

# Классификатор сообщений горячей линии вуза\*

\* <https://petrsu.ru/hotline/abit/ask>



ОПОРНЫЙ ВУЗ

ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ  
УНИВЕРСИТЕТ

ПетрГУ СЕГОДНЯ

ОБУЧЕНИЕ

НАУКА И ИННОВАЦИИ

СТУДЕНЧЕСТВО

МЕЖДУНАРОДНОСТЬ

Главная > ПетрГУ сегодня > Горячая линия

## Горячая линия

В этом разделе Вы можете задать вопросы по поступлению в Петрозаводский государственный университет (направления подготовки бакалавриата, специалитета, магистратуры).

На вопросы отвечает начальник отдела по организации приема студентов Семенова Марина Николаевна.

По вопросам поступления в аспирантуру обращайтесь на «горячую линию» по приему в аспирантуру.

Имя \*

E-mail \*

Город \*

Страна \*

Вопрос \*

# Горячая линия ПетрГУ

02.09.2021 19:49

**Денис:** Здравствуйте! Нужно ли сдавать профильную математику, чтобы поступить на юриспруденцию в ПетрГУ? Заранее спасибо!

*Петрозаводск, Россия*

Здравствуйте! Если в качестве вступительного испытания выступает математика, то это только профильная математика. Если в перечне вступительных испытаний нет математики (например, у медиков - химия, биология, русский язык), то можно сдавать базовую.

02.09.2021 10:18

**Артем :** Здравствуйте! Я учусь в первом медицинском вузе. Могу ли я сейчас перевестись на лечебное дело в петргу. Или могу я поступить на лечебное дело в петргу до 10 сентября?

*Петрозаводск, Россия*

Здравствуйте! Мы уже с вами разговаривали по телефону)

01.09.2021 21:35

**Софья:** Здравствуйте! Обязательно ли сдавать ЕГЭ, если я хочу поступить на очное обучение после колледжа?

*Череповец, Россия*

Здравствуйте! По правилам приема, действующим на этот год, выпускники техникумов могут поступать не по результатам ЕГЭ, а по внутренним испытаниям вуза.

# Постановка проблемы


- Сообщения с вопросами пользователей обрабатываются людьми
- В периоды повышенной нагрузки на горячую линию может поступать большое количество однотипных сообщений
- Время ожидания ответа на сообщение может превышать несколько дней
- Увеличение количества взаимодействий, происходящих дистанционно, ведет к возрастанию количества поступающих на горячую линию сообщений.



# Решение проблемы

Решение: разработка диалоговой системы, которая упростит и ускорит обработку сообщений пользователей.

Подзадача: создание классификатора входящих сообщений



Сколько бюджетных мест на  
профиль "Программная  
инженерия"?

Здравствуйте! На профиль  
"Программная инженерия"  
в 2022 году 20  
бюджетных мест. Больше  
информации по ссылке ...

Задача. Разработать классификатор сообщений в задаче многометочной классификации, который обеспечивает максимум метрике:

$$f1_{avg} = \frac{1}{N} \sum_{i=1}^N f1_i = \frac{1}{N} \sum_{i=1}^N \frac{2 * precision_i * recall_i}{precision_i + recall_i},$$

N - общее количество классов (меток)

$f1_i$  находится с помощью: [`sklearn.metrics.f1\_score\(y\_true, y\_pred, average='weighted'\)`](#), т.е. являются взвешенными (average='weighted')

Подзадачи:

1. Выбор/Разработка методов предварительной обработки текста
2. Выбор/Разработка способа векторизации текста
3. Разработка классификатора сообщений на основе одного метода ML/DL, настройка параметров классификатора
4. Применение ансамблей моделей.
5. Разработка других способов увеличения метрики

# Подготовка текстовых данных

Алгоритм подготовка текста:

1. Замена знаков препинания на пробелы.
2. Замена любых комбинаций пробельных символов одним символом пробела.
3. Текст разбивается на отдельные слова (токены).
4. Приведение всех слов к нижнему регистру.
5. Лемматизация – приведение слов к начальной (словарной) форме (напр. с помощью `ru morphology2`).
6. В любом тексте присутствует большое количество слов, которые мало способствуют определению интенции текста (предлоги, междометия, ...). Такие слова подлежат удалению из текста.
7. Замена имен собственных: конкретную фамилию на слово **ФАМИЛИЯ**, конкретное имя, на слово **ИМЯ**, ....

# Векторизация текста

## One-hot encoding

Обед: [0, ..., 0, 1, 0, ..., 0, 0, 0, ..., 0]

День: [0, ..., 0, 0, 0, ..., 0, 1, 0, ..., 0]

...

Кот: [0, ..., 0, 0, 0, ..., 0, 0, 1, ..., 0]

Размер словаря



Векторы, полученные  
прямым кодированием:

## TF-IDF



...  
**(ваш вариант)**

## Word Embeddings

Обед: [0.23, -0.77, 0.03, ..., -0.01, ..., 0.15]

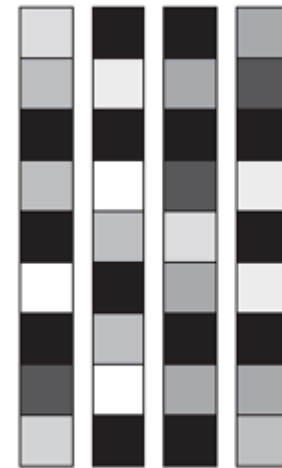
День: [0.13, -0.17, 0.15, ..., 0.15, ..., -0.11]

...

