Анализ текстов

Лекция

Классификация текстов. Анализ тональности.

Мурат Апишев (great-mel@yandex.ru)

3 октября, 2018

Содержание занятия

- Задача классификации текстов
- Метрики качества
- Отбор моделей
- Блендинг и стекинг
- Библиотека Vowpal Wabbit, hashing trick
- Библиотека FastText
- Задача анализа тональности
- Свёрточные сети для классификации текстов

Постановка задачи классификации

Данные:

- $d \in D$ – множество документов (объектов)
- $c \in C$ – множество меток классов

Типы задач:

- Бинарная классификация: $|C| = 2, \forall d \in D \leftrightarrow c \in C$
- Многоклассовая классификация [multiclass]:
  $|C| = K, K > 2, \forall d \in D \leftrightarrow c \in C$
- Многотемная классификация [multi-label]:
  $|C| = K, K > 2, \forall d \in D \leftrightarrow \tilde{C} \subseteq C$

Примеры задач

1. Фильтрация спама: $C = \{spam, good\}$
2. Анализ тональности (простой): $C = \{pos, neg, neutral\}$
3. Рубрикация: $C = \{sport, hobby, \dots\}$ – multiclass [+ multi-label]
4. Определение авторства:
   - Этим ли автором написан текст?
   - Каким(-и) из авторов написан текст?
   - Какова возрастная группа автора? Пол автора?

Метрики качества бинарной классификации

Простейшая метрика – *аккуратность* (accuracy):

$$\text{acc} = \frac{\#(correct)}{\#(total)}$$

В жизни почти не используется. **Почему?**

Чаще всего используются *точность* (precision) и *полнота* (recall):

- $precision = Pr = \frac{tp}{tp + fp}$
- $recall = R = \frac{tp}{tp + fn}$
- $F_1 = \frac{2}{\frac{1}{Pr} + \frac{1}{R}} = \frac{2 \cdot Pr \cdot R}{Pr + R}$

		gold standart	
		positive	negative
classification output	positive	tp	fp
	negative	fn	tn

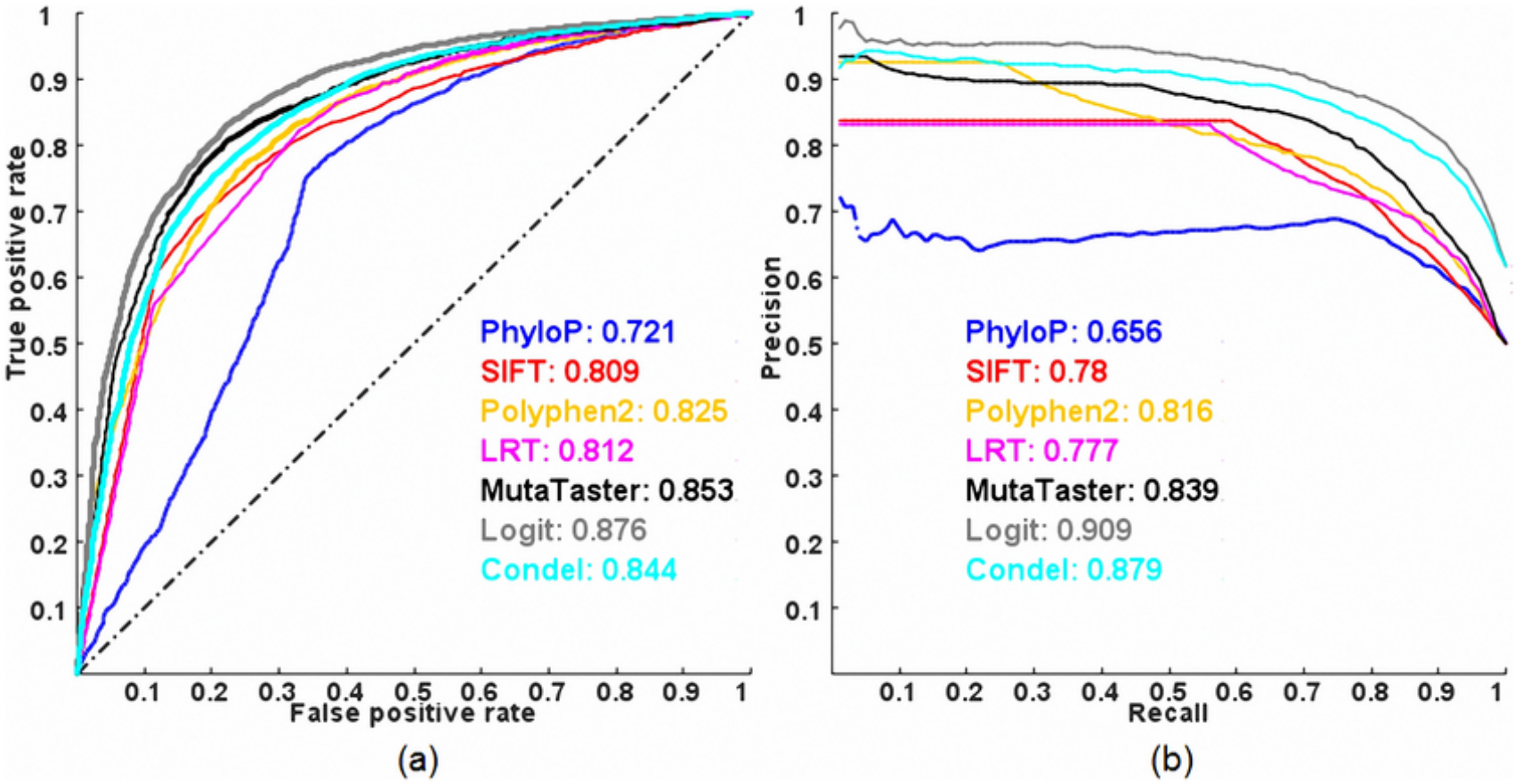
Метрики качества многоклассовой классификации

- Микро-усреднение:
  - $Pr_{micro} = \frac{\sum tp_i}{\sum tp_i + \sum fp_i}$
  - $R_{micro} = \frac{\sum tp_i}{\sum tp_i + \sum fn_i}$
- Макро-усреднение:
  - $Pr_{macro} = \frac{\sum Pr_i}{|C|}$
  - $R_{macro} = \frac{\sum R_i}{|C|}$

		gold standart		
		$class_1$	$class_2$	$class_3$
classification output	$class_1$	tp_1	fp_{12}	fp_{13}
	$class_2$	fn_{21}	tp_2	fp_{23}
	$class_3$	fn_{31}	fn_{32}	tp_3

- Микро-усреднение нивелирует влияние маленьких классов
- Макро-усреднение учитывает все классы в равной степени

Метрики качества: AUC-ROC, AUC-PR



AUC-PR лучше подходит для несбалансированных классов!

