Введение:

Данные лабораторные работы предоставляются на оценку по принципу "**AS IS**" ("как есть"). Никакие гарантии на адекватность выполнения и рациональное использование методов, предусмотренных соответствующими библиотеками, не прилагаются и не предусматриваются.

Все модели сводятся к идеологии построения как в примере с Iris. Вероятно, можно абстрагироваться каким-то образом и делать иначе, но на данном этапе это не тот вариант, на который можно было бы рассчитывать.

Структура описания: скрин этапа – коммент о том, как и почему

Названия старался оставлять такие же, как и в примерах с семинаров, для наибольшей доступности восприятия лабораторной (но это не значит, что был тупой копипаст, всё это исследовано и разобрано вдоль и поперёк)

Шаг 0: установка и настройка Jupyter на компьютере. На самом деле, это уже как лабораторная работа на 25, стоит задуматься.

Лабораторная работа №3, задание №2

Задание: Набор данных Wifi_localization содержит в себе качество сигнала для каждой из семи сетей wi-fi и комнату, в которой было снято наблюдение. Построить модель, предсказывающую комнату по семи сигналам.

Решение:

1)

```
In [1]: import pandas as pd
        import graphviz
        from sklearn.model_selection import train_test_split
        from sklearn import linear_model, metrics, tree
        from IPython.display import SVG, Image, HTML
```

Подключение всех нужных библиотек.

2)

```
In [2]: wifi = pd.read_fwf('wifi_localization.txt', sep=" ", header=None)
```

Загрузка файла с данными. Гугл сказал, что загружается так, и оно даже работает.

3)

```
In [3]: for i in range(len(wifi.columns.values)):
            wifi.columns.values[i]+=1
        col = wifi.columns.values
        col = col.tolist()
        col.pop()
        col

Out[3]: [1, 2, 3, 4, 5, 6, 7]
```

Немного бесполезный по своей натуре пункт. В цикле добавляем 1 ко всем названиям столбца, потому что нулевая сеть – это некрасиво и не эстетично. После чего заносим это в col и удаляем оттуда последний столбец, потому что он никак не относится к сетям и вообще его не должно быть здесь.

4)

```
In [4]: #Y_names = np.unique(wifi[8].tolist())
        y = wifi[8].tolist()
        #y
        #Y_names = list(Y_names)
        Y_names = ['1st room','2nd room','3rd room','4rd room']
        Y_names

Out[4]: ['1st room', '2nd room', '3rd room', '4rd room']
```

Неизвестно по каким причинам, но здесь сразу заполняется два «хранилища»:

- y – данные из последнего столбца, который отображает номер сети, в котором производилось измерение
- Y_names – номер комнаты

Комментарии в блоке оставил не случайно. Закомментирован наиболее гуманный способ задания названий, если у нас там не 4 варианта, а гораздо больше. Но здесь мы можем себе позволить описать доступно, что имеется ввиду (учитывая, что названия совпадают с названиями столбцов сетей).

5)

```
In [5]: wifi.drop(8, axis=1, inplace=True)
        X=wifi.values
        X

Out[5]: array([[-64, -56, -61, ..., -71, -82, -81],
               [-68, -57, -61, ..., -71, -85, -85],
               [-63, -60, -60, ..., -76, -85, -84],
               ...,
               [-62, -59, -46, ..., -45, -87, -88],
               [-62, -58, -52, ..., -41, -90, -85],
               [-59, -50, -45, ..., -45, -88, -87]], dtype=int64)
```

Удаляем последний столбец с номерами комнат. И из оставшегося датафрейма отправляем значения в X.

6)

```
In [6]: X_train, X_test, y_train, y_test = train_test_split(X,y, shuffle=True, stratify=y, test_size=0.2)
```

Тренируем, перемешивая и выбирая в пропорциях 80:20

7)

```
In [7]: log_full = linear_model.LogisticRegression()
log_full.fit(X_train, y_train)
```

```
C:\Users\Leonid\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Default solver will be changed
to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
C:\Users\Leonid\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:460: FutureWarning: Default multi_class will be ch
anged to 'auto' in 0.22. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)
```

```
Out[7]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

Тренируем. Ничего нового.

8)

```
In [8]: full_predict = log_full.predict(X_test)
#print(y_test)
#print(full_predict)
print("Accuracy", metrics.accuracy_score(y_test, full_predict))
```

```
Accuracy 0.9725
```

Вывод вероятности правильности. Опять же, ничего нового.

9)

```
In [9]: dt_pruned = tree.DecisionTreeClassifier()
dt_pruned.fit(X_train, y_train)
dt_pred = dt_pruned.predict(X_test)
```

Формируем дерево решений. Глубина не задана специально. Если что, то да, это указывается в параметрах первого метода. Но для отладки и проверки лучше начинать с самого большого, а там можно и с глубиной ~ 4 или 5. А можно и меньше, чего уж там.

10)