Кот: [0.71, 0.27, -0.63, ..., 0.23, ..., 0.95]



Фиксированный размер (50-1000)



Векторные представления

```
from keras.preprocessing.text import Tokenizer

samples = ['The cat sat on the mat.', 'The dog ate my homework.']
tokenizer = Tokenizer(num_words=10)
tokenizer.fit_on_texts(samples) # Создание индекса всех слов

# Преобразование строк в списки целочисленных индексов
sequences = tokenizer.texts_to_sequences(samples)
# [[1, 2, 3, 4, 1, 5], [1, 6, 7, 8, 9]]

one_hot_results = tokenizer.texts_to_matrix(samples, mode='binary')
# [[0. 1. 1. 1. 1. 1. 0. 0. 0. 0.]
#  [0. 1. 0. 0. 0. 0. 1. 1. 1. 1.]]

# Вычисленные индексы слов:
word_index = tokenizer.word_index
# {'the': 1, 'cat': 2, 'sat': 3, 'on': 4, 'mat': 5, 'dog': 6, 'ate': 7,
#  'my': 8, 'homework': 9}

print('Found %s unique tokens.' % len(word_index)) # Found 9 unique tokens.
```



**TF-IDF** (*TF* – *term frequency*, *IDF* – *inverse document frequency*) — статистическая мера, используемая для оценки важности слова в контексте документа, являющегося частью коллекции документов или корпуса. Вес некоторого слова пропорционален частоте употребления этого слова в документе и обратно пропорционален частоте употребления слова во всех документах коллекции.

Оценивается важность слова  $t_i$  в пределах отдельного документа  $d$ :

$$tf(t, d) = \frac{n_t}{\sum_k n_k},$$

где  $\sum_k n_k$  – общее количество слов в документе.

Для каждого уникального слова в пределах конкретной коллекции документов:

$$idf(t, D) = \log \frac{|D|}{|\{d_i \in D | t \in d_i\}|},$$

где  $|D|$  – количество документов в наборе,

$|\{d_i \in D | t \in d_i\}|$  – количество документов из  $D$ , в которых встречается  $t$ .

Величина основания логарифма не имеет значения, т.к. изменение основания приводит к изменению веса каждого слова на постоянный множитель, что не влияет на соотношение весов.

$$tf-idf(t, d, D) = tf(t, d) * idf(t, D)$$

# TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer

text = ["She sells seashells by the seashore", "The big sea.",
        "The seashore"] # list of text documents

vectorizer = TfidfVectorizer() # create the transform
vectorizer.fit(text) # tokenize and build vocab

# summarize
print(vectorizer.vocabulary_) # {'she': 6, 'sells': 5, 'seashells': 3, 'by': 1, 'the': 7,
                              'seashore': 4, 'big': 0, 'sea': 2}

print(vectorizer.idf_) # [1.69314718 1.69314718 1.69314718 1.69314718 1.28768207 1.69314718 1.69314718 1.]

