Второй отборочный этап

Описание задачи

Люди научили машины читать, но способны ли те понять прочитанное? Вам предстоит доказать, что машины способны не только прочесть текст и понять, что там происходит, но и правильно ответить на вопросы, которые требуют построения причинно-следственных связей.

Ниже вы найдете набор данных, состоящий из текстов и вопросов к ним с разными вариантами ответа. Ваша задача: написать алгоритм, который определит, какие из ответов верные.

Основные принципы:

- Ответ содержится в нескольких предложениях, а не в одном!
- Ответ не четко (дословно) прописан в тексте. Полное соответствие ответа в изначальном параграфе найти нельзя.
- Количество правильных ответов может быть любым.

Подобные системы нужны для автоматизации колл-центров и сервисов клиентской поддержки, в чат-ботах и поисковых машинах. Подробнее об AGI-based вопросно-ответных системах, их использовании и методах построения смотрите в запись вебинара от разработчиков задачи (https://www.youtube.com/watch?v=guPAeY10CEc).

Также вам доступен вебинар с разбором базового решения (https://youtu.be/Jbj8R5sZzBk) и дополнительный вебинар (https://youtu.be/xDUGBm0zbf4), где рассказывается о новых способах решения задачи бинарной классификации в контексте NLP и разобрается, как работают трансформеры, формирующие из слов вектор, и как все-таки сформировать подтверждение в нужном формате для загрузки на платформу.

Данные

Для решения задачи вам предоставлен датасет, в котором около 6 000 вопросов для более чем 800 текстов из 5 разных областей:

- Тексты начальной школы
- Новости
- Художественные тексты
- Сказки
- Краткое содержание сериалов

В архиве вы найдете 3 файла:

- train и val — для обучения модели и настройки параметров;
- test — проверочный файл, в котором нужно сделать предсказание и загрузить на платформу для проверки.

Данные для решения задачи: https://russiansuperglue.com/tasks/download/

MuSeRC.

Формат решений

Задача представляет собой бинарную классификацию (True/False).

В систему для проверки необходимо предоставить jsonl-файл с метками ответов: если ответ верен — 1, если нет — 0. Качество решения определяется, как среднее значение точности и полноты ответов — F1 average.

Также вам доступно базовое решение от разработчиков задачи (https://github.com/AI-Front/NTI).

Пример выходных данных:


```
{
    "id": 397,
    "text": "(1) Мужская сборная команда Норвегии по биатлону в рамках этапа Кубка
    ↪  мира в немецком Оберхофе выиграла эстафетную гонку. (2) Вторыми стали
    ↪  французы, а бронзу получила немецкая команда. (3) Российские биатлонисты не
    ↪  смогли побороться даже за четвертое место, отстав от норвежцев более чем на
    ↪  две минуты. (4) Это худший результат сборной России в текущем сезоне. (5)
    ↪  Четвертыми в Оберхофе стали австрийцы. (6) В составе сборной Норвегии на
    ↪  четвертый этап вышел легендарный Уле-Эйнар Бьорндален. (7) Впрочем, Норвегия с
    ↪  самого начала гонки была в числе лидеров, успешно проведя все четыре этапа.
    ↪  (8) За сборную России в Оберхофе выступали Иван Черезов, Антон Шипулин,
    ↪  Евгений Устюгов и Максим Чудов. (9) Гонка не задалась уже с самого начала:
    ↪  если на стрельбе из положения лежа Черезов был точен, то из положения стоя он
    ↪  допустил несколько промахов, в результате чего ему пришлось бежать один
    ↪  дополнительный круг. (10) После этого отставание российской команды от
    ↪  соперников только увеличивалось. (11) Напомним, что днем ранее российские
    ↪  биатлонистки выиграли свою эстафету. (12) В составе сборной России выступали
    ↪  Анна Богалий-Титовец, Анна Булыгина, Ольга Медведцева и Светлана Слепцова.
    ↪  (13) Они опередили своих основных соперниц - немок - всего на 0,3 секунды.",
    "questions": [
        {
            "question": "На сколько секунд женская команда опередила своих соперниц?",
            "answers": [
                {
                    "text": "Всего на 0,3 секунды.",
                    "label": 1
                },
                {
                    "text": "На 0,3 секунды.",
                    "label": 1
                },
                {
                    "text": "На секунду.",
                    "label": 0
                },
                {
                    "text": "На 0.5 секунд.",
                    "label": 0
                }
            ],
            "idx": 0
        }]
```

}

Решение

Импортируем нужные библиотеки

```
import json
import os, re
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%pylab inline
from sklearn.metrics import *
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer

from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier, ExtraTreeClassifier
from sklearn.linear_model import SGDClassifier,LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

```
Populating the interactive namespace from numpy and matplotlib
```

Подготовим данные

Здесь у вас есть большой простор для инженерных решений:

- есть тройки — текст, вопрос, ответ
- есть пары — вопрос и правильный ответ, вопрос и неправильный ответ
- есть пары — вопрос и ответ,
- и т. д.

В бейзлайне мы пойдем по самому простому пути — склеим вместе текст, вопрос и ответ — это будет один элемент классификации, к которому приписан лейбл (класс — 0 или 1).

