ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ

РАБОТА

по программе профессиональной переподготовки «Основы Data Science на языке Python»

на тему «Применение методов машинного обучения для определения ядовитости гриба по внешним признакам»

[1. Понимание целей исследования 3](#__RefHeading___1)

[1.1 Понимание проблемы 3](#__RefHeading___2)

[1.2 Доступные ресурсы 3](#__RefHeading___3)

[1.3 Риски 3](#__RefHeading___4)

[1.4 Ограничения 4](#__RefHeading___5)

[1.5 Цели исследования данных 4](#__RefHeading___6)

[1.6 Критерии успешности работы 4](#__RefHeading___7)

[2. Начальное изучение данных 4](#__RefHeading___8)

[2.1 Сбор данных 5](#__RefHeading___9)

[2.2 Описание данных 5](#__RefHeading___10)

[2.3 Исследование данных 6](#__RefHeading___11)

[3. Подготовка данных 7](#__RefHeading___12)

[4. Моделирование 11](#__RefHeading___13)

[4.1 Логистическая регрессия 11](#__RefHeading___14)

[4.2 Модель Решающего дерева 12](#__RefHeading___15)

[4.3 Модель Случайный лес 14](#__RefHeading___16)

[4.4. Модель Градиентный бустинг 15](#__RefHeading___17)

[4.5 Метод ближайших соседей 16](#__RefHeading___18)

[4.6 Наивный Байесовский метод 17](#__RefHeading___19)

[5.Результаты моделирования 19](#__RefHeading___20)

[6.Приложение. 22](#__RefHeading___21)

# Понимание целей исследования

## 1.1 Понимание проблемы

Многие люди не только любят есть грибы, но и собирать их. Сбор грибов специфическое занятие со своими нюансами и особенностями. Важной составляющей процесса является определение ядовитости гриба, так отравление грибами имеет, в том числе и летальные последствия.

На сегодняшний момент существует ряд мобильных приложений позволяющих определять по фотографии вид растения или животного. Но данные приложения ограничены собственной базой данных видов.

Подобные приложения можно дополнить модулем предсказывающим вероятность ядовитости гриба на основе внешних признаков. Данный модуль может работать обособленно от основной части программы и позволит принять более взвешенное решение при сборе грибов.

В данной работе предполагается, что мобильное приложение уже имеет развитый алгоритм обработки изображений и для типа "гриб" формирует и передает в обособленный модуль табличные данные описывающие внешние характеристики исследуемого объекта.

## 1.2 Доступные ресурсы

Для успешной реализации проекта необходимы следующие категории специалистов:

* аналитик данных,
* разработчик (в случае модернизации мобильного приложения),
* руководитель проекта (в случае модернизации мобильного приложения).

## 1.3 Риски

1. Несоблюдение сроков проекта;

  2. Риск неплатежеспособности заказчика исследования;

  3. Риск нехватки и неполноты данных получаемых от приложения;

 4. Риск несоответствия полученных результатов требованиям заказчика исследования;

## 1.4 Ограничения

Ограничение сроков: 1 месяцев (проект анализа данных)

Ставки по сотрудникам:

* Аналитик данных – 1 ставка;
* Программист – 1 ставка (в случае модернизации мобильного приложения);
* Руководитель проекта - 0,5 ставки (в случае модернизации мобильного приложения).

## 1.5 Цели исследования данных

Задачи анализа данных, решаемые в рамках проекта, следующие:

* Проведение анализа исходных данных и подготовка данных для дальнейшего моделирования.
* Построение модели с целью выявления зависимостей между исследуемыми показателями с помощью методов машинного обучения.
* Решение задачи определения ядовитости гриба по внешним признакам п с  помощью моделей машинного обучения.

## 1.6 Критерии успешности работы

* метрика "Площадь под ROC-кривой" показывает точность и качество построенных моделей. AUC равен доле пар объектов вида (объект класса 1, объект класса 0), которые алгоритм верно упорядочил, т.е. предсказание классификатора на первом объекте больше: Значения ближе к 1 предпочтительнее. Только значения больше 0,6 демонстрируют эффективность модели.
* вторым критерием будет сравнение метрик MSE (средняя квадратичная ошибка) у выбранной модели и тривиальной. За тривиальную возьмем модель, предсказывающую среднюю вероятность по обучающей выборке. Критерием качества модели будет меньшее значение MSE у выбранной модели по сравнению с тривиальной.

# 2. Начальное изучение данных

## 2.1 Сбор данных

Сбор данных осуществлялся через открытые Интернет ресурсы, а именно, с сайта <https://www.kaggle.com/>

## 2.2 Описание данных

Данные представляют собой 1 дата сет.