Источник картинки

Признаки для текстов

Классификация по правилам:

- если в предложении встречается личное местоимение первого лица и глагол с окончанием женского рода, то пол автора женский
- если доля положительно окрашенных прилагательных в отзыве больше доли отрицательно окрашенных прилагательных, то отзыв относится к классу positive

Признаки для применения ML:

- счётчики слов
- TF-IDF слов
- N-граммы (в т.ч. и символьные, + коллокации, термины, именованные сущности и т.п.)

Признаки для текстов

- Текстовые признаки, как правило, очень разреженные
  - в Python существует много типов разреженных матриц с разными свойствами (можно посмотреть здесь)
  - Почти все классификаторы `sklearn` работают с разреженными данными, исключение `GradientBoostingClassifier`
- Обычно используется «Bag-of-words» (но не всегда!)
  - `sklearn.feature_extraction.text.CountVectorizer`
  - `sklearn.feature_extraction.text.TfIdfVectorizer`
- Из разреженных признаков можно получить плотные путём сжатия признакового пространства (SVD, NNMF, PLSA)

Описание данных

- 10 тысяч вопросов со StackOverflow
- Каждый вопрос имеет либо тег **windows**, либо тег **linux**

```
1 import pandas as pd
2 texts = pd.read_csv('windows_vs_linux.10k.tsv',
3                      header=None, sep='\t')
4 texts.columns = ['text', 'is_windows']
5 texts.head(4)
```

	text	is_windows
0	so i find myself porting a game that was orig...	0
1	i ve been using tortoisesvn in a windows envi...	1
2	we are using wmv videos on an internal site a...	1
3	on one linux server running apache and php 5 ...	0

Примеры кода

Собрать «мешок слов» со счётчиками:

```
vectorizer = CountVectorizer(binary=True)
bow = vectorizer.fit_transform(texts.text)
print(type(bow))
# <class 'scipy.sparse.csr.csr_matrix'>
```

Собрать TF-IDF:

```
vectorizer = TfidfVectorizer()
tf_idf = vectorizer.fit_transform(texts.text)
```

Сжать пространство TF-IDF признаков:

```
svd = TruncatedSVD(n_components=200, n_iter=5)
tf_idf_svd = svd.fit_transform(tf_idf)
```

Выбор модели

Кросс-валидация:

- различными стратегиями делим обучающую выборку на N частей
- обучаем модель на $N - 1$ частях, на оставшейся тестируем перебираем все варианты
- параметры выбираем усреднением

```
1 params = {'C': np.logspace(-5, 5, 11)}
2
3 clf = LogisticRegression()
4 cv = GridSearchCV(clf, params, n_jobs=-1,
5                   scoring='roc_auc', cv=5)
6 cv.fit(bow, texts.is_windows);
```

Выбор модели

```
pd.DataFrame(cv.cv_results_)[['mean_test_score',
                              'params']]
  .sort_values('mean_test_score', ascending=False)
```

	mean_test_score	params
4	0.965813	{u'C': 0.1}
5	0.962415	{u'C': 1.0}
3	0.961759	{u'C': 0.01}
6	0.955353	{u'C': 10.0}
7	0.948635	{u'C': 100.0}
2	0.945693	{u'C': 0.001}
8	0.945429	{u'C': 1000.0}
9	0.943599	{u'C': 10000.0}
10	0.942903	{u'C': 100000.0}
1	0.790566	{u'C': 0.0001}
0	0.632507	{u'C': 1e-05}

Наиболее значимые слова

```
top = pd.DataFrame(
  [get_top_windows(cv.best_estimator_, 6),
   get_top_linux(cv.best_estimator_, 6)]
   ).T
top.columns = ['Windows', 'Linux']
top
```

	Windows	**Linux**
0	windows	ubuntu
1	win32	root
2	vista	mono
3	exe	linux
4	dll	kernel
5	batch	bash

Отбор признаков

```
vect = CountVectorizer(binary=True, ngram_range=(1, 4))
bow = vect.fit_transform(texts.text)
print(bow.shape)  # (10000, 2117115)
```

```
k_best = SelectKBest(k=50000)
bow_k_best = k_best.fit_transform(bow, texts.is_windows)
```

```
clf = LogisticRegression()
np.mean(cross_val_score(clf, bow_k_best,
                        texts.is_windows,
                        scoring='roc_auc', cv=5))
```

Получили более высокое качество на кросс-валидации (0.97955)

Есть ли подвох?

Код вычисления метрик (sklearn.metrics)

Аккуратность (доля верных ответов):

```
accuracy_score(predicted, target)
```

Точность, полнота, F_1-score:

```
precision_score(target, prediction)
recall_score(target, prediction)
f1_score(target, prediction)
```

F_1-score для многоклассового случая:

```
f1_score(target, prediction, average = 'micro')
f1_score(target, prediction, average = 'macro')
```

AUC-ROC, AUC-PR:

```
roc_auc_score(target, prediction)
average_precision_score(target, prediction)
```

Multiclass и multilabel классификация

```
1 texts = pd.read_csv('multi_tag.10k.tsv', header=None,
      sep='\t')
2 texts.columns = ['text', 'tags']
3 print(texts.shape)
4 texts.head(4)
```

	text	tags
0	i want to use a track bar to change a form s ...	c# winforms type-conversion decimal opacity
1	i have an absolutely positioned div containin...	html css css3 internet-explorer-7
2	given a datetime representing a person s birt...	c# .net datetime
3	given a specific datetime value how do i disp...	c# datetime datediff relative-time-span

Многоклассовая классификация

По сути – обобщение бинарной классификации.

