Рубежный контроль №2

Тема: Методы обработки текстов

Группа: ИУ5-21М

Необходимо решить задачу классификации текстов на основе любого выбранного Вами датасета (кроме примера, который рассматривался в лекции. Классификация может быть бинарной или многоклассовой. Целевой признак из выбранного Вами датасета может иметь любой физический смысл, примером является задача анализа тональности текста.

- Необходимо сформировать два варианта векторизации признаков - на основе CountVectorizer и на основе TfidfVectorizer.
- В качестве классификаторов необходимо использовать два классификатора по варианту для Вашей группы (LogisticRegression, Multinomial Naive Bayes (MNB))

In [ ]:
```
# !pip install category-encoders
# !pip install imbalanced-learn==0.8.0
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Датасет

https://www.kaggle.com/crowdflower/twitter-user-gender-classification

In [ ]:
```
!unzip twitter_gender_classification.zip
```

```
Archive:  twitter_gender_classification.zip
  inflating: gender-classifier-DFE-791531.csv
```

In [ ]:
```
df = pd.read_csv('gender-classifier-DFE-791531.csv', encoding='latin1')
df
```

Out[ ]:

	_unit_id	_golden	_unit_state	_trusted_judgments	_last_judgment_at	gender	gender:conf
0	815719226	False	finalized	3	10/26/15 23:24	male	
1	815719227	False	finalized	3	10/26/15 23:30	male	
2	815719228	False	finalized	3	10/26/15 23:33	male	

	_unit_id	_golden	_unit_state	_trusted_judgments	_last_judgment_at	gender	gender:conf
3	815719229	False	finalized	3	10/26/15 23:10	male	
4	815719230	False	finalized	3	10/27/15 1:15	female	
...	...	...	...	...	...	...	
20045	815757572	True	golden	259	NaN	female	
20046	815757681	True	golden	248	NaN	male	
20047	815757830	True	golden	264	NaN	male	
20048	815757921	True	golden	250	NaN	female	
20049	815757985	True	golden	249	NaN	female	

20050 rows × 26 columns

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20050 entries, 0 to 20049
Data columns (total 26 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   _unit_id               20050 non-null  int64
 1   _golden                20050 non-null  bool
 2   _unit_state            20050 non-null  object
 3   _trusted_judgments     20050 non-null  int64
 4   _last_judgment_at      20000 non-null  object
 5   gender                 19953 non-null  object
 6   gender:confidence      20024 non-null  float64
 7   profile_yn             20050 non-null  object
 8   profile_yn:confidence  20050 non-null  float64
 9   created                20050 non-null  object
 10  description            16306 non-null  object
 11  fav_number             20050 non-null  int64
 12  gender_gold            50 non-null     object
 13  link_color             20050 non-null  object
 14  name                   20050 non-null  object
 15  profile_yn_gold        50 non-null     object
 16  profileimage           20050 non-null  object
 17  retweet_count          20050 non-null  int64
 18  sidebar_color          20050 non-null  object
 19  text                   20050 non-null  object
```

```
 20  tweet_coord            159 non-null    object
 21  tweet_count            20050 non-null  int64
 22  tweet_created          20050 non-null  object
 23  tweet_id               20050 non-null  float64
 24  tweet_location         12566 non-null  object
 25  user_timezone          12252 non-null  object
dtypes: bool(1), float64(3), int64(5), object(17)
memory usage: 3.8+ MB
```

Обработка пропусков

In [ ]:
```
print(df.gender.isna().sum()) # наш целевой признак, придется сбросить строки с nan

df.dropna(subset=['gender'], inplace=True)
```