Объем данных это 23 параметра и 8124 исследования

Размер файла 366 Кб

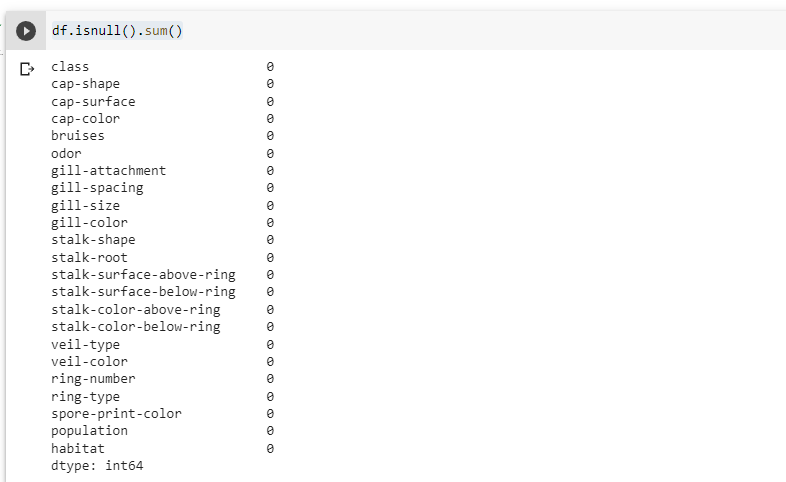
|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Наименование** | **Тип данных** | **Пояснение** |
| 1 | **class** | object | Ядовитость  edible=e, poisonous=p |
| 2 | cap-shape | object | форма шляпки  bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s |
| 3 | cap-surface | object | поверхность шляпки  fibrous=f,grooves=g,scaly=y,smooth=s |
| 4 | cap-color | object | цвет шляпки  brown=n,buff=b,cinnamon=c,gray=g,green=r,pink=p,purple=u,red=e,white=w,yellow=y |
| 5 | bruises | object | синяки  bruises=t,no=f |
| 6 | odor | object | запах  almond=a,anise=l,creosote=c,fishy=y,foul=f,musty=m,none=n,pungent=p,spicy=s |
| 7 | gill-attachment | object | крепление пластинок  attached=a, descending=d, free=f, notched=n |
| 8 | gill-spacing | object | расстояние между пластинками  close=c,crowded=w,distant=d |
| 9 | gill-size | object | размер пластинок  broad=b,narrow=n |
| 10 | gill-color | object | цвет пластинок  black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e,white=w,yellow=y |
| 11 | stalk-shape | object | форма ножки  enlarging=e,tapering=t |
| 12 | stalk-root | object | тип корня  bulbous=b,club=c,cup=u,equal=e,rhizomorphs=z,rooted=r,missing=? |
| 13 | stalk-surface-above-ring | object | поверхность ножки над кольцом  fibrous=f,scaly=y,silky=k,smooth=s |
| 14 | stalk-surface-below-ring | object | поверхность ножки под кольцом  fibrous=f,scaly=y,silky=k,smooth=s |
| 15 | stalk-color-above-ring | object | цвет ножки над кольцом  brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y |
| 16 | stalk-color-below-ring | object | цвет ножки под кольцом  brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y |
| 17 | veil-type | object | тип вуали  partial=p,universal=u |
| 18 | veil-color | object | цвет вуали  brown=n,orange=o,white=w,yellow=y |
| 19 | ring-number | object | количество колец на ножке  none=n,one=o,two=t |
| 20 | ring-type | object | тип кольца на ножке  cobwebby=c,evanescent=e,flaring=f,large=l,none=n,pendant=p,sheathing=s,zone=z |
| 21 | spore-print-color | object | цвет спор  black=k,brown=n,buff=b,chocolate=h,green=r,orange=o,purple=u,white=w,yellow=y |
| 22 | population | object | популяция  abundant=a,clustered=c,numerous=n,scattered=s,several=v,solitary=y |
| 23 | habitat | object | место произрастания  grasses=g,leaves=l,meadows=m,paths=p,urban=u,waste=w,woods=d |

**Сlass** является нашей исследуемой, таргетной переменной.

## 2.3 Исследование данных

Все переменные дата сета являются категориальными.

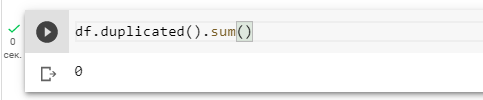
Проверим на отсутствующие данные:



Пропусков нет

Проверим данные на дубликаты.

Дублей по ID нет.

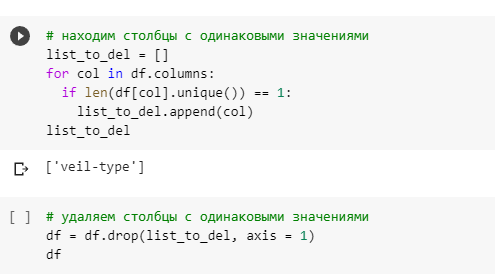


# 3. Подготовка данных

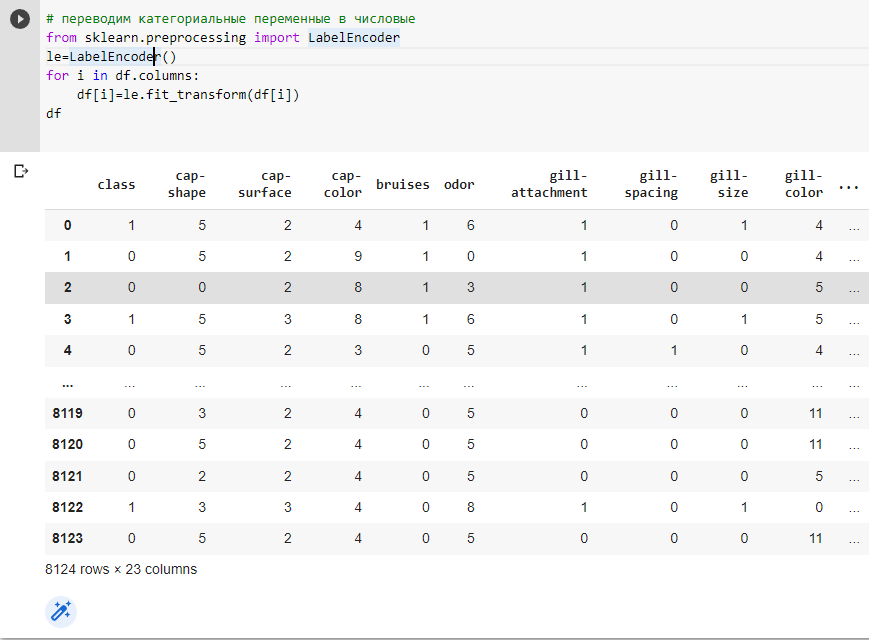
Проведем подготовку наших данных для дальнейшего применения методов МО.

Так как разработка модели проводится для мобильного приложения и статистические дынные будут получены на основании распознавания фотографии гриба, исключим из датасета параметр "Запах".

Проведем поиск столбцов с одинаковыми значениями и удалим таковые если будут найдены:



Переведем категориальные переменные в числовые:



Метод LabelEncoder сортирует по алфавиту уникальные значения, потом присваивает им порядковый номер.

В переменной Class "1" означает ядовитость гриба, "0" соответственно съедобность.