Часть классификаторов `sklearn` умеет работать с многоклассовым случаем:

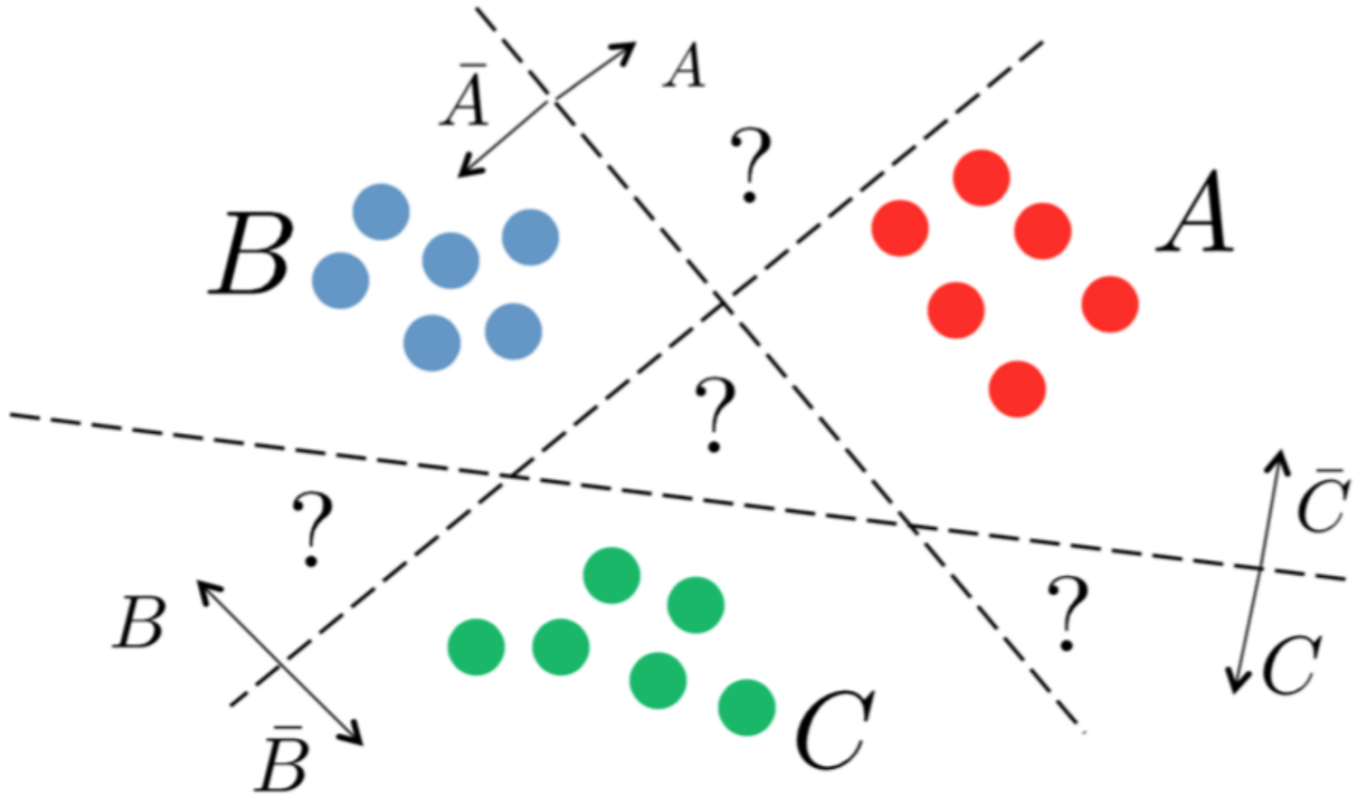
- `KNeighborsClassifier`
- `RandomForestClassifier`
- `SVC`

Ддя остальных есть адапторы:

- `OneVsRestClassifier`
- `OneVsOneClassifier`

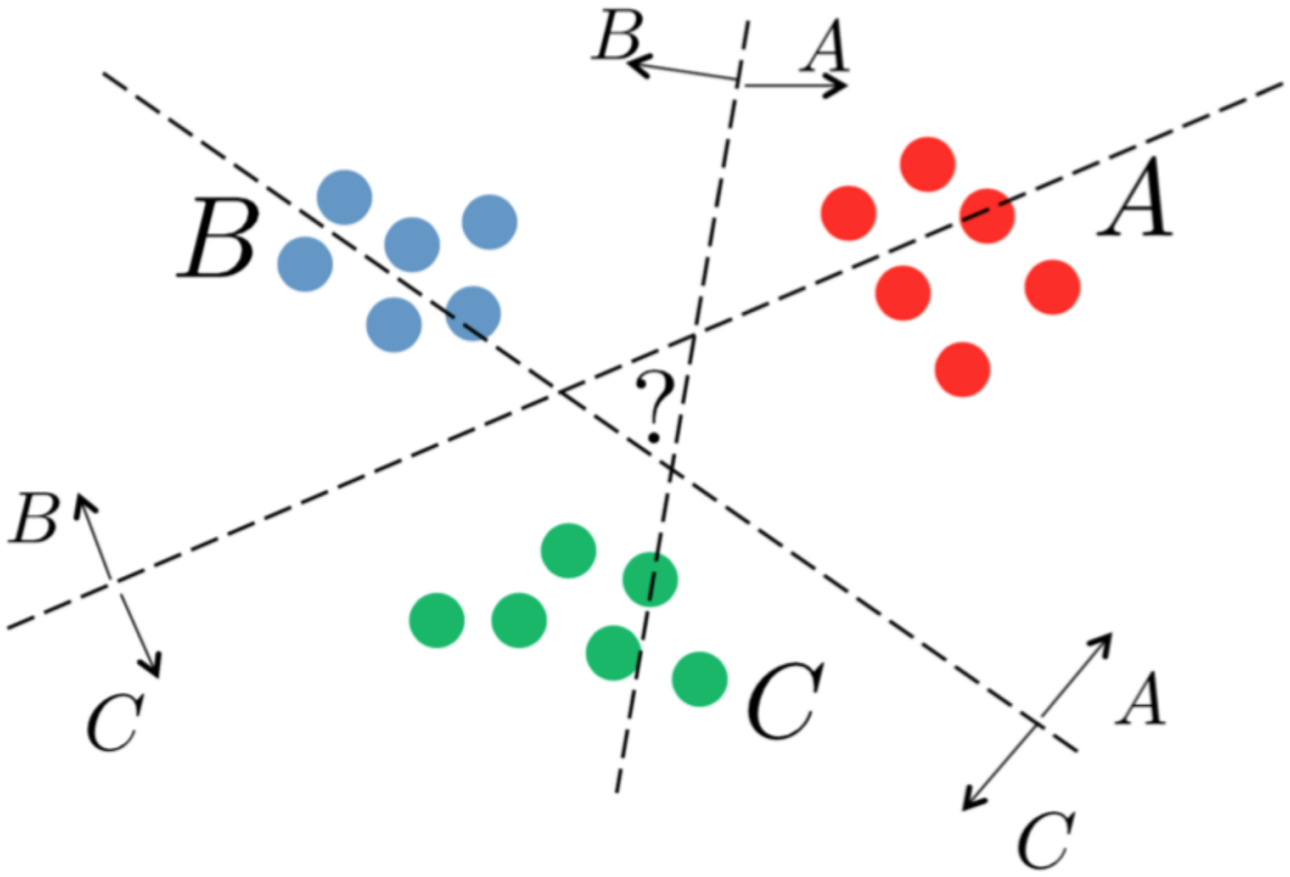
Многоклассовая классификация

One-vs-Rest: обучает $|C|$ моделей, для каждой метки отдельно



Многоклассовая классификация

One-vs-One: обучает $\frac{|C|\cdot(|C|-1)}{2}$ моделей, для каждой метки отдельно