# encode document
first_text_vector = vectorizer.transform([text[0]])

# summarize encoded vector
print(first_text_vector)
print(first_text_vector.shape) # (1, 8)
print(first_text_vector.toarray()) # [[0. 0.45050407 0. 0.45050407 0.34261996 0.45050407
0.45050407 0.26607496]]
```

one\_text\_vector=  
(0, 7) 0.2660749625405929  
(0, 6) 0.450504072643198  
(0, 5) 0.450504072643198  
(0, 4) 0.3426199591918006  
(0, 3) 0.450504072643198  
(0, 1) 0.450504072643198

0, т. к. 0-го слова 'big' нет в 1-м тексте

[https://scikit-learn.org/stable/modules/generated/sklearn.feature\\_extraction.text.TfidfVectorizer.html](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html)

# Классификаторы

RegEx



## ML/DL classification models

- DecisionTreeClassifier
- RandomForestClassifier
- Logistic regression
- SVM/SVC
- Perceptron
- ...

## *Boosting:*

- LightGBM
- CatBoost
- XGBoost
- ...

## *RNN:*

- LSTM
- GRU
- BiLSTM
- BiGRU
- ...

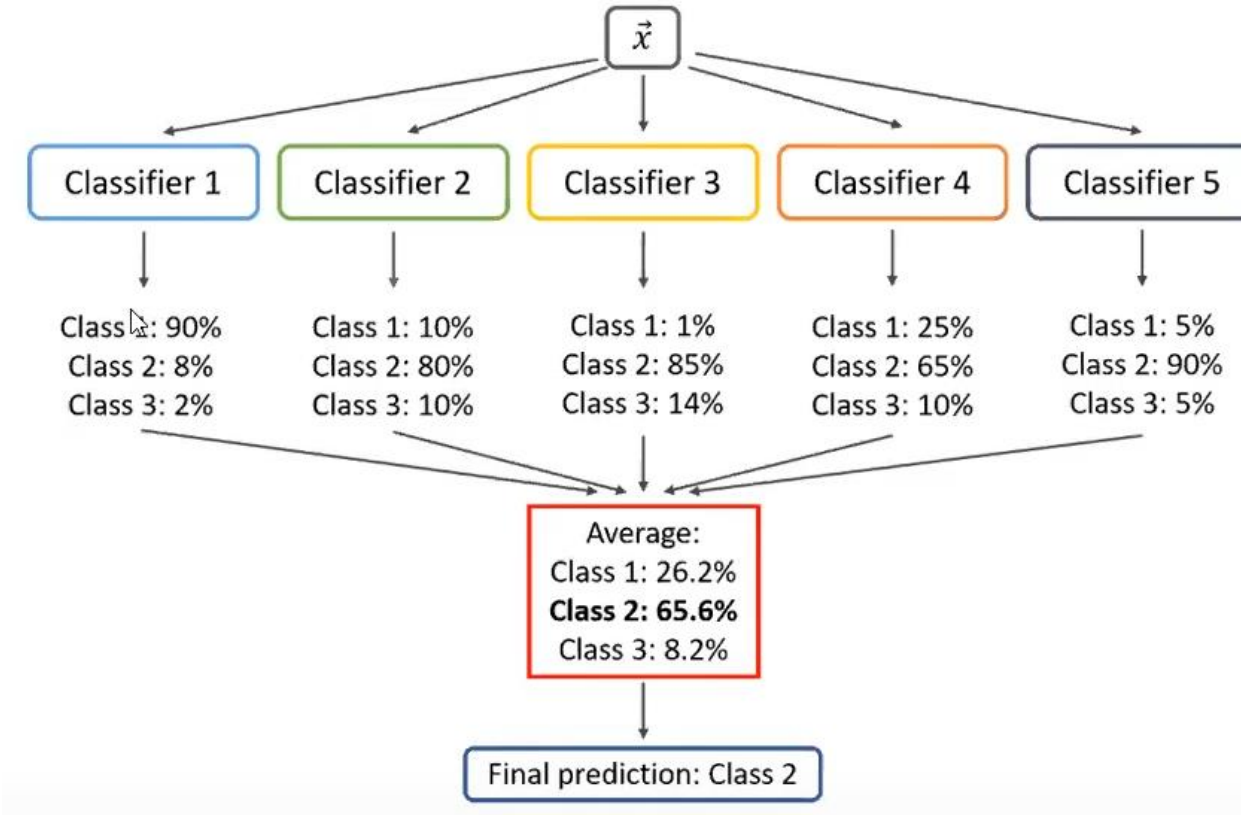
## Transformers: ! Hugging Face

- ruBert
- ELMO
- ERNIE
- ...
- **GPT-3**
- **T5**

...

# Ансамбли моделей

Ансамбль методов или моделей в ML использует несколько алгоритмов для увеличения показателей эффективности прогнозирования, чем при использовании одного метода.



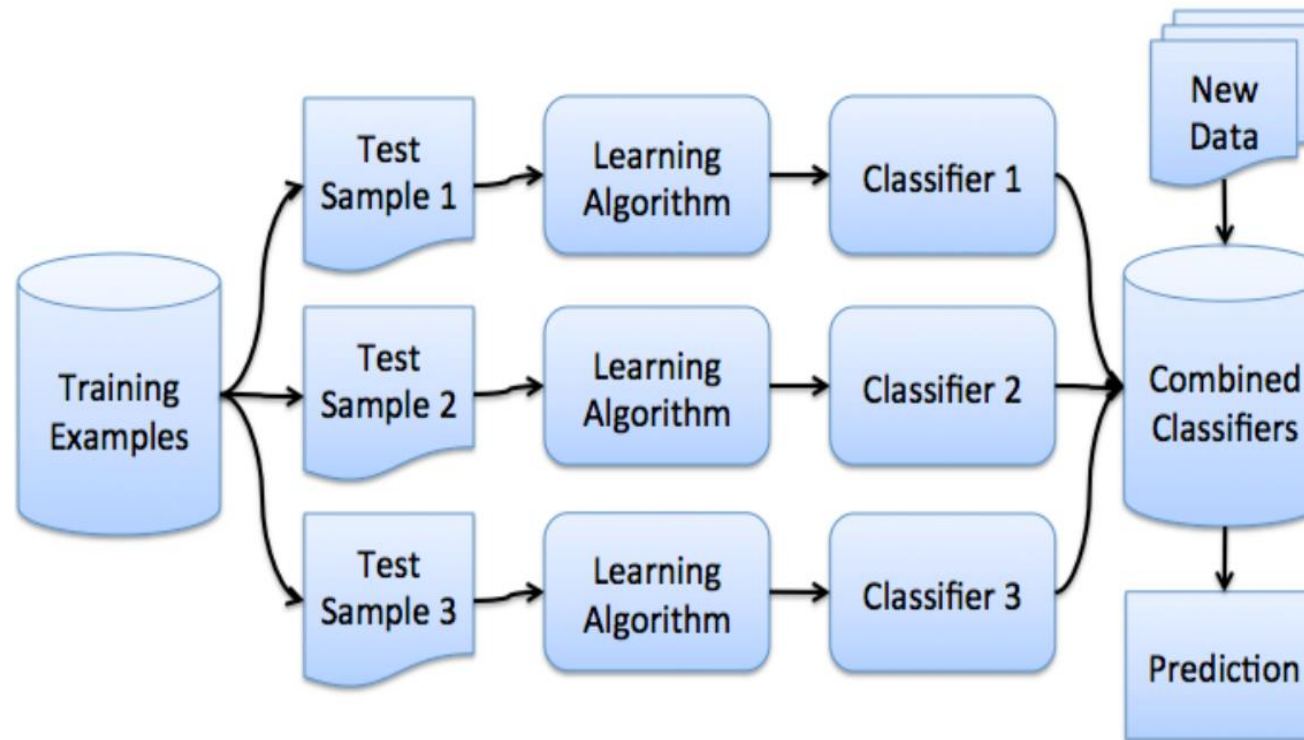
- Сильно отличающиеся методы улучшают результаты,
- «Слабые» методы будут делать ансамбль слабее, могут «усреднить» результаты сильных методов.

# Bagging

**Bagging** == **B**ootstrap **a**ggregation.

Пусть имеется обучающая выборка  $X$ . С помощью бутстрэпа сгенерируем из неё выборки  $X_1, \dots, X_M$ . На каждой выборке обучим свой классификатор  $a_i(x)$ .

Итоговый классификатор будет усреднять ответы всех этих алгоритмов (в случае классификации это соответствует голосованию): 
$$a(x) = \frac{1}{M} \sum_{i=1}^M a_i(x)$$



## Классы/метки сообщений

Номер класса	Название класса
0	Сроки
1	Документы
2	Места
3	Проходной и допустимый балл
4	Достижения
5	Льготы
6	Общежития
7	Оплата
8	Другие вопросы
9	Аспирантура
10	Перевод
11	Ошибки
12	Забрать заявление
13	Магистратура
14	Очно-заочное и зачное обучение

**Пример:** “Здравствуйте! Подскажите, пожалуйста, программу вступительных испытаний в магистратуру и стоимость обучения по направлению 08.04.01 Строительство (Реставрация и приспособление для современного использования объектов деревянного зодчества).” -> Класс 7, 13 -> 000000010000010

# Данные:

**data\_train\_1000.xlsx :**

Содержит 967 классифицированных сообщений

1	CONTENT	TYPE	CLASS
2	Здравствуйте! Есть ли специальнос	Горячая линия	4
3	Здравствуйте! Задавала вопрос о к	Горячая линия	8
4	Здравствуйте! Я хотел бы задать вс	Виртуальная приемная	3
49	Здравствуйте! Какой проходной ба	Горячая линия	3 5

**data\_without\_labels.xlsx :** Содержит все слова из train и test выборок.  
(необязателен к использованию)

**data\_test.xlsx :** тестовая выборка (появится на гугл диске в 8:30 7 августа)

# Требуемые результаты.

Выслать до 2022.08.07 9:30 на [smirnov\\_work@mail.ru](mailto:smirnov_work@mail.ru):

1. Название команды, ФИО участников.
2. Итоговую f1\_score на data\_test.xlsx
3. .ipynb-блокнот решения
4. .docx-файл с текстовым описанием разработанного классификатора (метод, тюнинг параметров, ...), вашими реализованными идеями, которые повысили метрику классификации.

## Оценивание

- Критерий победы: максимум метрики f1\_score на data\_test.xlsx
- Если максимальное значение f1\_score у нескольких команд, то оцениваются реализованные идеи команды, которые повысили метрику классификации.



Успехов в решении!!!

На следующих слайдах  
дополнительная информация с  
описанием и примерами применения  
ML методов:

# Сравнения решающего дерева, бэггинга и случайного леса в задаче классификации