Проверка на выбросы показала что выбросов нет

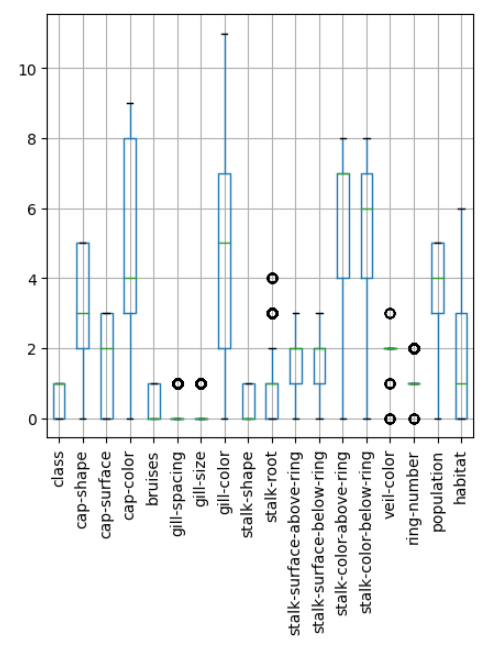


рис.1 Проверка на выбросы

Проведем поиск параметров с высокой корреляцией (более 0,6) и удалим таковые если будут найдены:

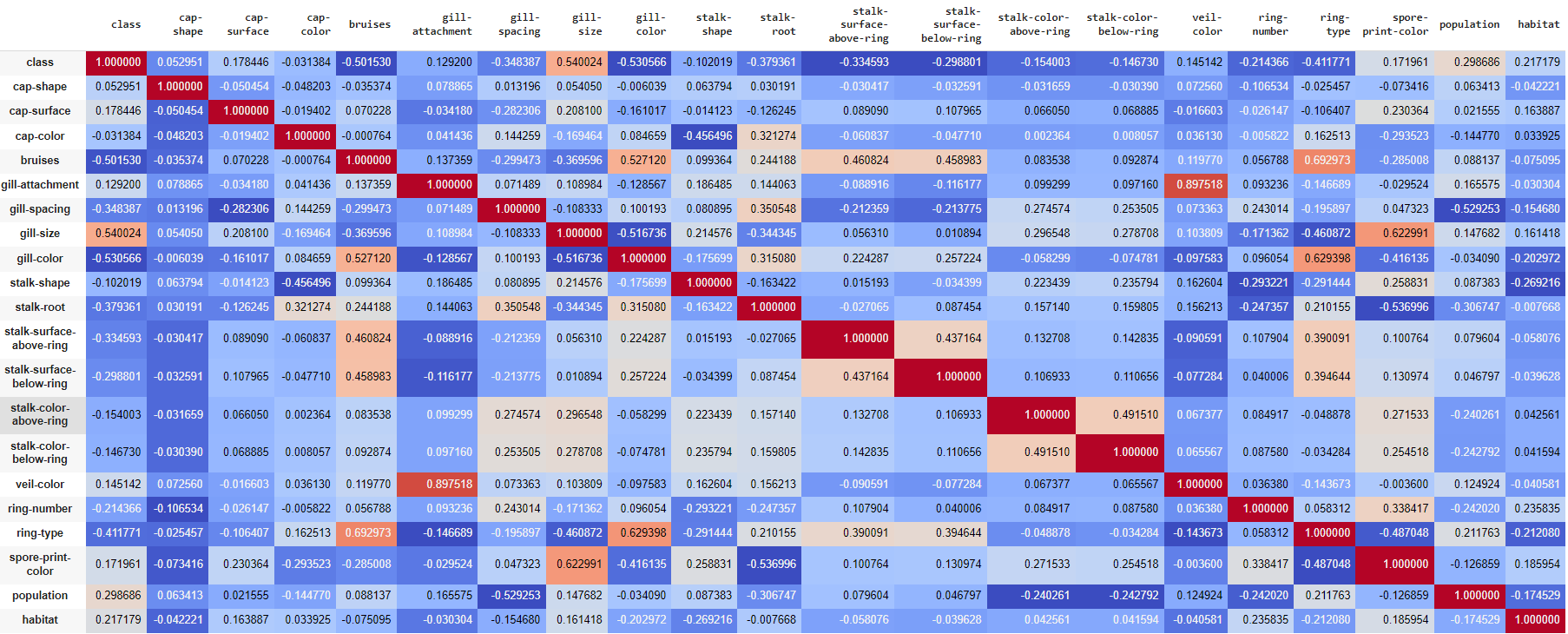
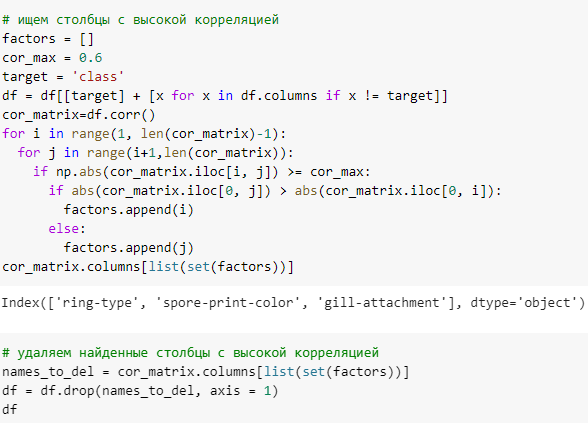
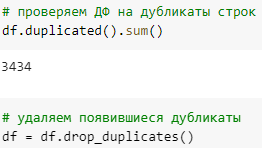


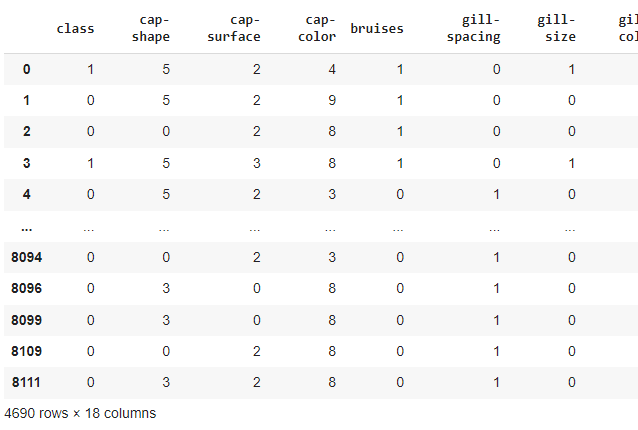
рис.2 Таблица корреляции факторов



После удаления столбцов еще раз проверим на дубликаты и удалим таковые если будут найдены:



В итоге получился следующий датасет для построения модели:



# 4. Моделирование

Для дальнейшего моделирования поделим наш тренировочный датасет на тренировочную и валидационную выборки в соотношении 80%к 20 %.

