder):
  number_of_features = 2
  name_pair = data['to_acc'].astype('str').str[:]+'
   ↪  '+data['from_acc'].astype('str').str[:]

  other_features = np.zeros((data.shape[0], number_of_features))
  other_features[:, 0] = encoder.transform(name_pair)
  other_features[:, 1] = data['payee_ОКВЭД_main']
  #other_features[:, 1] = (data['sum'] - data['sum'].mean())/data['sum'].std()

  return other_features

encoder = LabelEncoder()
encoder.fit(np.unique(np.hstack((
    train_data['to_acc'].astype('str').str[:]+'
     ↪  '+train_data['from_acc'].astype('str').str[:],
    test_data['to_acc'].astype('str').str[:]+'
     ↪  '+test_data['from_acc'].astype('str').str[:]))))
other_features_train = extract_features(train_data, encoder)

X_train, X_val, y_train, y_val, other_features_train, other_features_val =
 ↪  train_test_split(X_train, y_train, other_features_train, test_size=0.2,
 ↪  stratify=y_train, random_state=42)
X_test_text = np.array([' '.join(sentence) for sentence in X_test])
```

Представим текст набором чисел (для доступа к ембеддингам в будущем), приведем эти последовательности чисел к одному размеру (добавив нули в конце векторов).

```
tokenizer = Tokenizer(num_words=10000)
tokenizer.fit_on_texts(X_train)
X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)
X_val = tokenizer.texts_to_sequences(X_val)
vocab_size = len(tokenizer.word_index)+1

max_len = len(max(X_train,key=len))

X_train = pad_sequences(X_train, padding='post', maxlen=max_len)
X_test = pad_sequences(X_test, padding='post', maxlen=max_len)
X_val = pad_sequences(X_val, padding='post', maxlen=max_len)
```

Используем предобученную модель FastText чтобы выразить слова в нашем датасете

```
from gensim.models.fasttext import FastTextKeyedVectors

path_to_model =
↪  get_drive_path("data\\fasttext\\geowac_tokens_none_fasttextskipgram_300\\model.model")
output_file_path = get_drive_path("data\\fasttext\\ft_geowac_oov_300.vec")

embedding_matrix = load_embedding(output_file_path, tokenizer)
```

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, последовательность ембеддингов поступает в сверточный слой, потом проходит через слои макс пулинга, соединяется с дополнительными признаками и получившийся слой соединяется с выходным)

```
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()
```

```
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_27 (InputLayer)        [(None, 26)]              0
_________________________________________________________________________________________________
embedding_13 (Embedding)     (None, 26, 300)           1746600     input_27[0][0]
_________________________________________________________________________________________________
conv1d_13 (Conv1D)           (None, 22, 10)            15010       embedding_13[0][0]
_________________________________________________________________________________________________
dropout_13 (Dropout)         (None, 22, 10)            0           conv1d_13[0][0]
_________________________________________________________________________________________________
max_pooling1d_13 (MaxPooling1D) (None, 2, 10)          0           dropout_13[0][0]
_________________________________________________________________________________________________
input_28 (InputLayer)        [(None, 2)]               0
_________________________________________________________________________________________________
global_max_pooling1d_4 (GlobalM (None, 10)             0           max_pooling1d_13[0][0]
_________________________________________________________________________________________________
concatenate_13 (Concatenate)    (None, 12)             0           input_28[0][0]
                                                                   global_max_pooling1d_4[0][0]
_________________________________________________________________________________________________
dense_13 (Dense)             (None, 40)                520         concatenate_13[0][0]
=================================================================================================
Total params: 1,762,130
Trainable params: 1,762,130
Non-trainable params: 0
```

Figure 2: Одномерная сверточная нейронная сеть с двумя входными слоями

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)
                   )
```