```
import matplotlib.pyplot as plt
import numpy as np; np.random.seed(42)
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import make_circles # Make a large circle containing a smaller circle in 2d (toy dataset)
from sklearn.model_selection import train_test_split
plt.style.use('ggplot')
plt.rcParams['figure.figsize'] = 10, 6

# X - координаты центров кругов, y={0;1} - метки классов -> бинарная классификация
X, y = make_circles(n_samples=500, factor=0.1, noise=0.35, random_state=42)
X_train_circles, X_test_circles, y_train_circles, y_test_circles = train_test_split(X, y, test_size=0.2)

dtree = DecisionTreeClassifier(random_state=42)
dtree.fit(X_train_circles, y_train_circles)

# Получаем новые центры кругов
x_range = np.linspace(X.min(), X.max(), 100)
xx1, xx2 = np.meshgrid(x_range, x_range) # xx1.shape=(100,100), xx2.shape=(100,100)

# Классифицируем круги
y_hat = dtree.predict(np.c_[xx1.ravel(), xx2.ravel()]) # y_hat.shape = 10000
y_hat = y_hat.reshape(xx1.shape) # y_hat.shape = (100, 100)

# Отрисовка кругов
plt.contourf(xx1, xx2, y_hat, alpha=0.2) # Отрисовать область, разграничивающую два класса
plt.scatter(X[:,0], X[:,1], c=y, cmap='autumn') # Отрисовать круги
plt.title("Дерево решений")
plt.show()