Тестовый датасет оставим для заключительного тестирования в целях избежать переобучения и завышения/ занижения результатов.

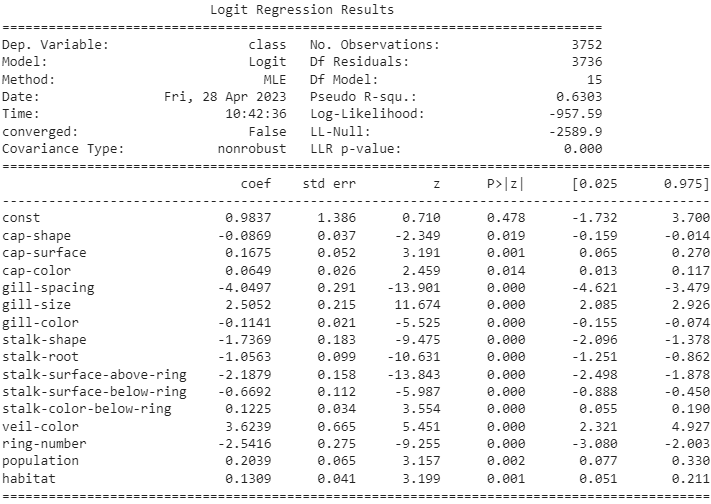
## 4.1 Логистическая регрессия

Одним из ключевых допущений множественной линейной регрессии является то, что ни одна независимая переменная в модели не имеет сильной корреляции с другой переменной . Когда две независимые переменные сильно коррелированы, это приводит к  мультиколлинеарности и может затруднить интерпретацию результатов регрессии. Факторы с сильной взаимозависимостью (выше 0,6) были удалены на этапе подготовки данных.

По значению  p-value можно сразу определить достоверность вычисленного коэффициента корреляции. Правило такое: если p-value <0,05, то  коэффициент корреляции достоверен.

Отсеем факторы в высоким p-value ( >0.05)

**Результаты логистической регрессии**



Для оценки качества построим Roc–Auc кривую и найдем площадь под ней. Сравним среднеквадратичные ошибки нашей модели и тривиальной.

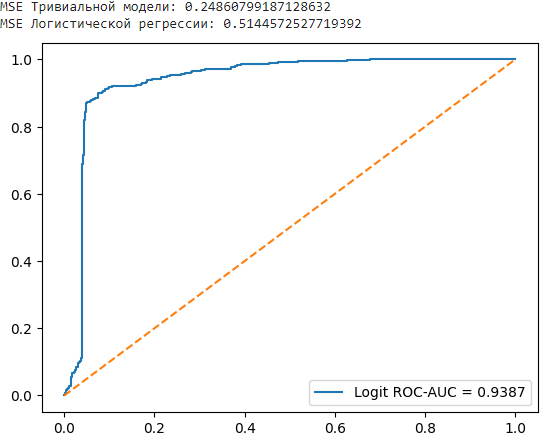


рис.3 ROC-AUC кривая логистической регрессии

## 4.2 Модель Решающего дерева

Решающее дерево предсказывает значение целевой переменной с помощью применения последовательности простых решающих правил (которые называются предикатами). Этот процесс в некотором смысле согласуется с естественным для человека процессом принятия решений

Обучим нашу модель по тренировочным данным и протестируем на тестовой выборке.