Обработка данных

- Собираем «мешок слов» с помощью `CountVectorizer`
- Отбираем 20 самых популярных тегов
- Тем, кто после фильтрации остался без тегов, ставим новый тег **other**

```
1 tags = texts.tags.apply(lambda x: x.split())
2 all_tags = reduce(lambda s, x: s + x, tags, [])
3 values, count = np.unique(all_tags, return_counts=True)
4
5 top_tags = sorted(zip(count, values), reverse=True)[:20]
```

Выбор модели

Преобразуем списки тегов в матрицу, которая будет содержать индикаторы наличия тега у вопроса:

```
binarizer = MultiLabelBinarizer()
y = binarizer.fit_transform(
  texts.tags.apply(lambda x: filter_tags(x.split())))
```

Также будет использовать LogisticRegression, но уже вместе с OneVsRestClassifier:

```
params = {'estimator__C': np.logspace(-5, 5, 11)}
clf = OneVsRestClassifier(LogisticRegression())

cv = GridSearchCV(clf, params, n_jobs=-1, cv=5,
                  scoring=make_scorer(f1_score,
                                      average='samples'))

cv.fit(bow, y);
```

Выбор модели

```
pd.DataFrame(cv.cv_results_)[['mean_test_score',
                              'params']]
            .sort_values('mean_test_score',
                         ascending=False)
```

- F_1-score = 0.40473
- Задача стала существенно сложнее
- Попробуем улучшить качество

	mean_test_score	params
10	0.404732	{u'estimator__C': 100000.0}
9	0.404212	{u'estimator__C': 10000.0}
8	0.404140	{u'estimator__C': 1000.0}
7	0.403066	{u'estimator__C': 100.0}
6	0.402630	{u'estimator__C': 10.0}
5	0.390346	{u'estimator__C': 1.0}
4	0.319707	{u'estimator__C': 0.1}
3	0.092920	{u'estimator__C': 0.01}
2	0.000300	{u'estimator__C': 0.001}
0	0.000000	{u'estimator__C': 1e-05}
1	0.000000	{u'estimator__C': 0.0001}

Изменим признаки

Заменим счётчики на TF-IDF:

```
vectorizer = TfidfVectorizer()
tf_idf = vectorizer.fit_transform(texts.text)
```

Попробуем выбрать модель:

```
params = {'estimator__C': np.logspace(-5, 5, 11)}

clf = OneVsRestClassifier(LogisticRegression())
cv = GridSearchCV(clf, params, n_jobs=-1, cv=5,
                  scoring=make_scorer(f1_score,
                                      average='samples'))

cv.fit(tf_idf, y);
```

По логике должно получиться лучше

Выбор модели

```
pd.DataFrame(cv.cv_results_)[['mean_test_score', 'params']]
            .sort_values('mean_test_score', ascending=False)
```

- F_1-score = 0.38217
- Получилось ощутимо хуже, чем с «мешком слов»
- **Почему?**

	mean_test_score	params
10	0.382173	{u'estimator__C': 100000.0}
9	0.380033	{u'estimator__C': 10000.0}
8	0.376427	{u'estimator__C': 1000.0}
7	0.364680	{u'estimator__C': 100.0}
6	0.334247	{u'estimator__C': 10.0}
5	0.182323	{u'estimator__C': 1.0}
4	0.000700	{u'estimator__C': 0.1}
0	0.000000	{u'estimator__C': 1e-05}
1	0.000000	{u'estimator__C': 0.0001}
2	0.000000	{u'estimator__C': 0.001}
3	0.000000	{u'estimator__C': 0.01}

Выбираем порог

- При вызове predict возвращается 1, если вероятность принадлежности к классу больше 0.5
- Можно выбирать порог самому через кросс-валидацию

```
clf = OneVsRestClassifier(LogisticRegression(C=100000))
y_hat_bow = cross_val_predict(clf, bow, y,
                              method='predict_proba')
y_hat_tf_idf = cross_val_predict(clf, tf_idf, y,
                                 method='predict_proba')
```

Определим функцию, выставляющую тег в зависимости от порога:

```
def get_score(alpha, y, y_hat):
    return f1_score(y, (y_hat > alpha).astype('int'),
                    average='samples')
```

Подбор порога

```
alphas = np.linspace(0.0, 0.01, 100)
scores = [get_score(a, y, y_hat_bow) for a in alphas]

print(np.max(scores))
print(alphas[np.argmax(scores)])
```

BOW:

- Качество с порогом по умолчанию: 0.40473
- Качество с подобранным порогом: 0.45435

TF-IDF:

- Качество с порогом по умолчанию: 0.38217
- Качество с подобранным порогом: **0.49397**

Hashing Trick

- Модели лучше строить с N-граммами
- Но использование N-грамм (даже с грамотным отбором) существенно увеличивает объём словаря
- Выход – вместо самих N-грамм хранить заданное количество хэшей от них:

```
1 vectorizer = HashingVectorizer(binary=True,
2                                ngram_range=(1, 3),
3                                n_features=50000)
4 bow = vectorizer.fit_transform(texts.text)
```

Блендинг

- При наличии несколько моделей можем получать смешенное предсказание
- Если модели не сильно скоррелированы, это может улучшить качество результирующей модели

```
vectorizer = HashingVectorizer(binary=True,
                               ngram_range=(1, 3),
                               n_features=50000)
bow = vectorizer.fit_transform(texts.text)
```

```
vectorizer = TfidfVectorizer()
tf_idf = vectorizer.fit_transform(texts.text)
```

Блендинг

С помощью кросс-валидации предскажем обучающую выборку для каждой модели:

```
clf_lr = OneVsRestClassifier(LogisticRegression(C=10000))

y_hat_lr = cross_val_predict(clf_lr, bow,
                             y, cv=folds,
                             method='predict_proba')
```

```
clf_lr = OneVsRestClassifier(LogisticRegression(C=10000))

y_hat_lr_tf_idf = cross_val_predict(clf_lr, tf_idf,
                                    y, cv=folds,
                                    method='
    predict_proba')
```