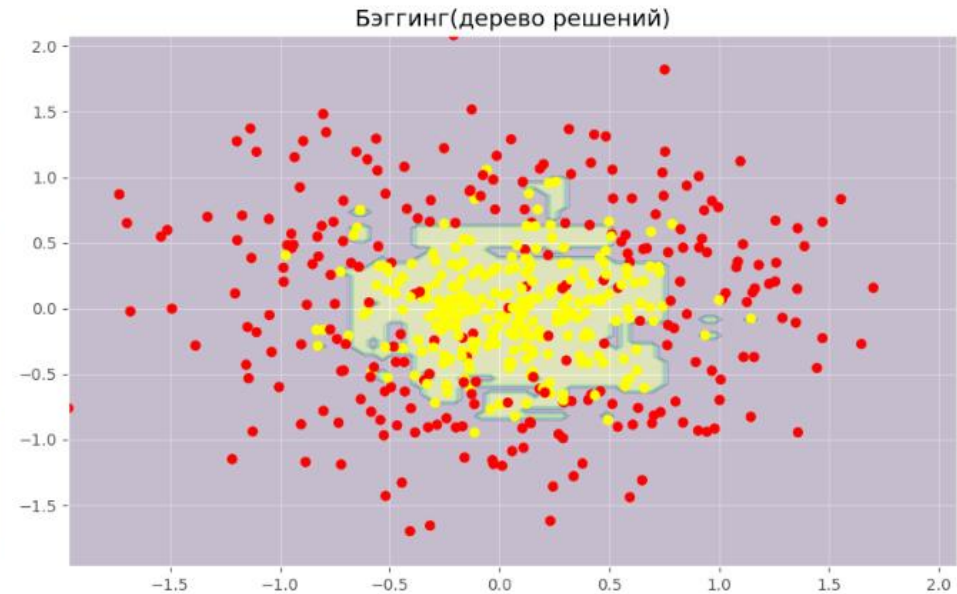
b_dtree = BaggingClassifier(DecisionTreeClassifier(), n_estimators=300, random_state=42)
b_dtree.fit(X_train_circles, y_train_circles)

x_range = np.linspace(X.min(), X.max(), 100)
xx1, xx2 = np.meshgrid(x_range, x_range)
y_hat = b_dtree.predict(np.c_[xx1.ravel(), xx2.ravel()])
y_hat = y_hat.reshape(xx1.shape)
plt.contourf(xx1, xx2, y_hat, alpha=0.2)
plt.scatter(X[:,0], X[:,1], c=y, cmap='autumn')
plt.title("Бэггинг(дерево решений)")
plt.show()

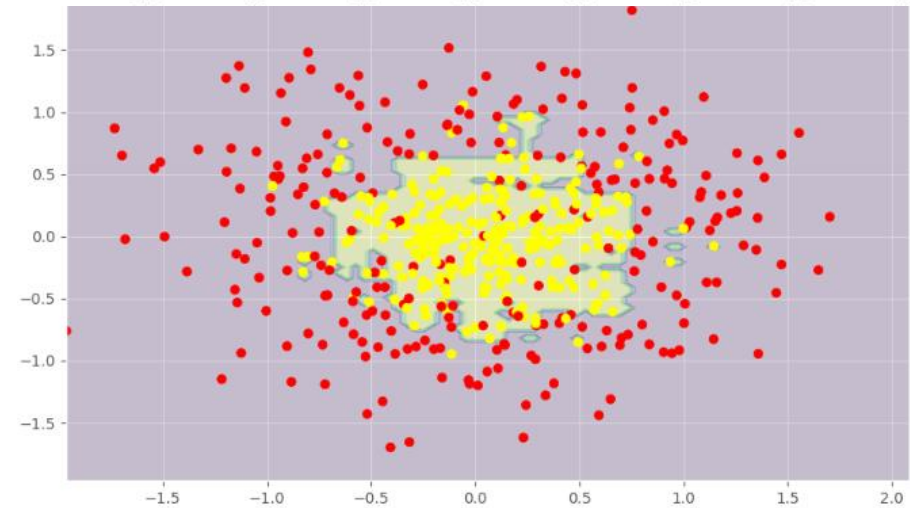
rf = RandomForestClassifier(n_estimators=300, random_state=42)
rf.fit(X_train_circles, y_train_circles)