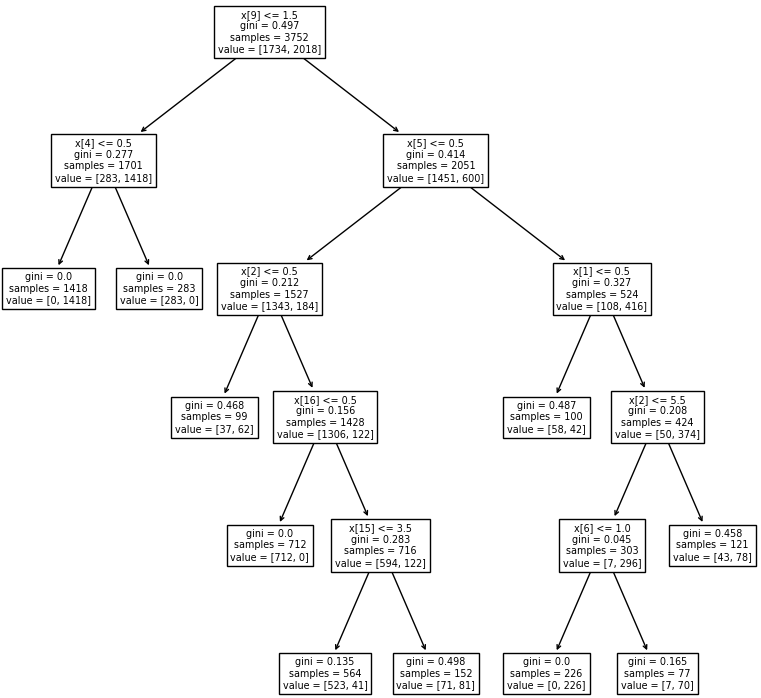


рис.4 Визуализация решающего дерева

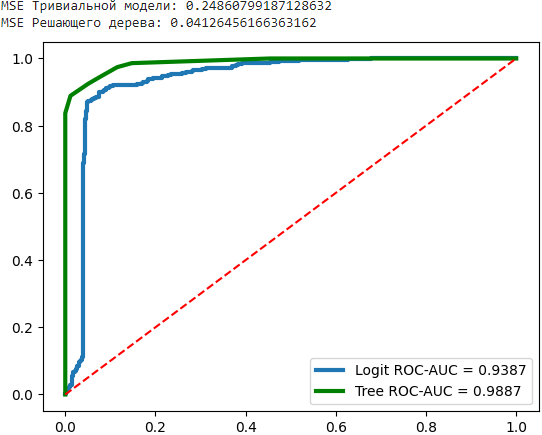


рис.5 ROC-AUC кривая логистической регрессии и Решающего дерева

## 4.3 Модель Случайный лес

Алгоритм случайного леса (**Random Forest**) — универсальный алгоритм машинного обучения, суть которого состоит в использовании ансамбля решающих деревьев. Само по себе решающее дерево предоставляет крайне невысокое качество классификации, но из-за большого их количества результат значительно улучшается.

Обучим нашу модель по тренировочным данным и протестируем на тестовой выборке.

Выведем график значимостей факторов модели:

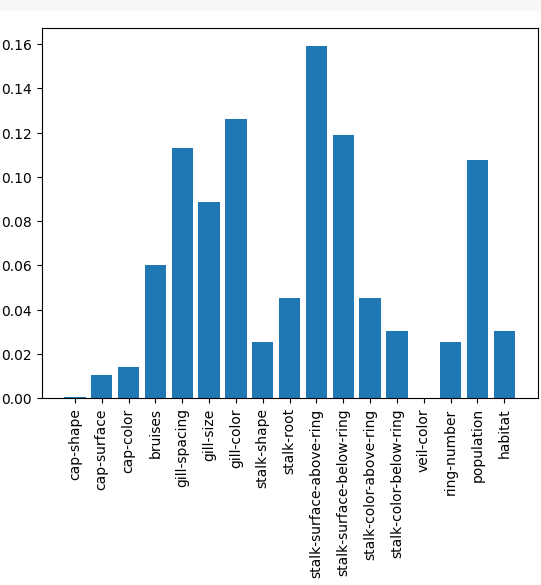


рис.6 график значимостей факторов модели

Удаляем малозначимые факторы, после переобучаем модель и предсказываем данные. В итоге получаем следующие результаты:

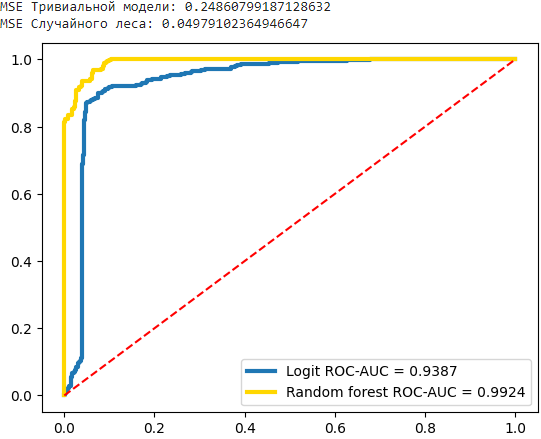


рис.7 ROC-AUC кривая логистической регрессии и Случайного Леса

## 4.4. Модель Градиентный бустинг

Градиентный бустинг представляет собой ансамбль деревьев решений. В основе данного алгоритма лежит итеративное обучение деревьев решений с целью минимизировать функцию потерь. Каждое последующее дерево компенсирует ошибки предсказания предыдущего дерева. Благодаря особенностям деревьев решений градиентный бустинг способен работать с категориальными признаками, справляться с нелинейностями.

Обучим нашу модель по тренировочным данным и протестируем на тестовой выборке.

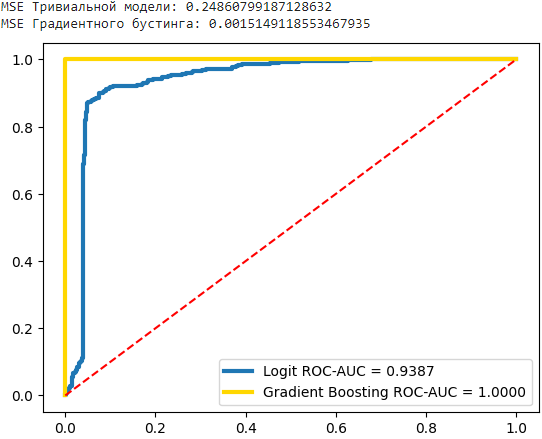


рис.8 ROC-AUC кривая лог. регрессии и Градиентного Бустинга

## 4.5 Метод ближайших соседей

Алгоритм находит расстояния между запросом и всеми примерами в данных, выбирая определенное количество примеров (**k**), наиболее близких к запросу, затем голосует за наиболее часто встречающуюся метку (в случае задачи классификации).

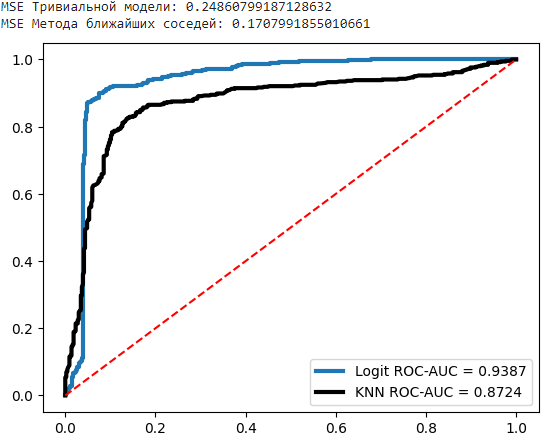
Метод чувствителен к нормализации данных и значимости факторов. Обучение и тестирование проведем на Z– нормализованных данных

рис.8 ROC-AUC кривая логистической регрессии и К-ближайших соседей

## 4.6 Наивный Байесовский метод

Как следует из названия, этот алгоритм делает предположение, что все переменные в наборе данных "наивные", т.е. не коррелируют друг с другом.

Высчитывается условная вероятность наступления события А при наступлении события Б.