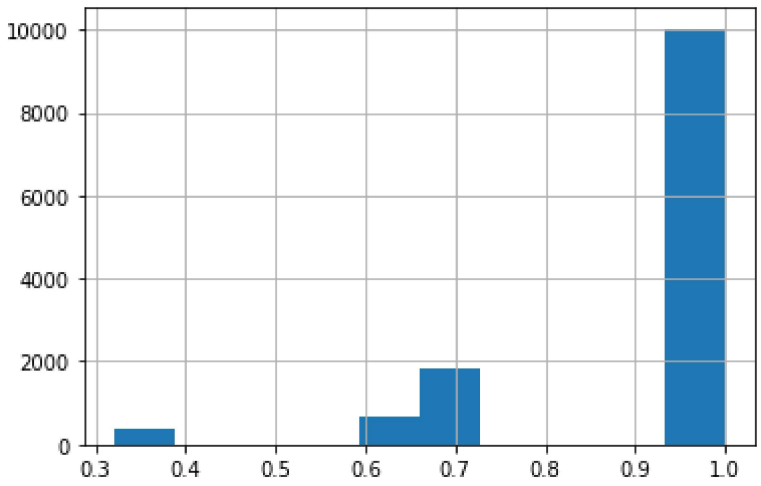
```
97
```

In [ ]:
```
df.gender.unique()
```

Out[ ]:
```
array(['male', 'female', 'brand', 'unknown'], dtype=object)
```

In [ ]:
```
df = df[(df.gender != 'unknown') & (df.gender != 'brand')]
```

In [ ]:
```
df['gender:confidence'].hist()
df = df[df['gender:confidence'] > 0.65] # уверенность в гендере попробуем сначала сд
# для обучения, то будем сортировать как уверенность > 0.65

# затем сбрасываем эту колонку
# df.drop(columns=['gender:confidence'])
```



In [ ]:
```
df.dropna(subset=['gender:confidence'], inplace=True)
```

In [ ]:
```
df.description.notna().sum() # количество записей с описанием профиля после фильтрац
```

Out[ ]:
```
10675
```

In [ ]:
```
df.dropna(subset=['description'], inplace=True)
```

In [ ]:
```
df.gender_gold.unique()
```

Out[ ]:
```
array([nan, 'male', 'female', 'female\nunknown'], dtype=object)
```

In [ ]:
```
# сбрасываем столбцы со слишком большим количеством пропусков
df.drop(columns=['gender_gold', '_golden', 'profile_yn_gold', 'tweet_coord'], inplac
```

In [ ]:
```
print(df.tweet_location.isna().sum()) # попробуем заполнить пропуски категориальным
df.tweet_location.fillna('not known', inplace=True)
```

```
2829
```

In [ ]:
```
df.user_timezone.isna().sum()
df.user_timezone.fillna('not known', inplace=True)
```

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10675 entries, 0 to 20049
Data columns (total 22 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   _unit_id               10675 non-null  int64
 1   _unit_state            10675 non-null  object
 2   _trusted_judgments     10675 non-null  int64
 3   _last_judgment_at      10643 non-null  object
 4   gender                 10675 non-null  object
 5   gender:confidence      10675 non-null  float64
 6   profile_yn             10675 non-null  object
 7   profile_yn:confidence  10675 non-null  float64
 8   created                10675 non-null  object
 9   description            10675 non-null  object
 10  fav_number             10675 non-null  int64
 11  link_color             10675 non-null  object
 12  name                   10675 non-null  object
 13  profileimage           10675 non-null  object
 14  retweet_count          10675 non-null  int64
 15  sidebar_color          10675 non-null  object
 16  text                   10675 non-null  object
 17  tweet_count            10675 non-null  int64
 18  tweet_created          10675 non-null  object
 19  tweet_id               10675 non-null  float64
 20  tweet_location         10675 non-null  object
 21  user_timezone          10675 non-null  object
dtypes: float64(3), int64(5), object(14)
memory usage: 1.9+ MB
```

Обработка категориальных признаков

In [ ]:
```
df.dtypes
```

Out[ ]:
```
_unit_id                  int64
_unit_state              object
_trusted_judgments        int64
_last_judgment_at        object
gender                   object
gender:confidence       float64
profile_yn               object
profile_yn:confidence   float64
created                  object
description              object
```

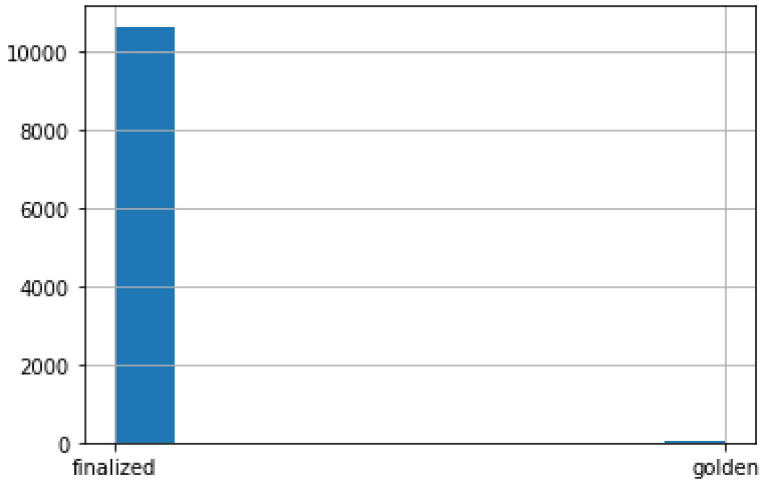
```
fav_number                int64
link_color               object
name                     object
profileimage             object
retweet_count             int64
sidebar_color            object
text                     object
tweet_count               int64
tweet_created            object
tweet_id                float64
tweet_location           object
user_timezone            object
dtype: object
```

кодируем таргет

In [ ]:
```
from sklearn.preprocessing import LabelEncoder