Блендинг

Получим качество на каждой модели в отдельности и на их смеси (веса возьмём равными):

```
alphas = np.linspace(0.0, 0.02, 100)

[get_score(a, y, y_hat_lr) for a in alphas]
[get_score(a, y, y_hat_lr_tf_idf) for a in alphas]

[get_score(a, y, 0.5 * y_hat_lr_tf_idf + 0.5 * y_hat_lr)
           for a in alphas]
```

- Качество первой модели: 0.50923
- Качество второй модели: 0.49355
- Качество смеси: **0.52709**

Стекинг

Вместо ручного смешивания результатов можно подавать их на вход другому алгоритму (*мета-алгоритму*)

Подготовим переменную `stacked`, которая будет содержать предсказания предыдущих алгоритмов

```
1 stacked = np.hstack([y_hat_lr, y_hat_lr_tf_idf])
2
3 clf_stacked = OneVsRestClassifier(
4   RandomForestClassifier(n_estimators=100))
5
6 y_hat_stacked = cross_val_predict(clf_stacked, stacked,
7                                    y, cv=folds,
8                                    method='predict_proba')
```

Стекинг

```
alpha = np.linspace(0, 1, 100)
scores = [get_score(a, y, y_hat_stacked) for a in alpha]

plot(alpha, scores);
scatter(alpha[np.argmax(scores)], np.max(scores));

print(np.max(scores))
print(alpha[np.argmax(scores)])
```

```
0.547874126984
0.232323232323
```

После подбора порога получили $F_1 = \mathbf{0.54787}$, что больше всех предыдущих результатов.

Стекинг

Стекинг – важная и часто полезная на практике техника, но в её использовании есть подводные камни, связанные с переобучением, поэтому «стекать» надо грамотно.

Полезные ресурсы:

1. Kaggle Ensembling Guide
2. Стекинг и блендинг (Дьяконов)

Библиотека Vowpal Wabbit

- Разработка Yahoo и потом Microsoft
- Библиотека и CLI программа, позволяющая строить линейные модели
- Способна обрабатывать миллиарды объектов с сотнями тысяч признаков

Установка:

- Ubuntu – `apt-get instal vowpal-wabbit`
- Mac OS – `port install vowpal_wabbit`
- Windows – скачать установочник тут
- Варианты установки из официальной вики

Формат ввода

- Использует необычный формат входных данных
- `Label [weight] |Namespace Feature ...|Namespace ...`
  - `Label` – метка класса для задачи классификации или действительное число для задачи регрессии
  - `weight` – вес объекта, по умолчанию у всех одинаковый
  - `Namespace` – все признаки разбиты на области видимости, может использоваться для раздельного использования или создания квадратичных признаков между областями
  - `Feature` – `string[:value]` или `int[:value]` строки будут хешированы, числа будут использоваться как индекс в векторе признаков. `value` по умолчанию равно 1

Параметры

Hashing Trick: Вводится функция h, с помощью которой получается индекс для записи значения в вектор признаков объекта:

$$h : F \to \{0, \ldots, 2^b - 1\}$$

С помощью `--b` можно задавать размер области значений хеш-функции.

Оптимизация: может использоваться SGD или L-BFGS

- SGD по умолчанию, позволяет делать онлайн обучение. Почти всегда необходимо несколько проходов по данным
- L-BFGS включается с помощью `--bfgs`, работает только с данными небольшого размера
- Количество проходов для SGD задаётся с помощью параметра `--passes}`

Параметры оптимизации

Проходим по всем элементам обучающей выборки много раз, на каждом объекте делаем поправку весов:

$$w_{t+1} = w_t + \eta_t \bigtriangledown_w \ell(w_t, x_t)$$

$$\eta_t = \lambda d^k \left(\frac{t_0}{t_0 + w} \right)$$

Здесь t – порядковый номер объекта обучения, k — номер прохода по всей выборке

- λ: `-l` (learning rate)
- d: `--decay_learning_rate`
- t_0: `--initial_t`
- p: `--power_t`
- k_{max}: `--passes`

Оценка качества

average loss – loss by progressive validation

$$\text{Progressive error} = \frac{e_1 + e_2 + \cdots + e_s}{s}$$

e_i – loss на объекте x_i при обучении на объектах $\{x_1, \ldots, x_{i-1}\}$

Функция потерь задаётся параметром `--loss_function`

Vowpal Wabbit поддерживает несколько основных функций потерь для классификации и регрессии (squared, logistic, hinge и т.д.)

Возможность использовать **регуляризацию** с помощью флагов:

- `--l1`
- `--l2`

Данные для бинарной классификации

- Два класса с признаками A и B:

```
-1 | A:1 B:10
1 | A:-1 B:12
```

- Можно использовать текст без обработки (решим задачу windows vs. linux):

```
1 | i have a bat file shown below echo off for f d
1 | i need a way to determine whether the computer
-1 | my c application uses 3rd libraries which do
-1 | currently i m trying to install php 5 3 0 on
1 | i how to get the windowproc for a form in c fo
```

Обучение и оценивание

Запуск обучения:

```
!vw -d win_vs_lin.train.vw --loss_function logistic -P
    10000 -f model.vw --passes 100 -c
```

Применение:

```
!vw -i model.vw -t -p output.csv win_vs_lin.test.vw --
    loss_function logistic
```

Результат:

```
y_hat = pd.read_csv('output.csv', header=None)
roc_auc_score(
  test_texts[1].replace({'-1 ': 0, '1 ': 1}), y_hat[0])
```

0.96415

Многоклассовая классификация

Включается с помощью флага `--multilabel_oaa n`, где n – число классов

Данные будут выглядеть так:

```
3 | i want to use a track bar to change a form s
6 | i have an absolutely positioned div containing
0,3 | given a datetime representing a person s
3 | given a specific datetime value how do i
6,8 | is there any standard way for a web server
```

В остальном всё аналогично бинарному случаю.