x_range = np.linspace(X.min(), X.max(), 100)
xx1, xx2 = np.meshgrid(x_range, x_range)
y_hat = rf.predict(np.c_[xx1.ravel(), xx2.ravel()])
y_hat = y_hat.reshape(xx1.shape)
plt.contourf(xx1, xx2, y_hat, alpha=0.2)
plt.scatter(X[:,0], X[:,1], c=y, cmap='autumn')
plt.title("Случайный лес")
plt.show()
```

# Сравнения решающего дерева, бэггинга и случайного леса в задаче классификации



Разделяющая граница дерева решений «рваная» и на ней много острых углов, что говорит о переобучении и слабой обобщающей способности. У бэггинга и случайного леса граница достаточно сглаженная и практически нет признаков переобучения.



# Параметры RF

`class sklearn.ensemble.RandomForestRegressor(`

**n\_estimators** – число деревьев (по дефолту – 10)

**criterion** – функция, которая измеряет качество разбиения ветки дерева (по дефолту – "mse", так же можно выбрать "mae")

**max\_features** – число признаков, по которым ищется разбиение. Можно указать конкретное число или % или выбрать из: "auto" (все признаки), "sqrt", "log2". По дефолту стоит "auto"

**max\_depth** – максимальная глубина дерева (по дефолту глубина не ограничена)

**min\_samples\_split** – минимальное число объектов, необходимое для разделения внутреннего узла. Можно задать числом или процентом от общего числа объектов (по дефолту – 2)

**min\_samples\_leaf** – минимальное число объектов в листе. Можно задать числом или % от общего числа объектов (по дефолту – 1)

**min\_weight\_fraction\_leaf** – минимальная взвешенная доля от общей суммы весов (всех входных объектов) должна быть в листе (по дефолту имеют одинаковый вес)

**max\_leaf\_nodes** – максимальное число листьев (по дефолту нет ограничения)

**min\_impurity\_split** – порог для остановки наращивания дерева (по дефолту 1e-7)

**bootstrap** – применять ли бустрэп для построения дерева (по дефолту True)

**oob\_score** – использовать ли out-of-bag объекты для оценки  $R^2$  (по дефолту False)

**n\_jobs** – количество ядер для построения модели и предсказаний (по дефолту 1, если поставить -1, то будут использоваться все ядра)

**random\_state** – начальное значение для генерации случайных чисел (по дефолту его нет, для воспроизведения результатов нужно указать любое число типа int).

**verbose** – вывод логов по построению деревьев (по дефолту 0)

**warm\_start** – использовать ли уже натренированную модель (по дефолту False)

# Параметры RF

В RandomForestClassifier для задачи классификации все те же параметры, отличаются лишь:

`class` sklearn.ensemble.RandomForestClassifier(

**criterion** – по дефолту выбран критерий **"gini"**, можно выбрать **"entropy"**

**class\_weight** – вес каждого класса (по дефолту все веса равны 1, но можно передать словарь с весами, либо явно указать **"balanced"**, тогда веса классов будут равны их исходным частям в генеральной совокупности; также можно указать **"balanced\_subsample"**, тогда веса на каждой подвыборке будут меняться в зависимости от распределения классов на этой подвыборке)

Параметры, на которые в первую очередь стоит обратить внимание при построении модели:

- **n\_estimators** – число деревьев в "лесу"
- **criterion** – критерий для разбиения выборки в вершине
- **max\_features** – число признаков, по которым ищется разбиение
- **min\_samples\_leaf** – минимальное число объектов в листе
- **max\_depth** – максимальная глубина дерева

# GridSearchCV

`from sklearn.model_selection import GridSearchCV`

**estimator** – estimator **object** (Метод, параметры которого настраиваем).

**param\_grid** – **dict or list** of dictionaries. Словарь параметров со значениями для перебора.

**scoring** – str, callable, list, tuple **or** dict, default=**None**. Оценка модели

**n\_jobs** – int, default=**None**. Количество задействованных ядер процессора. **None**==1, -1==все ядра процессора.

**pre\_dispatch** – int, **or** str, default=n\_jobs. Управление количеством задач на параллельном выполнении. **None** == все задания будут созданы сразу, int == точное количество созданных задач, str == выражение от n\_jobs, как в **"2 \* n\_jobs"**.

**cv** – int, cross-validation generator **or** an iterable, default=**None**.

cross-validation splitting strategy. **None** == 5-fold cross validation, int == точное количество фолдов (Stratified)KFold, есть и другие ...

**refit** – **bool**, str, **or** callable, default=**True**. Переобучить метод используя лучшие значения параметров.

**verbose** – int. Количество выводимой информации при обучении.

> **1**: отображается время вычисления каждого фолда и каждого значения параметра,

> **2**: также отображается оценка;

> **3**: индексы фолдов и параметров отображаются вместе со временем начала вычисления.

**error\_score** – **'raise'** **or** numeric, default=np.nan. Значение score (оценки) в случае возникновения ошибки. If **set** to **'raise'**, the error **is** raised. If a numeric value **is** given, FitFailedWarning **is** raised.

**return\_train\_score** – bool, default=**False**. If **False**, the cv\_results\_ attribute will **not** include training scores. Computing training scores **is** used to get insights on how different parameter settings impact the overfitting/underfitting trade-off.

```

import pandas as pd
from sklearn.model_selection import cross_val_score, StratifiedKFold, GridSearchCV, train_test_split
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier

# Загружаем данные
df = pd.read_csv(".././../...csv")

# Разделяем на признаки и объекты
X_train, X_test, Y_train, Y_test = train_test_split(df.drop(columns='class'), df['class'],
                                                    test_size=0.3, stratify=df['class'])

X_train = X_train.astype(float) # Может понадобится и для test тоже

# Инициализируем наш классификатор
rfc = RandomForestClassifier(bootstrap=False, random_state=42, n_jobs=-1, criterion='entropy')

# Задаем словарь для перебора значений параметров
params = {'n_estimators': [150, 155, 160]}

clf = GridSearchCV(rfc, params, scoring='f1', verbose=50, n_jobs=-1)
clf.fit(X_train, Y_train)
"""
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   6.0min
...
[Parallel(n_jobs=-1)]: Done  15 out of  15 | elapsed: 52.3min finished

GridSearchCV(cv=None, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=False, ccp_alpha=0.0,
             class_weight=None, criterion='entropy', 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=None, oob_score=False, random_state=None, verbose=0,
             warm_start=False),
             iid='deprecated', n_jobs=1,
             param_grid={'n_estimators': 150, 155, 160}},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='f1', verbose=50)
"""