target_le = LabelEncoder()

df['gender_encoded'] = target_le.fit_transform(df.gender)


df.drop(columns=['gender'], inplace=True)
```

In [ ]:
```
df._unit_state.unique()
df._unit_state.hist()
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f935d5d8c50>
```



In [ ]:
```
from category_encoders import TargetEncoder

ustate_te = TargetEncoder()

df['_unit_state'] = ustate_te.fit_transform(df._unit_state, df.gender_encoded)
```

```
/usr/local/lib/python3.7/dist-packages/category_encoders/utils.py:21: FutureWarning:
is_categorical is deprecated and will be removed in a future version.  Use is_catego
rical_dtype instead
  elif pd.api.types.is_categorical(cols):
```

In [ ]:
```
df.drop(columns=['_last_judgment_at', 'created', 'tweet_created'], inplace=True) # n
```

In [ ]:
```
pyn_te = TargetEncoder()
```

```
df['profile_yn'] = pyn_te.fit_transform(df.profile_yn, df.gender_encoded)
```

```
/usr/local/lib/python3.7/dist-packages/category_encoders/utils.py:21: FutureWarning:
is_categorical is deprecated and will be removed in a future version.  Use is_catego
rical_dtype instead
  elif pd.api.types.is_categorical(cols):
```

In [ ]:
```
df.tweet_location.unique().__len__() # придется все таки сбросить tweet_location сли
# а кодировать текстом возможно мало смысла
df.drop(columns=['tweet_location'], inplace=True)
```

In [ ]:
```
df.user_timezone.unique().__len__()

ustz_te = TargetEncoder()

df['user_timezone'] = ustz_te.fit_transform(df.user_timezone, df.gender_encoded)
```

```
/usr/local/lib/python3.7/dist-packages/category_encoders/utils.py:21: FutureWarning:
is_categorical is deprecated and will be removed in a future version.  Use is_catego
rical_dtype instead
  elif pd.api.types.is_categorical(cols):
```

Обработка "цветовых фич"

In [ ]:
```
df.link_color


def get_color_vals_rgb(color_text):
  if color_text != '0' and len(color_text) == 6:
    return tuple(int(color_text[i:i+2], 16) for i in (0, 2, 4))
  else:
    return (0, 0, 0)

r = []
g = []
b = []
for col in list(df.link_color):
  tmp_col = get_color_vals_rgb(col)
  r.append(tmp_col[0])
  g.append(tmp_col[1])
  b.append(tmp_col[2])