Библиотека FastText

- Разработка Facebook
- Библиотека для получения векторных представлений слов и классификации текстов
- Архитектурно схожа с моделью skipgram (word2vec)
- Основана на символьных N-граммах, использует hashing trick
- Работает быстро и качественно
- Ссылки на статьи: 1, 2
- Ссылка на репозиторий
- Документация пакета для Python

Архитектура классификатора FastText

- Двухслойная нейронная сеть, вход – эмбеддинг документа
- На выходе стоит иерархический софтмакс с деревом Хаффмана
- Эмбеддинг документа получается путём усреднения эмбеддингов входящих в него слов (а эмбеддинги слов могут обучаться сами по себе, а могут быть усреднением после обучения N-символьных эмбеддингов)

Формат данных и обучение

Данные подаются в виде файла, каждый документ – одна строка вида

```
__label__food-safety __label__acidity Dangerous pathogens capable of growing in acidic environments
```

Обучение и сохранение модели:

```
1 classifier = fasttext.supervised('data.train.txt',
2                                  'model')
```

Применение обученной модели к тестовым данным:

```
1 result = classifier.test('test.txt')
```

Оценивание результатов

Оценивание качества на тесте:

```
print 'P@1:', result.precision
print 'R@1:', result.recall
print 'Number of examples:', result.nexamples
```

Наиболее вероятные классы (и их вероятности):

```
labels = classifier.predict(texts, k=3)
labels = classifier.predict_proba(texts, k=3)
```

Задачи сантимент-анализа

Три типа задач (по возрастанию сложности):

1. Полярная тональность (positive / negative / neutral)
2. Ранжированная тональность («звёздочки» от 1 до N)
3. Выявление источника / цели или более сложные типы

http://www.greenbookblog.org/2012/01/02/from-sentiment-analysis-to-enterprise-applications/

Ключевые моменты

- Токенизацию и лемматизацию нужно производить аккуратно, как и фильтрацию словаря
- С опечатками можно бороться с помощью буквенных N-грамм
- `не_` лучше приклеивать к впередистоящему слову, инвертируя его тональность
- Очень важно выделять смайлы и экспрессивную пунктуацию (регулярные выражения)
- Обучение и тестирование нужно производить на схожих данных
- Для учёта редких слов можно логарифмировать частоты

Сложности

Помимо чисто технических проблем, возникают также более сложные семантические:

- Отзывы могут иметь ясный смысл, но при этом не содержать позитивных или негативных слов:

  Это фильм заставляет прочувствовать всю гамму эмоций от «А» до «Я».

- Отзыв может содержать позитивные слова, но на самом деле выражает ожидание:

  Это фильм должен был быть супер крутым. Но не был.

Подходы к решению

- Правила (точно, трудозатратно)

  Я люблю кофе – если сказуемое в группе положительных глаголов и нет отрицаний то positive

- Словари (просто, зависит от предметной области)

слово i из текста	соответствующая тональность $a_i \in [1\text{-}9]$ из словаря
хороший	7.47
счастливый	8.21
...	...
скучный	2.95

$$a_{text} = \frac{\sum_{i=1}^{n} a_i}{n}$$

Подходы к решению

- ML: обучение с учителем (классификация) (точно при достаточной обучающей выборке, требуется данные для обучения)
  - Получаем размеченную коллекцию текстов
  - Выделяем признаки (специфичные для тональности)
  - Обучаем классификатор
- ML: обучение без учителя (просто, не требуются данные для обучения, нужен словарь, низкая точность)
  - Выделить ключевые слова (например, по TF-IDF)
  - Определить тональность ключевых слов
  - Сделать вывод о тональности предложения/текста

Сантимент-лексикон

- Существуют наборы слов с оценённой степенью позитивности/негативности, активности.пассивности и т.п.

  Примеры[2]: MPQA subjectivity cues lexicon, SentiWordNet

- Сантимент-лексикон можно размечать самостоятельно, это задача частичного обучения. Необходимо разметить часть примеров и описать правила пополнения.

  Пример: `bad and ugly; cruel but amazing`

  Если два слова связаны and, и тональность одного нам известна, то второе тоже будет иметь такую тональность.

  Если через but — то противоположную.

[2] Ссылка на ресурсы

О классификации

Naive Bayes:

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}}\, p(c) \prod_{w \in d} \hat{p}(w|c)$$

$$\hat{p}(w|c) = \frac{count(w, c) + 1}{count(c) + |W|}$$

- $count(w, c)$ — число раз, когда слово w встретилось в документе с классом c;
- $count(c)$ — общее число слов в документах с классом c.

Пример классификации

```
1  from nltk.classify.util import accuracy
2  from nltk.classify import NaiveBayesClassifier
3  from nltk.corpus import movie_reviews
4
5  def get_feats(tokens):
6      return dict([(token, True) for token in tokens])
7
8  def create_sample(tag, train_ratio):
9      ids = movie_reviews.fileids(tag)
10     feats = [(get_feats(
11                 movie_reviews.words(fileids=[f])),
12                 tag) for f in ids]
13
14     idx = int(len(feats) * train_ratio)
15     train, test = feats[: idx], feats[idx: ]
16     return train, test
```

Пример классификации

```
1  train_pos, test_pos = create_sample('pos', 0.75)
2  train_neg, test_neg = create_sample('neg', 0.75)
3  train = train_pos + train_neg
4  test = test_pos + test_neg
5
6  print('Train on {} docs, test on P{} docs'.format(
7            len(train), len(test)))
8
9  classifier = NaiveBayesClassifier.train(train)
10
11 print('Accuracy: {}'.format(accuracy(classifier, test)))
12
13 classifier.show_most_informative_features()
```

Результаты

```
Train on 1500 documents, test on 4000 documents
Accuracy: 0.728

Most Informative Features
     magnificent = True      pos : neg = 15.0 : 1.0
     outstanding = True      pos : neg = 13.6 : 1.0
     insulting = True        neg : pos = 13.0 : 1.0
     vulnerable = True       pos : neg = 12.3 : 1.0
     ludicrous = True        neg : pos = 11.8 : 1.0
     avoids = True           pos : neg = 11.7 : 1.0
     uninvolving = True      neg : pos = 11.7 : 1.0
     astounding = True       pos : neg = 10.3 : 1.0
     fascination = True      pos : neg = 10.3 : 1.0
     idiotic = True          neg : pos = 9.8 : 1.0
```

Модели и признаки для классификации

Используется всё, что для других задач классификации:

Модели:

- Лог-регрессия
- Рекурентные сети
- Свёрточные сети
- FastText

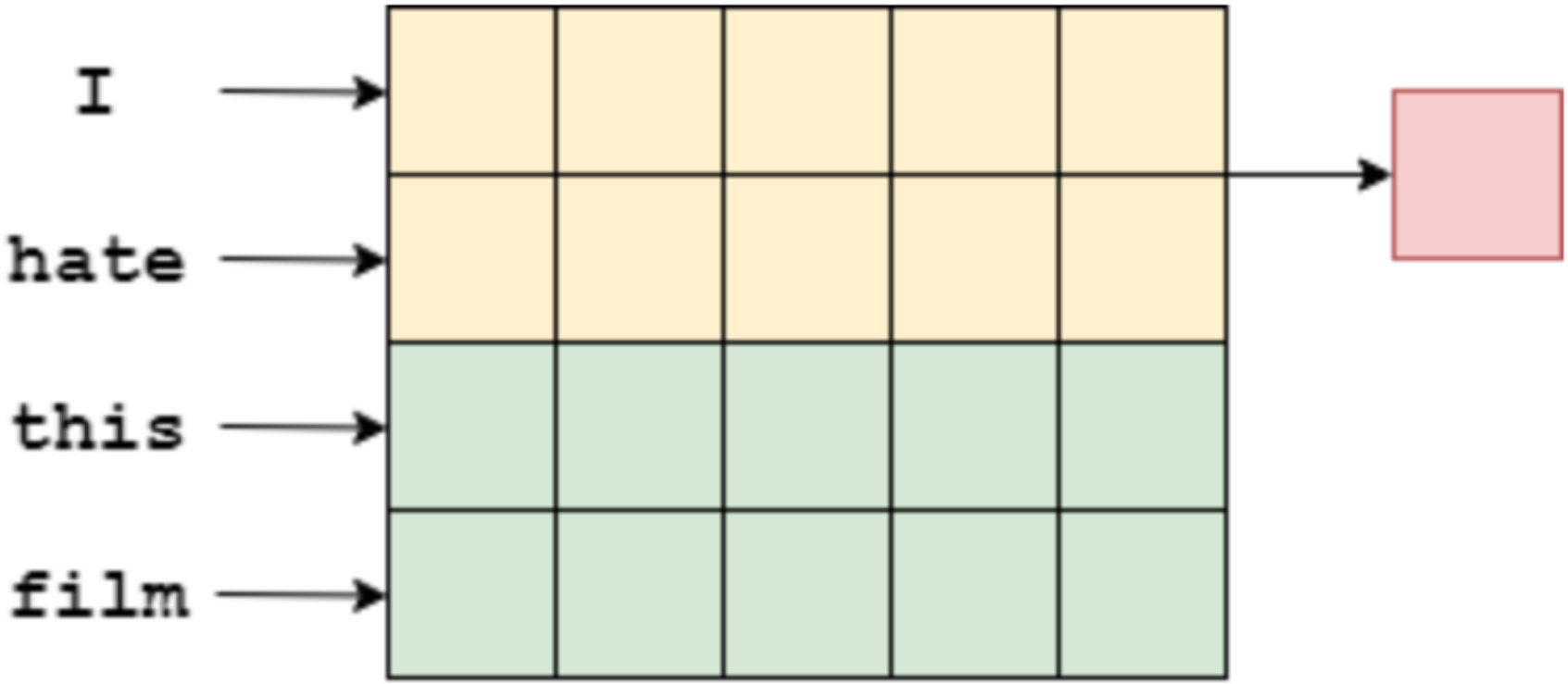
Признаки (в зависимости от выбранной модели):

- One-hot encoding
- TF-IDF
- Словарные эмбеддинги
- Всё это на N-граммах
- N-символьные эмбеддинги

Классификация свёрточными сетями

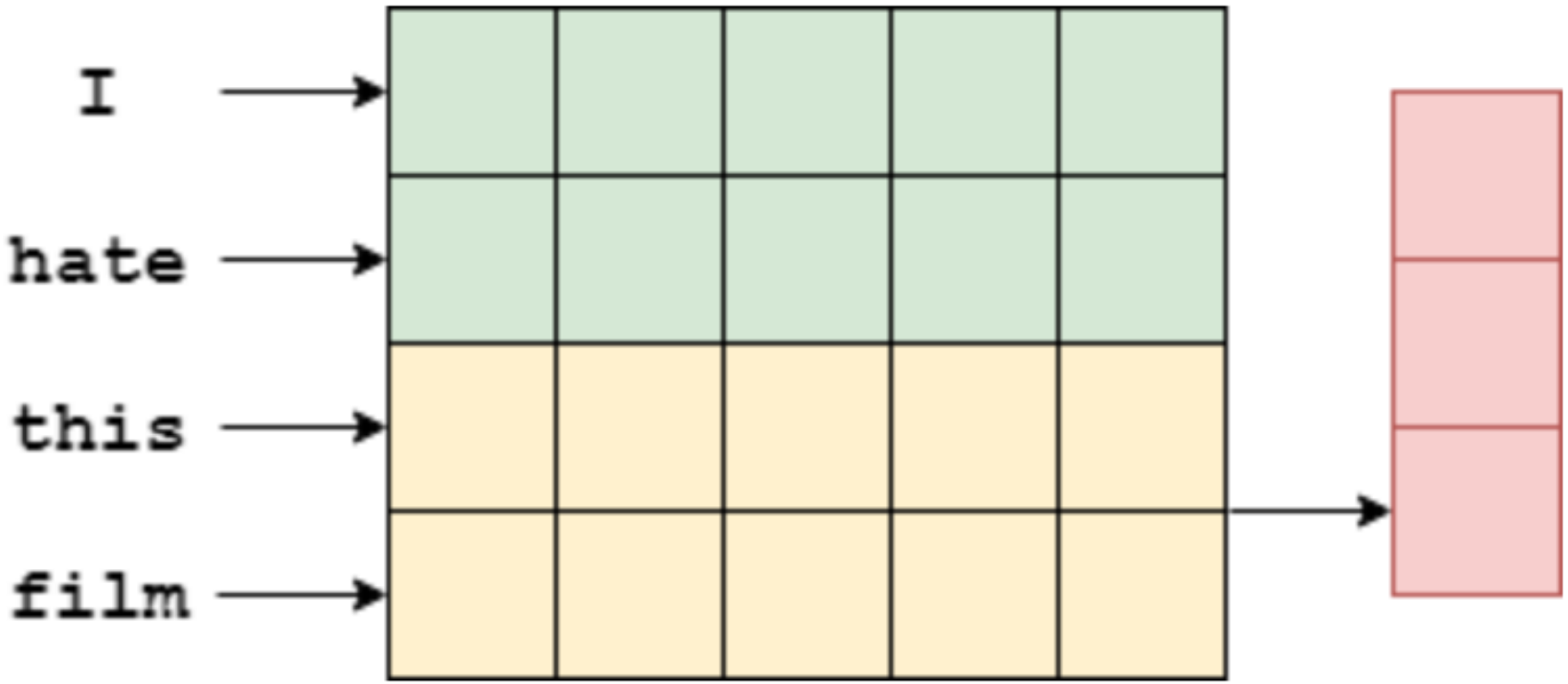
- Для применения свёрточных сетей к текстам необходимо привести текст к матричному виду.
- Для этого можно воспользоваться эмбеддингами
- Все объекты в одном батче должны быть одного размера, поэтому нужно либо нарезать тексты фрагментами фиксированной длины, длины использовать паддинг
- Для работы с текстами используются одномерные свёртки (вторая размерность всегда равна размерности эмбеддингов)