# для предсказания используется классификатор с наилучшим recall
best_clf = clf.best_estimator_

# метрики результата предсказания
Y_pred = best_clf.predict(X_test)
print(classification_report(Y_test, Y_pred))

```

# BaggingClassifier

**base\_estimator** – object, default=**None**. **None**==DecisionTreeClassifier.  
**n\_estimators** – int, default=**10**. Количество алгоритмов в ансамбле.  
**max\_samples** – int **or** float, default=**1.0**. Количество экземпляров из X на обучение каждого алгоритма. **int** == указанное количество. **float** ==  $\text{max\_samples} * X.\text{shape}[0]$  samples.  
**max\_features** – int **or** float, default=**1.0**. **int** == наибольшее количество рассматриваемых признаков. **float** ==  $\text{max\_features} * X.\text{shape}[1]$  features.  
**Bootstrap** – bool, default=**True**. Применить ли бутстрап при отборе экземпляров выборки.  
**bootstrap\_features** – bool, default=**False**. Применить ли бутстрап при отборе признаков.  
**oob\_score** – bool, default=**False**. Whether to use out-of-bag samples to estimate the generalization error.  
**warm\_start** – bool, default=**False**. **True** == продолжить обучение предыдущей итерации, иначе обучение нового ансамбля.  
**n\_jobs** – int, default=**None**.  
**random\_state** – int, default=**None**  
**verbose** – int, default=**0**



# BaggingClassifier

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import GridSearchCV

# Загружаем данные
df = pd.read_csv("../.../....csv")

# Разделяем на признаки и объекты
X_train, X_test, Y_train, Y_test = train_test_split(df.drop(columns='class'), df['class'],
                                                    test_size=0.3, stratify=df['class'])

X_train = X_train.astype(float) # Может понадобится и для test тоже

# Инициализируем наш классификатор
bbc = BaggingClassifier(random_state=42)

# Задаем словарь для перебора значений параметров
params = {'n_estimators': [110], 'bootstrap': [False]}

clf = GridSearchCV(bbc, params, scoring='f1', verbose=50, n_jobs=-1)
clf.fit(X_train, Y_train)

# для предсказания используется классификатор с наилучшим recall
best_clf = clf.best_estimator_

# метрики результата предсказания
Y_pred = best_clf.predict(X_test)
print(classification_report(Y_test, Y_pred))
```

# Linear Support Vector Classification. LinearSVC

**penalty:** {'l1', 'l2'}, default='l2'. Норма штрафа.

**loss:** {'hinge', 'squared\_hinge'}, default='squared\_hinge'. Функция потерь. 'hinge' == standard SVM loss ([https://en.wikipedia.org/wiki/Hinge\\_loss](https://en.wikipedia.org/wiki/Hinge_loss)), 'squared\_hinge' == square of the hinge loss. The combination of penalty='l1' and loss='hinge' is not supported.

**dual:** bool, default=True. algorithm to either solve the dual or primal optimization problem. Prefer dual=False when n\_samples > n\_features.

**tol:** float, default=1e-4. Допуск для stopping criteria.

**C:** float, default=1.0. Параметр регуляризации. Степень применения регуляризации пропорциональная C. Must be strictly positive.

**multi\_class:** {'ovr', 'crammer\_singer'}, default='ovr'. Определяет мультiclassовую стратегию, если в y более двух классов. "ovr" trains n\_classes one-vs-rest classifiers, while "crammer\_singer" optimizes a joint objective over all classes. While crammer\_singer is interesting from a theoretical perspective as it is consistent, it is seldom used in practice as it rarely leads to better accuracy and is more expensive to compute. If "crammer\_singer" is chosen, the options loss, penalty and dual will be ignored.

**fit\_intercept** bool, default=True. Whether to calculate the intercept for this model. If set to false, no intercept will be used in calculations (i.e. data is expected to be already centered).

**intercept\_scaling:** float, default=1. When self.fit\_intercept is True, instance vector x becomes [x, self.intercept\_scaling], i.e. a "synthetic" feature with constant value equals to intercept\_scaling is appended to the instance vector. The intercept becomes intercept\_scaling \* synthetic feature weight Note! the synthetic feature weight is subject to l1/l2 regularization as all other features. To lessen the effect of regularization on synthetic feature weight (and therefore on the intercept) intercept\_scaling has to be increased.

**class\_weight:** dict or 'balanced', default=None. Set the parameter C of class i to class\_weight[i]\*C for SVC. If not given, all classes are supposed to have weight one. The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n\_samples / (n\_classes \* np.bincount(y)).

**verbose:** int, default=0

**random\_state:** int, RandomState instance or None, default=None

**max\_iter:** int, default=1000. The maximum number of iterations to be run.

# Linear Support Vector Classification. LinearSVC

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV

# Загружаем данные
df = pd.read_csv("../.../....csv")

# Разделяем на признаки и объекты
X_train, X_test, Y_train, Y_test = train_test_split(df.drop(columns='class'),
df['class'], test_size=0.3, stratify=df['class'])

X_train = X_train.astype(float) # Может понадобится и для test тоже

# Инициализируем наш классификатор
svc = LinearSVC(loss='squared_hinge', penalty='l2', dual=False, verbose=50,
random_state=42)
params = {'class_weight': [None, 'balanced']}
clf = GridSearchCV(svc, params, scoring='f1', verbose=50, n_jobs=-1)
clf.fit(X_train, Y_train)