r,g,b

df['r'] = r
df['g'] = g
df['b'] = b

df.drop(columns=['link_color'], inplace=True)
```

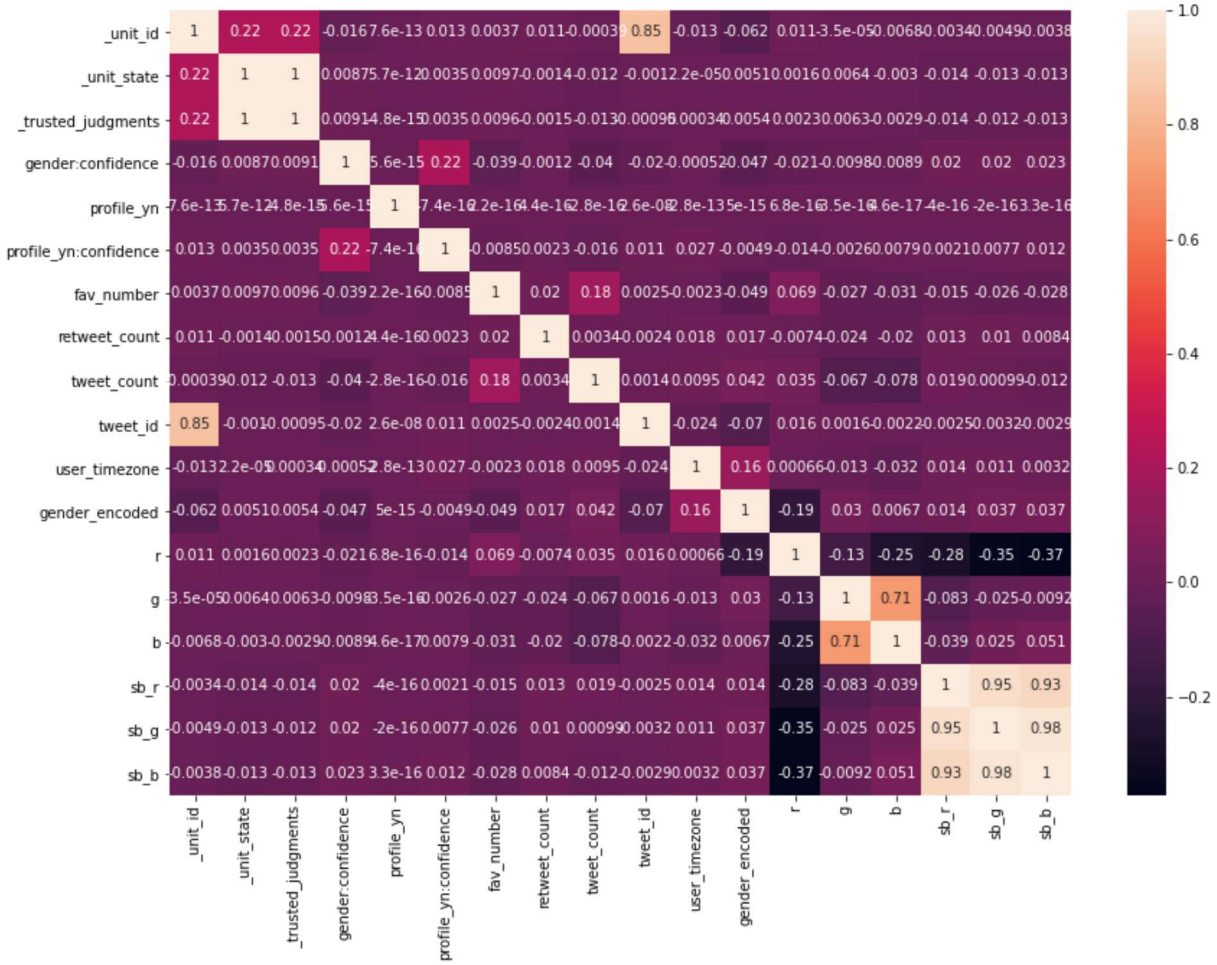
In [ ]:
```
df.drop(columns=['profileimage'], inplace=True)
```

In [ ]:
```
df.sidebar_color.unique()

sb_r = []
sb_g = []
sb_b = []
```

```
for col in list(df.sidebar_color):
  tmp_col = get_color_vals_rgb(col)
  sb_r.append(tmp_col[0])
  sb_g.append(tmp_col[1])
  sb_b.append(tmp_col[2])

df['sb_r'] = sb_r
df['sb_g'] = sb_g
df['sb_b'] = sb_b

df.drop(columns=['sidebar_color'], inplace=True)
```

Проведем корреляционный анализ по признакам не связанным с текстами

In [ ]:
```
plt.figure(figsize=(14, 10))
ax = sns.heatmap(df.corr(), annot=True)
plt.show()
```



Разделим выборки на тестовую и тренировочную

In [ ]:
```
from sklearn.model_selection import train_test_split

df.drop(columns=['_unit_id', 'tweet_id',
                 '_trusted_judgments', 'sb_r', 'sb_g', 'g'], inplace=True)

X = df[[i for i in df.columns if i !='gender_encoded']]
y = df.gender_encoded
```

```
x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=123,
                                                    stratify=y, test_size=0.2)
```

Кодирование текстовых фич

In [ ]:
```
import re
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()

def prepare_text(text):