Она показывает, как часто происходит событие A *при наступлении события B*, обозначается как **P(A|B)**и имеет второе название «апостериорная вероятность». При этом мы должны знать:

1. как часто происходит событие B *при наступлении события A*, что обозначается в формуле как **P(B|A)**;
2. какова вероятность того, что A не зависит от других событий, обозначаемая в формуле как **P(A)**;
3. какова вероятность того, что B не зависит от других событий. В формуле она обозначается как **P(B)**.

Метод чувствителен к взаимозависимости факторов и нормализации данных.

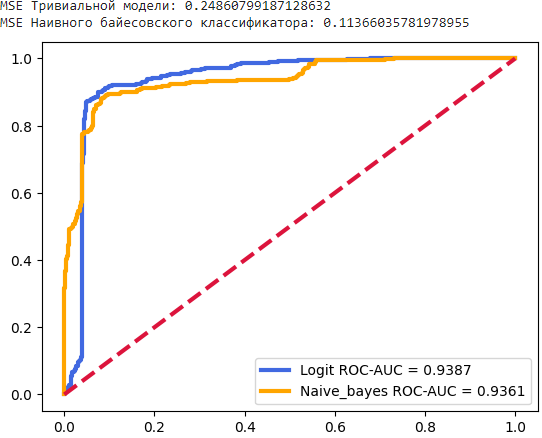


рис.9 ROC-AUC кривая логистической регрессии и Наивного байесовского классификатора

# 5.Результаты моделирования

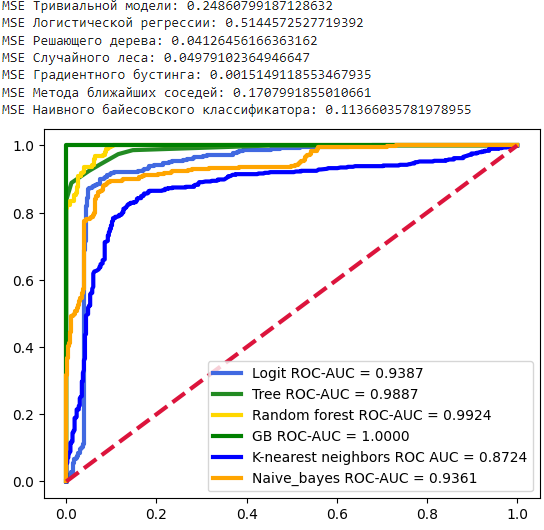


рис.10 Результирующая ROC-AUC по всем построенным моделям

Сводная таблица результатов по построенным моделям

|  |  |  |
| --- | --- | --- |
| **Наименование метода/метрики** | **Roc-Auc** | **MSE** |
| тривиальная модель | 0.5 | 0.2486 |
| Логистическая регрессия | 0.9387 | 0.5144 |
| Решающее дерево | 0.9887 | 0.0413 |
| Случайный лес | 0.9924 | 0.0480 |
| **Градиентный бустинг** | **1.000** | **0.0015** |
| Метод ближайших соседей | 0.8724 | 0.1708 |
| Наивный Байесовский метод | 0.9361 | 0.1136 |

Лучше всех себя показала модель Градиентный бустинг, за счет независимости от количества переменных (при условие, что переменных меньше чем наблюдений), вида переменных, влияния этих переменных друг на друга и возможностью обрабатывать нелинейные связи. Эта модель проста в использовании, имеет много настроек.

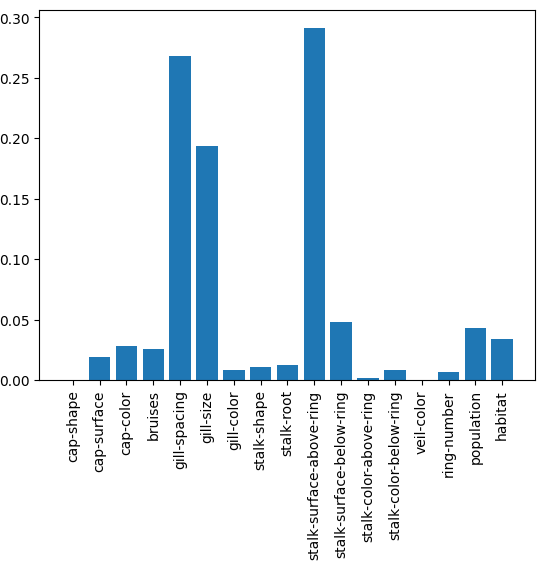


рис.11 график значимостей факторов модели Градиентный бустинг

Модель Градиентный бустинг выявила, что самыми значимыми параметрами являются "поверхность ножки над кольцом", "расстояние между пластинками", "размер пластинок"

|  |  |  |
| --- | --- | --- |
| **Фактор** | **Значение** | **Среднее значение** У |
| поверхность ножки над кольцом | волокнистый | 0.3972 |
| шелковистый | 0.9102 |
| гладкий | 0.2954 |
| чешуйчатый | 0.0499 |
| расстояние между пластинками | близко | 0.6376 |
| далеко | 0.1009 |
| размер пластинок | широкий | 0.4581 |
| узкий | 0.8136 |

## 6.Приложение.

Код на Python

# импортируем библиотеки, зашружаем датасет с гугл.диска

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from google.colab import drive

import statsmodels.api as sm

drive.mount('/content/drive')

df = pd.read\_csv('/content/drive/My Drive/REU/mushrooms.csv')

# проверяем структуру и типы данных ДФ

df.info()

# проверяем ДФ на прошущенные данные

df.isnull().sum()

# проверяем ДФ на дубликаты строк

df.duplicated().sum()

# удаляем неприменимый технически параметр

df = df.drop(['odor'], axis = 1)

# находим столбцы с одинаковыми значениями

list\_to\_del = []

for col in df.columns:

if len(df[col].unique()) == 1:

list\_to\_del.append(col)

list\_to\_del

# удаляем столбцы с одинаковыми значениями

df = df.drop(list\_to\_del, axis = 1)