```
def get_X_y(data_json_file):
    X, y = [], []
    with open(data_json_file, 'r') as json_file:
        json_list = list(json_file)
        #print(json_list[0])
        for json_str in json_list:
            item = json.loads(json_str)
            text = item['passage']['text']
            #print(item['passage'].keys())
            questions = item['passage']['questions']
            for q in questions:
                query = q['question']
                ans = q['answers']
                for a in ans:
                    X.append(text+' Query: '+query+' Answer: '+a['text'])
                    y.append(a['label'])
    return X, y
```

```
X_train, y_train = get_X_y('train.jsonl')
X_test, y_test = get_X_y('val.jsonl')
```

Посмотрим на количество примеров и состав классов:

```
len(X_train)
```

```
11950
```

```
len(X_test)
```

```
2235
```

```
from collections import Counter
print(Counter(y_train))
print(Counter(y_test))
```

```
Counter({0: 6568, 1: 5382})
Counter({0: 1242, 1: 993})
```

Запуск baseline

Перебор моделей и параметров — классический подход.

Попробуем модели разного типа, выберем лучший классификатор из простых, а потом будем перебирать его настройки.

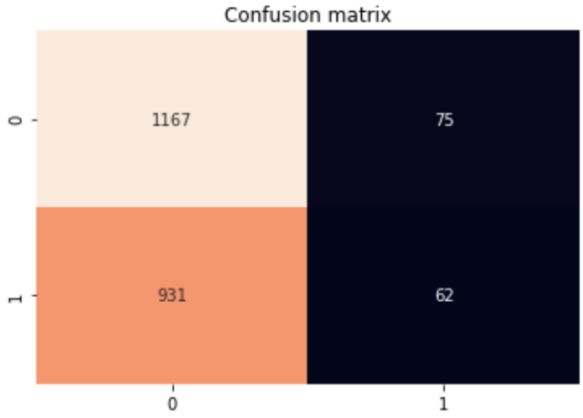
```
rs = 42

clf = LogisticRegression(random_state=rs)
clf2 = RandomForestClassifier(random_state=rs, n_jobs =-1)
clf3 = SGDClassifier()
clf4 = SVC(random_state =rs)
clf5 = DecisionTreeClassifier(random_state=rs)
"""clf6 =  SVC(class_weight="balanced", random_state =rs)
clf7 = DecisionTreeClassifier()
clf8 = ExtraTreeClassifier()
clf9 = LinearRegression()
clf10 = LogisticRegressionCV()
clf11 = GradientBoostingClassifier(random_state =rs)"""
clflist = [clf, clf2, clf3, clf4, clf5]
```

Запустим и оценим решение

```
for classif in clflist:
    clf = Pipeline([
    ('vect', CountVectorizer(ngram_range=(1,3), analyzer='word', max_features=10000)),
    ('tfidf', TfidfTransformer(sublinear_tf=True)),
    #('reducer', TruncatedSVD(n_components=Val3)),
    ('clf', classif),
    ])
    clf.fit(X_train, y_train)
    predictions = clf.predict(X_test)
    print(classif)
```

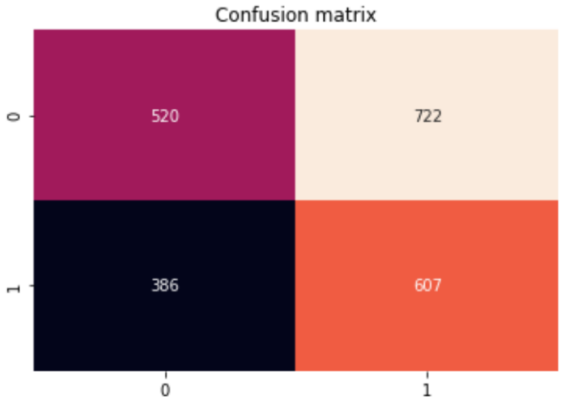
```
    print("Precision: {0:6.2f}".format(precision_score(y_test, predictions,
    ↪  average='macro')))
    print("Recall: {0:6.2f}".format(recall_score(y_test, predictions,
    ↪  average='macro')))
    print("F1-measure: {0:6.2f}".format(f1_score(y_test, predictions,
    ↪  average='macro')))
    print("Accuracy: {0:6.2f}".format(accuracy_score(y_test, predictions)))
    print(classification_report(y_test, predictions))
    labels = clf.classes_
    sns.heatmap(data=confusion_matrix(y_test, predictions), annot=True, fmt="d",
    ↪  cbar=False, xticklabels=labels, yticklabels=labels)
    plt.title("Confusion matrix")
    plt.show()
```