# для предсказания используется классификатор с наилучшим recall
best_clf = clf.best_estimator_

# метрики результата предсказания
Y_pred = best_clf.predict(X_test)
print(classification_report(Y_test, Y_pred))
```

## Бустинг —

техника построения ансамблей, в которой модели применяются последовательно, причем следующая модель учится на ошибках предыдущей. Чаще используются те модели, что дают наибольшую ошибку. Модели обучаются на ошибках, совершенных предыдущими, поэтому требуется меньше времени на обучение. Градиентный бустинг — это пример бустинга.

<https://neurohive.io/ru/osnovy-data-science/gradientyj-busting/>

# Light Gradient Boosted Machine (**LightGBM**)

*Библиотека с эффективной реализацией алгоритма градиентного бустинга.*

<https://lightgbm.readthedocs.io/en/latest/>

**Алгоритм LightGBM.** Две ключевые идеи: **GOSS** и **EFB**.

**Градиентная односторонняя выборка (GOSS)** – модификация градиентного бустинга с фокусированием внимания на примерах, которые приводят к большему градиенту. GOSS исключает значительную долю экземпляров данных с небольшими градиентами и используем только остальные экземпляры для оценки прироста информации.

**Exclusive Feature Bundling (объединение взаимоисключающих признаков) EFB**, – подход объединения разрежённых (в основном нулевых) взаимоисключающих признаков, таких как категориальные переменные входных данных, закодированные унитарным кодированием. Это тип автоматического подбора признаков.

**GOSS** и **EFB** могут ускорить время обучения алгоритма до 20 раз при сохранении точности.

`pip install lightgbm`

<https://habr.com/ru/company/skillfactory/blog/530594/>

# Ансамбль LightGBM для классификации

Функция [`make\_classification\(\)`](#) создает синтетическую задачу бинарной классификации с 1000 примерами и 20 входными признаками.

```
# test classification dataset  
from sklearn.datasets import make_classification  
# define dataset  
X, y = make_classification(n_samples=1000, n_features=20,  
n_informative=15, n_redundant=5, random_state=7)
```

Оценить LightGBM с помощью повторной стратифицированной k-кратной кросс-валидации с тремя повторами и k, равным 10. На вывод: среднее и стандартное отклонения точности модели по всем повторениям и сгибам.

# LightGBM для классификации

```
# evaluate lightgbm algorithm for classification
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from lightgbm import LGBMClassifier

# define dataset
X, y = make_classification(n_samples=1000, n_features=20,
n_informative=15, n_redundant=5, random_state=7)

# define the model
model = LGBMClassifier()

# evaluate the model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv,
n_jobs=-1)

# report performance
print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

Accuracy: 0.925 (0.031)

# LightGBM для классификации

Ансамбль LightGBM подходит для всех доступных данных и позволяет вызвать функцию `predict()`, чтобы сделать прогнозы по новым данным. Пример бинарной классификации:

```
# make predictions using lightgbm for classification
from sklearn.datasets import make_classification
from lightgbm import LGBMClassifier

# define dataset
X, y = make_classification(n_samples=1000, n_features=20,
n_informative=15, n_redundant=5, random_state=7)

model = LGBMClassifier() # define the model
model.fit(X, y) # fit the model on the whole dataset

# make a single prediction
row = [0.2929949, -4.21223056, -1.288332, -2.17849815, -0.64527665, 2.58097719,
0.28422388, -7.1827928, -1.91211104, 2.73729512, 0.81395695, 3.96973717,
-2.66939799, 3.34692332, 4.19791821, 0.99990998, -0.30201875, -4.43170633,
-2.82646737, 0.44916808]

yhat = model.predict([row])
print('Predicted Class: %d' % yhat[0])
```

Out: Predicted Class: 1



# LightGBM. Исследование количества деревьев

Важный гиперпараметр LightGBM – количество деревьев решений в ансамбле. Деревья добавляются в модель последовательно для исправления и улучшения прогнозов, сделанных предыдущими деревьями. Часто работает правило: больше деревьев – лучше. Количество деревьев `n_estimators` по умолчанию равно 100. В примере исследуется влияние количества деревьев, взяты значения от 10 до 5000:

```
# explore lightgbm number of trees effect on performance
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from lightgbm import LGBMClassifier
from matplotlib import pyplot

# get the dataset
def get_dataset():
    X, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redundant=5, random_state=7)
    return X, y

# get a list of models to evaluate
def get_models():
    models = dict()
    trees = [10, 50, 100, 500, 1000, 5000]
    for n in trees:
        models[str(n)] = LGBMClassifier(n_estimators=n)
    return models

# evaluate a give model using cross-validation
def evaluate_model(model):
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
    return scores

# define dataset
X, y = get_dataset()
# get the models to evaluate
models = get_models()
# evaluate the models and store results
results, names = list(), list()
for name, model in models.items():
    scores = evaluate_model(model)
    results.append(scores)
    names.append(name)
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
# plot model performance for comparison
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()
```

>10 0.859 (0.031)  
>50 0.913 (0.027)  
>100 0.930 (0.027)  
>500 0.940 (0.027)  
>1000 0.941 (0.028)  
>5000 0.939 (0.027)



<https://catboost.ai/docs/concepts/python-quickstart.html>

<https://habr.com/ru/company/otus/blog/527554/>

`pip install catboost`

Install visualization tools:

a. Install the ipywidgets Python package (version 7.x or higher is required):

`pip install ipywidgets`

b. Turn on the widgets extension:

`jupyter nbextension enable --py widgetsnbextension`

# CatBoostClassifier

```
import numpy as np
from catboost import CatBoostClassifier, Pool

# initialize data
train_data = np.random.randint(0, 100, size=(100, 10))
train_labels = np.random.randint(0, 2, size=(100))
test_data = catboost_pool = Pool(train_data, train_labels)

model = CatBoostClassifier(iterations=2,
                           depth=2,
                           learning_rate=1,
                           loss_function='Logloss',
                           verbose=True)

# train the model
model.fit(train_data, train_labels)
# make the prediction using the resulting model
preds_class = model.predict(test_data)
preds_proba = model.predict_proba(test_data)
print("class = ", preds_class)
print("proba = ", preds_proba)
```