# переводим категориальные переменные в числовые

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

for i in df.columns:

df[i]=le.fit\_transform(df[i])

# Проверяем на выбросы

df.boxplot(figsize = (5, 5))

plt.xticks(rotation = 90)

plt.show()

# строим корреляционную матрицу

df.corr().style.background\_gradient(cmap="coolwarm")

# ищем столбцы с высокой корреляцией

factors = []

cor\_max = 0.6

target = 'class'

df = df[[target] + [x for x in df.columns if x != target]]

cor\_matrix=df.corr()

for i in range(1, len(cor\_matrix)-1):

for j in range(i+1,len(cor\_matrix)):

if np.abs(cor\_matrix.iloc[i, j]) >= cor\_max:

if abs(cor\_matrix.iloc[0, j]) > abs(cor\_matrix.iloc[0, i]):

factors.append(i)

else:

factors.append(j)

cor\_matrix.columns[list(set(factors))]

# удаляем найденные столбцы с высокой корреляцией

names\_to\_del = cor\_matrix.columns[list(set(factors))]

df = df.drop(names\_to\_del, axis = 1)

# проверяем ДФ на дубликаты строк

df.duplicated().sum()

# удаляем появившиеся дубликаты

df = df.drop\_duplicates()

# Разбиваем ДФ после препроцессинга на тренинговую и тестовую выборки (80% к 20%)

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(df.drop('class', axis = 1), df['class'], test\_size = 0.2)

**Логистическая регрессия**

X\_train\_lg, X\_test\_lg, Y\_train\_lg, Y\_test\_lg = X\_train, X\_test, Y\_train, Y\_test

model = sm.Logit(Y\_train\_lg, sm.add\_constant(X\_train\_lg)).fit(method='bfgs')

print(model.summary())

p\_max = 0.05

while np.max(model.pvalues[1:]) > p\_max:

col = X\_train\_lg.columns[np.argmax(model.pvalues[1:])]

X\_train\_lg = X\_train\_lg.drop(col, axis = 1)

X\_test\_lg = X\_test\_lg.drop(col, axis = 1)

model = sm.Logit(Y\_train\_lg, sm.add\_constant(X\_train\_lg)).fit(method='bfgs')

print(model.summary())

# Строим РОК-кривую

from sklearn.metrics import roc\_curve, roc\_auc\_score

logit\_pred = model.predict(sm.add\_constant(X\_test\_lg))

fpr, tpr, thresholds = roc\_curve(Y\_test\_lg, logit\_pred)

logit\_auc = roc\_auc\_score(Y\_test\_lg, logit\_pred)

plt.plot(fpr, tpr, label = 'Logit ROC-AUC = %.4f' %logit\_auc)

plt.plot([0, 1], [0, 1], linestyle = '--')

plt.legend()

print('MSE Тривиальной модели:', np.mean((Y\_test - Y\_train.mean()) \*\* 2))

print('MSE Логистической регрессии:', np.mean((Y\_test - logit\_pred.reset\_index(drop = True)) \*\* 2))

**Деревья решений**

from sklearn import tree

X\_train\_tr, X\_test\_tr, Y\_train\_tr, Y\_test\_tr = X\_train, X\_test, Y\_train, Y\_test

decision\_tree = tree.DecisionTreeClassifier(max\_depth = 5, min\_weight\_fraction\_leaf=0.02, min\_impurity\_decrease=0.0001).fit(X\_train\_tr, Y\_train\_tr)

tree\_pred = decision\_tree.predict\_proba(X\_test\_tr)[:, 1]

plt.figure(figsize = (10, 10))

tree.plot\_tree(decision\_tree)

plt.show()

# Строим РОК-кривую

fpr1, tpr1, thresholds1 = roc\_curve(Y\_test\_tr, tree\_pred)

tree\_auc = roc\_auc\_score(Y\_test\_tr, tree\_pred)

plt.plot(fpr, tpr, label = 'Logit ROC-AUC = %.4f' %logit\_auc, linewidth=3)

plt.plot(fpr1, tpr1, label = 'Tree ROC-AUC = %.4f' %tree\_auc, color = 'green', linewidth=3)

plt.legend()

plt.plot([0, 1], [0, 1], color = 'red', linestyle = '--')

print('MSE Тривиальной модели:', np.mean((Y\_test - Y\_train.mean()) \*\* 2))

print('MSE Решающего дерева:', np.mean((Y\_test - tree\_pred) \*\* 2))

**Случайный лес**

from sklearn import ensemble

X\_train\_rf, X\_test\_rf, Y\_train\_rf, Y\_test\_rf = X\_train, X\_test, Y\_train, Y\_test

rf = ensemble.RandomForestClassifier(n\_estimators = 500, max\_depth = 4, min\_weight\_fraction\_leaf = 0.02, min\_impurity\_decrease = 0.0001).fit(X\_train\_rf, Y\_train\_rf)

rf\_pred = rf.predict\_proba(X\_test\_rf)[:, 1]

plt.bar(X\_train\_rf.columns, rf.feature\_importances\_)

plt.xticks(rotation = 90)

plt.show()

# удаляем малозначимые факторы, переобучаем модель и предсказываем данные

X\_train\_rf = X\_train\_rf.drop(['cap-shape', 'veil-color'], axis = 1)

X\_test\_rf = X\_test\_rf.drop(['cap-shape', 'veil-color'], axis = 1)

rf = ensemble.RandomForestClassifier(n\_estimators = 500, max\_depth = 4, min\_weight\_fraction\_leaf = 0.02, min\_impurity\_decrease = 0.0001).fit(X\_train\_rf, Y\_train\_rf)

rf\_pred = rf.predict\_proba(X\_test\_rf)[:, 1]