```
  LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                     intercept_scaling=1, l1_ratio=None, max_iter=100,
                     multi_class='auto', n_jobs=None, penalty='l2',
                     random_state=42, solver='lbfgs', tol=0.0001, verbose=0,
                     warm_start=False)
Precision:   0.50
Recall:   0.50
F1-measure:   0.40
Accuracy:   0.55
              precision    recall  f1-score   support

           0       0.56      0.94      0.70      1242
           1       0.45      0.06      0.11       993

    accuracy                           0.55      2235
   macro avg       0.50      0.50      0.40      2235
weighted avg       0.51      0.55      0.44      2235
```

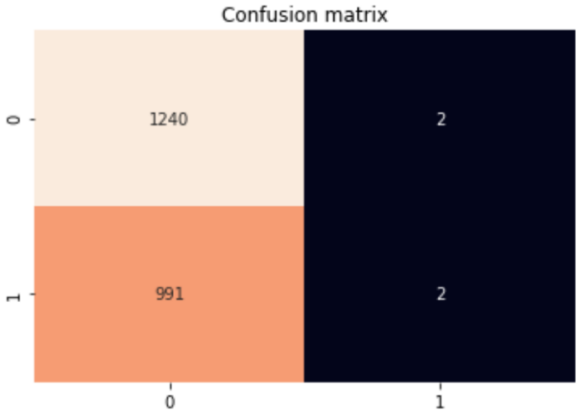



```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=-1, oob_score=False, random_state=42, verbose=0,
                       warm_start=False)
Precision:   0.52
Recall:   0.51
```

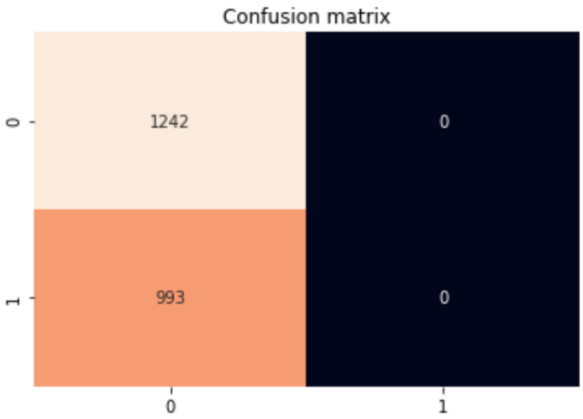
```
F1-measure:   0.50
Accuracy:   0.50
              precision    recall  f1-score   support

           0       0.57      0.42      0.48      1242
           1       0.46      0.61      0.52       993

    accuracy                           0.50      2235
   macro avg       0.52      0.51      0.50      2235
weighted avg       0.52      0.50      0.50      2235
```



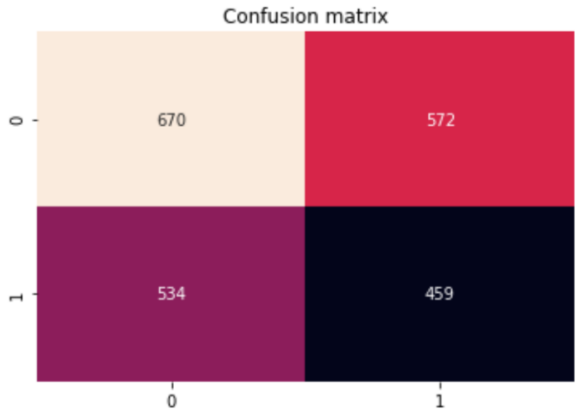

```
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
Precision:   0.53
Recall:   0.50
F1-measure:   0.36
Accuracy:   0.56
              precision    recall  f1-score   support

           0       0.56      1.00      0.71      1242
           1       0.50      0.00      0.00       993

    accuracy                           0.56      2235
   macro avg       0.53      0.50      0.36      2235
weighted avg       0.53      0.56      0.40      2235
```



```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=42, shrinking=True, tol=0.001,
    verbose=False)
Precision:   0.28
Recall:   0.50
F1-measure:   0.36
Accuracy:   0.56
              precision    recall  f1-score   support

