# Домашнее задание по курсу «Методы машинного обучения»

Кузьмин Роман, ИУ5-25М

В качестве домашнего задания выполнена лабораторная работа по теме "Классификация текста"

## Импорт библиотек

In [1]:
```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorize
r
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import pandas as pd
import time
```

Для произвольного набора данных, предназначенного для классификации текстов, решите задачу классификации текста двумя способами:

Способ 1. На основе CountVectorizer или TfidfVectorizer.

Способ 2. На основе моделей word2vec или Glove или fastText.

Сравните качество полученных моделей.

Датасет: Spam Emails (https://www.kaggle.com/datasets/abdallahwagih/spam-emails). Содержит спам и не-спам емейлы

In [2]:
```
# Загрузка данных
df = pd.read_csv("spam.csv")

# Заменяем целевую переменную класса на числовое значение
df.Category = df.Category.apply(lambda x: 1 if x == 'spam' else 0)

df.head()
```

Out[2]:

| | Category | Message |
|---|---|---|
| **0** | 0 | Go until jurong point, crazy.. Available only ... |
| **1** | 0 | Ok lar... Joking wif u oni... |
| **2** | 1 | Free entry in 2 a wkly comp to win FA Cup fina... |
| **3** | 0 | U dun say so early hor... U c already then say... |
| **4** | 0 | Nah I don't think he goes to usf, he lives aro... |

In [3]:
```
df.shape
```

Out[3]:
```
(5572, 2)
```

In [4]:
```
X_train, X_test, y_train, y_test = train_test_split(df['Message'], df['Category'], test_size=0.2, random_state=42)
```

# Классификация SVC + TFIDFVectorizer

In [5]:
```
tfidf_vectorizer = TfidfVectorizer()
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
```

In [6]:
```
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

In [7]:
```
svc_classifier_tfidf = SVC()
```

In [8]:
```
svc_classifier_tfidf.fit(X_train_tfidf, y_train)
```

Out[8]:
```
SVC()
```

In [9]:
```
svc_accuracy = svc_classifier_tfidf.score(X_test_tfidf, y_test)
print('Точность SVC + TFIDF:', svc_accuracy)
```

```
Точность SVC + TFIDF: 0.989237668161435
```

# Классификация word2vec

In [35]:
```
from nltk import WordPunctTokenizer
from nltk.corpus import stopwords
import nltk
import re
from gensim.models import word2vec
import numpy as np
from sklearn.pipeline import Pipeline

nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[35]:
```
True
```

In [17]:
```
corpus = []
stop_words = stopwords.words('english')
tok = WordPunctTokenizer()
for line in df['Message'].values:
    line1 = line.strip().lower()
    line1 = re.sub("[^a-zA-Z]"," ", line1)
    text_tok = tok.tokenize(line1)
    text_tok1 = [w for w in text_tok if not w in stop_words]
    corpus.append(text_tok1)
```

In [40]:
```
# количество текстов в корпусе не изменилось и соответствует целевому призн
аку
assert df.shape[0]==len(corpus)
```

In [41]:
```
%%time
model = word2vec.Word2Vec(corpus, workers=4, min_count=10, window=10, sampl
e=1e-3)
```

```
CPU times: user 550 ms, sys: 4.39 ms, total: 554 ms
Wall time: 534 ms
```

In [42]:
```
print(model.wv.most_similar(positive=['early'], topn=5))
```

```
[('late', 0.9993149638175964), ('well', 0.9993125200271606), ('buy', 0.9993
104338645935), ('think', 0.9992956519126892), ('work', 0.9992709755897522)]
```

In [43]:
```
def accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray):
    """
    Вычисление метрики accuracy для каждого класса
    y_true - истинные значения классов
    y_pred - предсказанные значения классов
    Возвращает словарь: ключ - метка класса,
    значение - Accuracy для данного класса
    """
    # Для удобства фильтрации сформируем Pandas DataFrame
    d = {'t': y_true, 'p': y_pred}
    df = pd.DataFrame(data=d)
    # Метки классов
    classes = np.unique(y_true)
    # Результирующий словарь
    res = dict()
    # Перебор меток классов
    for c in classes:
        # отфильтруем данные, которые соответствуют
        # текущей метке класса в истинных значениях
        temp_data_flt = df[df['t']==c]
        # расчет accuracy для заданной метки класса
        temp_acc = accuracy_score(
            temp_data_flt['t'].values,
            temp_data_flt['p'].values)
        # сохранение результата в словарь
        res[c] = temp_acc
    return res

def print_accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray):
    """
    Вывод метрики accuracy для каждого класса
    """
    accs = accuracy_score_for_classes(y_true, y_pred)
    if len(accs)>0:
        print('Метка \t Accuracy')
    for i in accs:
        print('{} \t {}'.format(i, accs[i]))
```

In [44]:
```
class EmbeddingVectorizer(object):
    '''
    Для текста усредним вектора входящих в него слов
    '''
    def __init__(self, model):
        self.model = model
        self.size = model.vector_size

    def fit(self, X, y):
        return self

    def transform(self, X):
        return np.array([np.mean(
            [self.model[w] for w in words if w in self.model]
            or [np.zeros(self.size)], axis=0)
            for words in X])
```

In [36]:
```
def spam(v, c):
    model = Pipeline(
        [("vectorizer", v),
         ("classifier", c)])
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print_accuracy_score_for_classes(y_test, y_pred)
```

In [45]:
```
spam(EmbeddingVectorizer(model.wv), LogisticRegression(C=5.0))
```

```
Метка    Accuracy
0        0.9979296066252588
1        0.0
```

Word2vec показывает лучшее качество по выделению класса не spam. Но в этом случае скорее всего происходит переобучение и модель максимимзирует accuracy, хотя надо выбирать другую метрику при дисбалансе классов. Так что если работать именно с accuracy, лучше выбирать модель классификации из TFIDF Vectorizer + SVC Classifier.