  # TODO: дописать лемматизацию или стемминг если качество не устроит
  text = re.sub(r'[^a-zA-Z0-9 \n]', '', text).lower()
  return ' '.join([stemmer.stem(w) for w in text.split(' ')])

train_text_preprocessed = x_train.text.apply(prepare_text)
test_text_preprocessed = x_test.text.apply(prepare_text)


tfidf_text_vect = TfidfVectorizer()
count_text_vect = CountVectorizer()

train_text_tfidf = tfidf_text_vect.fit_transform(train_text_preprocessed)
test_text_tfidf = tfidf_text_vect.transform(test_text_preprocessed)

train_text_count = count_text_vect.fit_transform(train_text_preprocessed)
test_text_count = count_text_vect.transform(test_text_preprocessed)

train_text_count.shape, test_text_count.shape
```

Out[ ]:
```
((8540, 18943), (2135, 18943))
```

In [ ]:
```
train_description_preprocessed = x_train.description.apply(prepare_text)
test_description_preprocessed = x_test.description.apply(prepare_text)

tfidf_descr_vect = TfidfVectorizer()
count_descr_vect = CountVectorizer()

train_description_tfidf = tfidf_descr_vect.fit_transform(train_description_preproces
test_description_tfidf = tfidf_descr_vect.transform(test_description_preprocessed)

train_description_count = count_descr_vect.fit_transform(train_description_preproces
test_description_count = count_descr_vect.transform(test_description_preprocessed)

train_text_count.shape, test_text_count.shape
```

Out[ ]:
```
((8540, 18943), (2135, 18943))
```

In [ ]:
```
# удаляем  поле name т.к. слишком уникальные слова нет смысла использовать кодирован
# остальные удаляем тк есть уже закодированные
x_train.drop(columns=['name','description', 'text'], inplace=True)
x_test.drop(columns=['name','description', 'text'], inplace=True)
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py:4174: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  errors=errors,
```

Создание выборок с учетом матриц по текстам

In [ ]:
```
from scipy import sparse

train_features_sparse = sparse.coo_matrix(x_train)
test_features_sparse = sparse.coo_matrix(x_test)

train_count = sparse.hstack((train_text_count, train_description_count, train_featur
train_tfidf = sparse.hstack((train_text_tfidf, train_description_tfidf, train_featur

test_count = sparse.hstack((test_text_count, test_description_count, test_features_s
test_tfidf = sparse.hstack((test_text_tfidf, test_description_tfidf, test_features_s
```

In [ ]:
```
train_count.shape, test_count.shape
```

Out[ ]:
```
((8540, 38233), (2135, 38233))
```

Logistic Regression CountVectorizer

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

logreg_clf = LogisticRegression(random_state=123)

logreg_clf.fit(train_count, y_train)

pred_logreg_count = logreg_clf.predict(test_count)

print(classification_report(y_test, pred_logreg_count))
```

```
              precision    recall  f1-score   support

           0       0.60      0.50      0.55      1101
           1       0.55      0.64      0.59      1034

    accuracy                           0.57      2135
   macro avg       0.57      0.57      0.57      2135
weighted avg       0.57      0.57      0.57      2135
```

LogisticRegression TfidfVectorizer

In [ ]:
```
logreg_clf_tfidf = LogisticRegression(random_state=123)

logreg_clf_tfidf.fit(train_tfidf, y_train)

pred_logreg_tfidf = logreg_clf_tfidf.predict(test_tfidf)

print(classification_report(y_test, pred_logreg_tfidf))
```

```
              precision    recall  f1-score   support

           0       0.60      0.50      0.55      1101
           1       0.55      0.64      0.59      1034
```

```
    accuracy                           0.57      2135
   macro avg       0.57      0.57      0.57      2135
weighted avg       0.57      0.57      0.57      2135
```

Multinomial Naive Bayes CountVectorizer

In [ ]:
```
from sklearn.naive_bayes import MultinomialNB

mnb_clf_count = MultinomialNB()

mnb_clf_count.fit(train_count, y_train)

pred_mnb_count = mnb_clf_count.predict(test_count)

print(classification_report(y_test, pred_mnb_count))
```

```
              precision    recall  f1-score   support

           0       0.58      0.63      0.60      1101
           1       0.56      0.50      0.53      1034

    accuracy                           0.57      2135
   macro avg       0.57      0.57      0.57      2135
weighted avg       0.57      0.57      0.57      2135
```

Multinomial Naive Bayes TfidfVectorizer

In [ ]:
```
mnb_clf_tfidf = MultinomialNB()

mnb_clf_tfidf.fit(train_tfidf, y_train)

pred_mnb_tfidf = mnb_clf_tfidf.predict(test_tfidf)

print(classification_report(y_test, pred_mnb_tfidf))
```

```
              precision    recall  f1-score   support

           0       0.58      0.63      0.60      1101
           1       0.56      0.50      0.53      1034

    accuracy                           0.57      2135
   macro avg       0.57      0.57      0.57      2135
weighted avg       0.57      0.57      0.57      2135
```