           0       0.56      1.00      0.71      1242
           1       0.00      0.00      0.00       993

    accuracy                           0.56      2235
   macro avg       0.28      0.50      0.36      2235
weighted avg       0.31      0.56      0.40      2235
```




```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=42, splitter='best')
Precision:   0.50
Recall:   0.50
F1-measure:   0.50
Accuracy:   0.51
              precision    recall  f1-score   support

           0       0.56      0.54      0.55      1242
           1       0.45      0.46      0.45       993

    accuracy                           0.51      2235
   macro avg       0.50      0.50      0.50      2235
weighted avg       0.51      0.51      0.51      2235
```



Попробуем BERT

Перезапустите среду выполнения! И тут лучше переключиться на GPU!

Установите все библиотеки ниже, и затем заново загрузите данные, импортируйте sklearn — так вы избежите конфликта версий.

```
!pip install transformers
!pip install tensorboardx
!pip install simpletransformers


train_df = pd.DataFrame({
    'text': X_train,
    'label':y_train
})

print(train_df.head())

eval_df = pd.DataFrame({
    'text': X_test,
    'label': y_test
})
print(eval_df.head())
```

```
                                                text  label
0  (1) Но люди не могут существовать без природы,...      1
1  (1) Но люди не могут существовать без природы,...      0
2  (1) Но люди не могут существовать без природы,...      0
3  (1) Но люди не могут существовать без природы,...      1
4  (1) Но люди не могут существовать без природы,...      0
                                                text  label
0  (1) Самый первый «остров» Архипелага возник в ...      1
1  (1) Самый первый «остров» Архипелага возник в ...      0
2  (1) Самый первый «остров» Архипелага возник в ...      0
3  (1) Самый первый «остров» Архипелага возник в ...      0
4  (1) Самый первый «остров» Архипелага возник в ...      1
```

```
from simpletransformers.classification import ClassificationModel

model = ClassificationModel('bert', 'bert-base-multilingual-uncased',use_cuda=False)
```

```
# Train the model
model.train_model(train_df)

# Evaluate the model
result, model_outputs, wrong_predictions = model.eval_model(eval_df)


HBox(children=(HTML(value='Downloading'), FloatProgress(value=0.0, max=625.0),
↪  HTML(value='')))
HBox(children=(HTML(value='Downloading'), FloatProgress(value=0.0, max=672271273.0),
↪  HTML(value='')))

HBox(children=(HTML(value='Downloading'), FloatProgress(value=0.0, max=871891.0),
↪  HTML(value='')))

HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=11950.0), HTML(value='')))
HBox(children=(HTML(value='Epoch'), FloatProgress(value=0.0, max=1.0),
↪  HTML(value='')))
HBox(children=(HTML(value='Running Epoch 0 of 1'), FloatProgress(value=0.0,
↪  max=1494.0), HTML(value='')))


predictions, raw_outputs = model.predict(X_test)

print("Precision: {0:6.2f}".format(precision_score(y_test, predictions,
↪  average='macro')))
print("Recall: {0:6.2f}".format(recall_score(y_test, predictions, average='macro')))
print("F1-measure: {0:6.2f}".format(f1_score(y_test, predictions, average='macro')))
print("Accuracy: {0:6.2f}".format(accuracy_score(y_test, predictions)))
print(classification_report(y_test, predictions))
confusion_matrix(y_test, predictions)
```

Что еще можно улучшить?

Ограничение модели BERT — обработка только первых 512 токенов текста.

Нам это не очень подходит — если обрезать тексты по 512 слову, как это происходит в simple transformers, но мы можем обрезать и вопрос, и ответ! Тогда наш классификатор не сможет научиться различать неправильные ответы.

Обрежем тексты по началу, а не по концу:

```
def text_splitter(text, amount=500):
    tokens = text.split(' ')
    new_text = ' '.join(tokens[-amount:])
    return new_text

def get_X_y_for_bert(data_json_file):
    X, y = [], []
    with open(data_json_file, 'r') as json_file:
        json_list = list(json_file)
        #print(json_list[0])
        for json_str in json_list:
            item = json.loads(json_str)
            text = item['passage']['text']
            questions = item['passage']['questions']
            for q in questions:
                query = q['question']
                ans = q['answers']
                for a in ans:
```

```
                X.append(text_splitter(text+' Query: '+query+' Answer:
                ↪  '+a['text']))
                y.append(a['label'])
    return X, y
```

Еще идеи:

- больше эпох обучения;
- попробовать другие модели для классификации документов из simple transformers — только мультиязычные! См. документацию (https://simpletransformers.ai/docs/classification-models/).