# Строим РОК-кривую

plt.plot(fpr, tpr, label = 'Logit ROC-AUC = %.4f' %logit\_auc, linewidth=3)

fpr2, tpr2, thresholds2 = roc\_curve(Y\_test\_rf, rf\_pred)

rf\_auc = roc\_auc\_score(Y\_test\_rf, rf\_pred)

plt.plot(fpr2, tpr2, label = 'Random forest ROC-AUC = %.4f' %rf\_auc, color = 'gold', linewidth=3)

plt.legend()

plt.plot([0, 1], [0, 1], color = 'red', linestyle = '--')

print('MSE Тривиальной модели:', np.mean((Y\_test - Y\_train.mean()) \*\* 2))

print('MSE Случайного леса:', np.mean((Y\_test - rf\_pred) \*\* 2))

**Градиентный бустинг**

X\_train\_gb, X\_test\_gb, Y\_train\_gb, Y\_test\_gb = X\_train, X\_test, Y\_train, Y\_test

gb = ensemble.GradientBoostingClassifier(n\_estimators = 300, learning\_rate = 0.02, min\_weight\_fraction\_leaf = 0.01, max\_depth = 4, min\_impurity\_decrease = 0.0001).fit(X\_train\_gb, Y\_train\_gb)

gb\_pred = gb.predict\_proba(X\_test\_gb)[:, 1]

# Строим РОК-кривую

plt.plot(fpr, tpr, label = 'Logit ROC-AUC = %.4f' %logit\_auc, linewidth=3)

fpr3, tpr3, thresholds3 = roc\_curve(Y\_test\_gb, gb\_pred)

gb\_auc = roc\_auc\_score(Y\_test\_gb, gb\_pred)

plt.plot(fpr3, tpr3, label = 'Gradient Boosting ROC-AUC = %.4f' %gb\_auc, color = 'gold', linewidth=3)

plt.legend()

plt.plot([0, 1], [0, 1], color = 'red', linestyle = '--')

print('MSE Тривиальной модели:', np.mean((Y\_test - Y\_train.mean()) \*\* 2))

print('MSE Градиентного бустинга:', np.mean((Y\_test - gb\_pred) \*\* 2))

# выводим график значимостей факторов модели

plt.bar(X\_train\_gb.columns, gb.feature\_importances\_)

plt.xticks(rotation = 90)

plt.show()

**Метод ближайших соседей**

X\_train\_knn = (X\_train - X\_train.mean()) / X\_train.std()

X\_test\_knn = (X\_test - X\_train.mean()) / X\_train.std()

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors = 1500, p = 2).fit(X\_train\_knn, Y\_train)

pred\_knn = knn.predict\_proba(X\_test\_knn)[:, 1]

# Строим РОК-кривую

plt.plot(fpr, tpr, label = 'Logit ROC-AUC = %.4f' %logit\_auc, linewidth =3)

fpr4, tpr4, thresholds4 = roc\_curve(Y\_test, pred\_knn)

kn\_auc = roc\_auc\_score(Y\_test, pred\_knn)

plt.plot(fpr4, tpr4, label = 'KNN ROC-AUC = %.4f' %kn\_auc, color = 'black', linewidth=3)

plt.legend()

plt.plot([0, 1], [0, 1], color = 'red', linestyle = '--')

print('MSE Тривиальной модели:', np.mean((Y\_test - Y\_train.mean()) \*\* 2))

print('MSE Метода ближайших соседей:', np.mean((Y\_test - pred\_knn) \*\* 2))

**Наивный байесовский классификатор**

from sklearn.naive\_bayes import GaussianNB

bayes = GaussianNB().fit(X\_train, Y\_train)

pred\_bayes = bayes.predict\_proba(X\_test)[:, 1]

# Строим РОК-кривую

fpr5, tpr5, thresholds5 = roc\_curve(Y\_test, pred\_bayes)

bayes\_auc = roc\_auc\_score(Y\_test, pred\_bayes)

plt.plot(fpr, tpr, label = 'Logit ROC-AUC = %.4f' %logit\_auc, color = 'royalblue', linewidth=3)

plt.plot(fpr5, tpr5, label = 'Naive\_bayes ROC-AUC = %0.4f' %bayes\_auc, color = 'orange', linewidth=3)

plt.legend()

plt.plot([0, 1], [0, 1], color = 'crimson', linestyle = '--', linewidth = 3)

print('MSE Тривиальной модели:', np.mean((Y\_test - Y\_train.mean()) \*\* 2))

print('MSE Наивного байесовского классификатора:', np.mean((Y\_test - pred\_bayes) \*\* 2))

# Строим общий график

plt.plot(fpr, tpr, label = 'Logit ROC-AUC = %.4f' %logit\_auc, color = 'royalblue', linewidth=3)

plt.plot(fpr1, tpr1, label = 'Tree ROC-AUC = %.4f' %tree\_auc, color = 'forestgreen', linewidth=3)

plt.plot(fpr2, tpr2, label = 'Random forest ROC-AUC = %.4f' %rf\_auc, color = 'gold', linewidth=3)

plt.plot(fpr3, tpr3, label = 'GB ROC-AUC = %.4f' %gb\_auc, color = 'green', linewidth=3)

plt.plot(fpr4, tpr4, label = 'K-nearest neighbors ROC AUC = %0.4f' %kn\_auc, color = 'blue', linewidth=3)

plt.plot(fpr5, tpr5, label = 'Naive\_bayes ROC-AUC = %0.4f' %bayes\_auc, color = 'orange', linewidth=3)

plt.legend()

plt.plot([0, 1], [0, 1], color = 'crimson', linestyle = '--', linewidth = 3)

print('MSE Тривиальной модели:', np.mean((Y\_test - Y\_train.mean()) \*\* 2))

print('MSE Логистической регрессии:', np.mean((Y\_test - logit\_pred.reset\_index(drop = True)) \*\* 2))

print('MSE Решающего дерева:', np.mean((Y\_test - tree\_pred) \*\* 2))

print('MSE Случайного леса:', np.mean((Y\_test - rf\_pred) \*\* 2))

print('MSE Градиентного бустинга:', np.mean((Y\_test - gb\_pred) \*\* 2))

print('MSE Метода ближайших соседей:', np.mean((Y\_test - pred\_knn) \*\* 2))

print('MSE Наивного байесовского классификатора:', np.mean((Y\_test - pred\_bayes) \*\* 2))