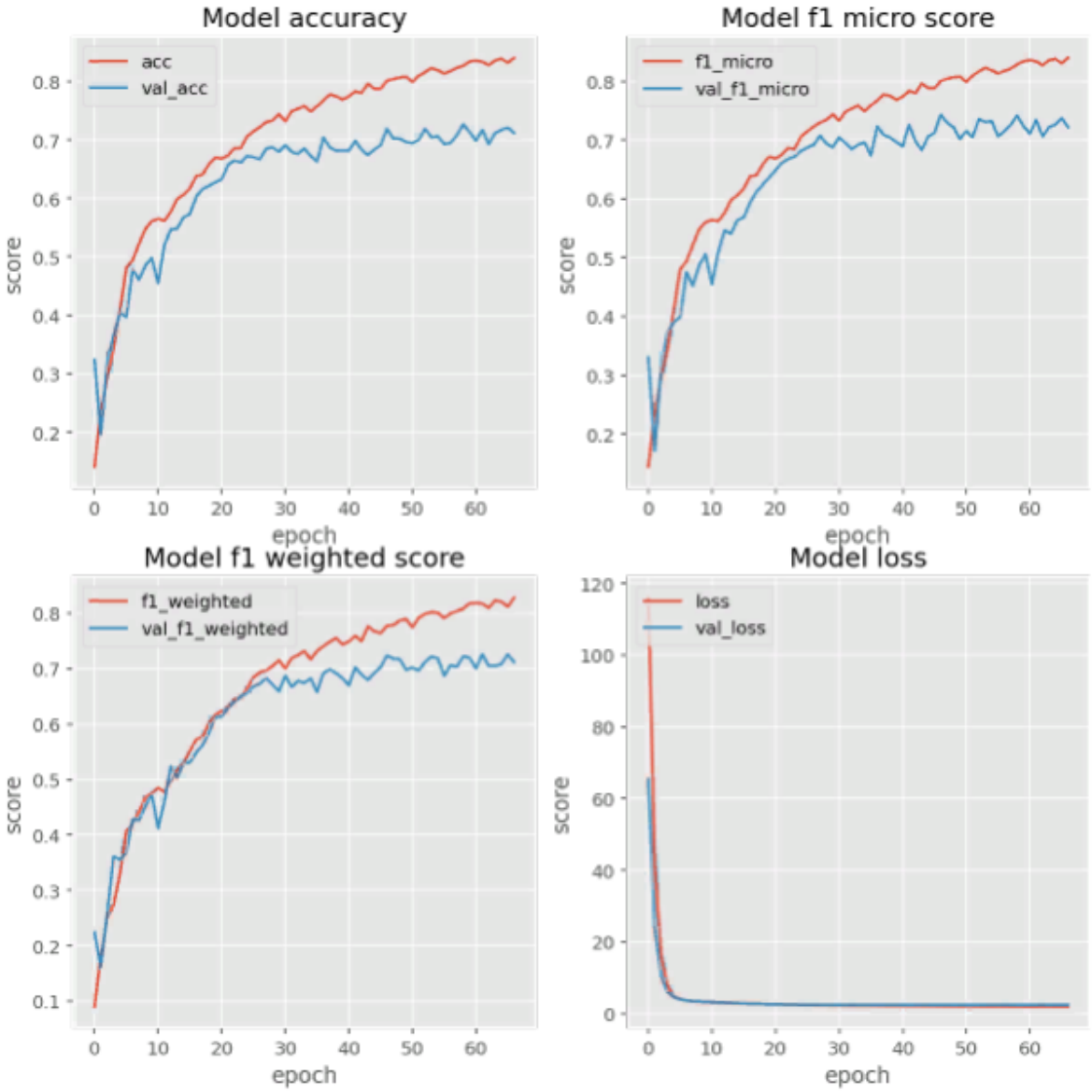

Figure 3: Оценка модели на эпохах обучения

Оценим модель с лучшей валидационной f1 micro

```
model.load_weights(get_drive_path('saved models\\saved-model-conv.hdf5'))

other_features_test = extract_features(test_data, encoder)
pred = np.round(model.predict([X_test, other_features_test]))

print('f1 micro', f1_score(y_test, pred, average='micro'))
print('f1 weighted', f1_score(y_test, pred, average='weighted'))
```

Получаем f1 micro 0.7419, f1 weighted 0.6859