Классификация свёрточными сетями



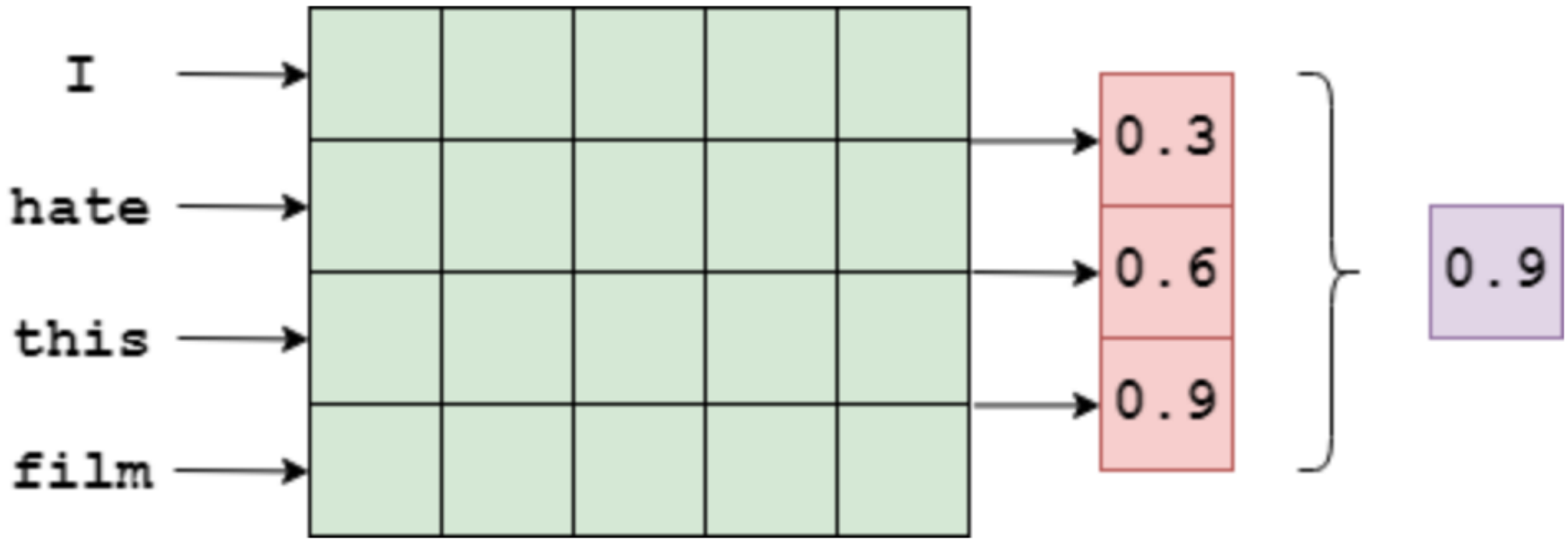
Ссылка на источник картинок

Классификация свёрточными сетями



Классификация свёрточными сетями

Следующий шаг, как и в свёрточных сетях для изображений – max-пуллинг:



Классификация свёрточными сетями

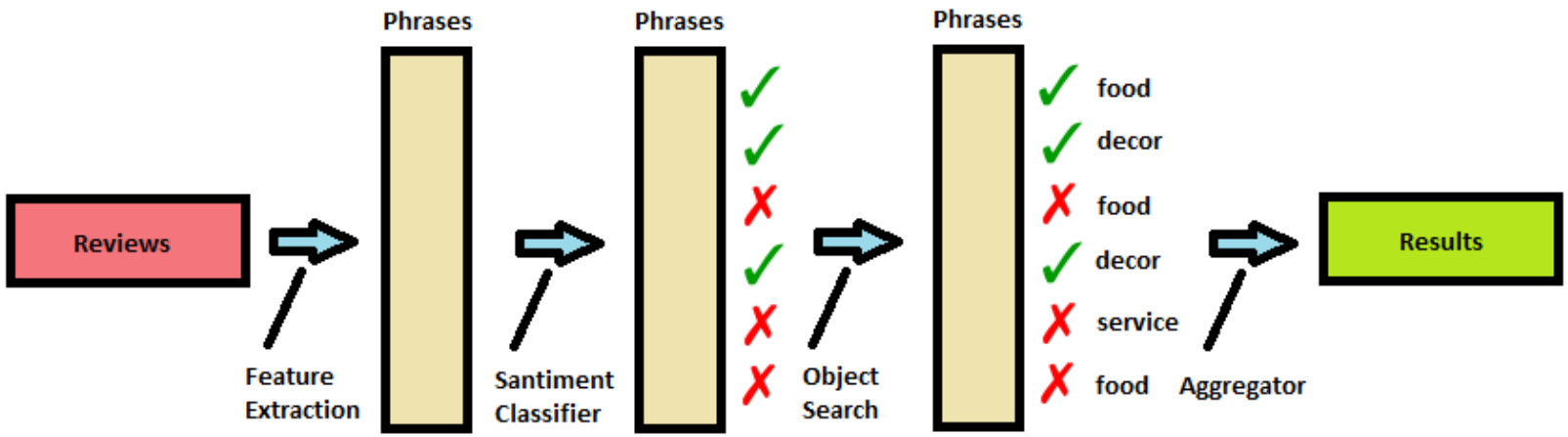
- Основная размерность свёрток может быть разной, что позволяет выделять признаки на N-граммах различной длины
- Идея в том, что наибольшее значение наиболее важного признака соответствует наиболее важной N-грамме текста.
- Получить его можно с помощью max-пуллинга, за счёт изменения backpropagation-ом весов свёрток для максимизации значения признака, наиболее влияющего на предсказание метки класса.
- Число фильтров будет определять размерность выходного эмбеддинга текста, который далее передаётся в линейный слой с софтмаксом на выходе для предсказания тональности.

Более сложные задачи

Поиск атрибутов и их объектов — в одном ревью могут по-разному оцениваться разные вещи. (Здесь могут помочь синтаксические зависимости!)

Для этого ищем частые фразы, которые нам интересны, и смотрим на то, сколько раз они встречаются сразу после сантиментных слов (пример: `great fish tacos`).

Такую работу несложно проделать для ресторанов, отелей и т.п.: food, decor, service и синонимы.



Online-анализаторы тональности

- Sentiment Analysis with Python NLTK Text Classification
  - Использует наивный байесовский классификатор
  - Не работает с русским языком
- http://ston.apphb.com/index.html
  - Размечает слова по тональности (с помощью словарей)
  - Работает с русским языком