```
In [10]: dot = graphviz.Source(tree.export_graphviz(dt_pruned, feature_names=col, class_names=Y_names))
svg_bytes = dot.pipe(format='svg')
no_wrap_div = '<div style="width:80% ;height:80% ;">{}</div>'
HTML(no_wrap_div.format(svg_bytes.decode()))
```

Формируем графическое решение дерева. Вывод Out[10] отсутствует здесь по религиозным причинам.

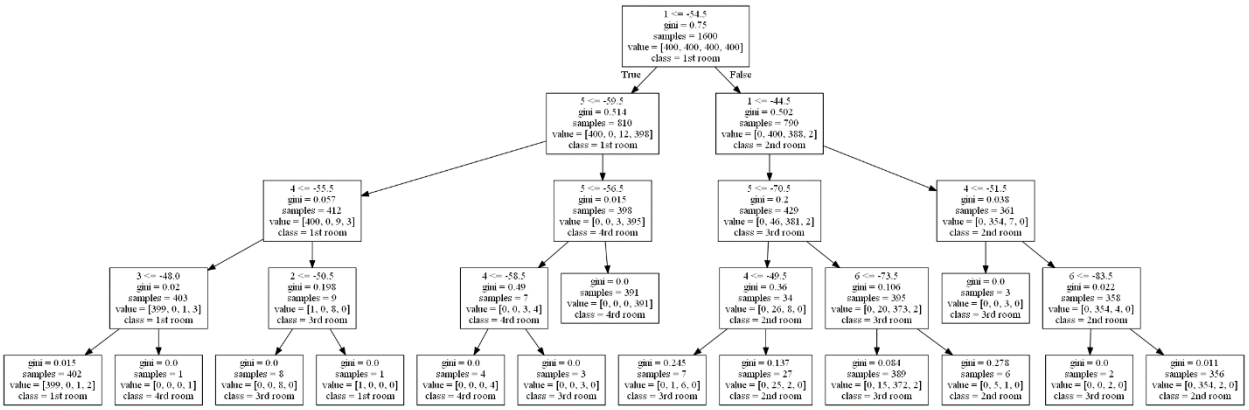
11)

```
In [11]: dot.format = 'png'
         dot.render('dtree_render',view=True)

Out[11]: 'dtree_render.png'
```

Выводим граф в .png и рассматриваем получившееся дерево решений под всеми возможными углами. Файл не вставляется, потому что в доке всё равно не видно, легче запустить и посмотреть уж там.

После непродолжительных раздумий, было принято решение вставить дерево хотя бы с глубиной = 4 для демонстрации того, что оно действительно работает и можно было бы не проверять через проект.



Соответственно, в первой строке – «если значение в «номер столбца» меньше или равно чем «значение, полученное при тренировках»»

Вывод: стоп, это всё?

Нет.

Лабораторная работа №3, задание №3

Задание: Набор данных mushrooms содержит в себе описание множества грибов по 22 атрибутам (особенности шляпки, запах, особенности пластинок, ножки, покрывала, среда обитания и др.). Построить модель, предсказывающую съедобность и ядовитость гриба по этим признакам. Отобрать наиболее важные признаки. Попытаться построить модель без самого доминирующего признака.

1)

```
In [1]: import pandas as pd
        import numpy as np
        import graphviz
        from sklearn.model_selection import train_test_split
        from sklearn import linear_model, metrics, tree
        from IPython.display import SVG, Image, HTML
```

Подгруз библиотек

2)

```
In [2]: mush = pd.read_csv('D:/mushrooms.csv')

        #mush = mush.drop(mush.columns[5],axis='columns')
```

Загрузка файла mushrooms.csv
Примечание №1: указать прямой путь, а не на файл в корень с проектом (т.е. read_csv('mushrooms.csv'), поскольку Jupyter отказывается читать файлы с папок в документах пользователя. Я НЕ ЗНАЮ почему, косяк ли это с правами доступа, или это нормальные вещи, но проще было закинуть файл в любое другое место.
Примечание №2: закомментированная надпись – неспроста. Но к ней мы вернёмся чуть позже.

3)

```
In [3]: Y_names = np.unique(mush['class']).tolist()
        y = mush['class'].values.tolist()
```

По аналогии с предыдущей лабораторной, формируем Y_names и y.

Немного обобщенной информации: предполагается, что ключевым атрибутом, как номер комнаты в той лабе, является класс «class» (дай бог здоровья за такое название, сразу отбивает желание примитивно описывать или указывать этот класс в коде, потому что это банально не работает)

4)

```
In [4]: mush = mush.drop(columns=['class'])
```

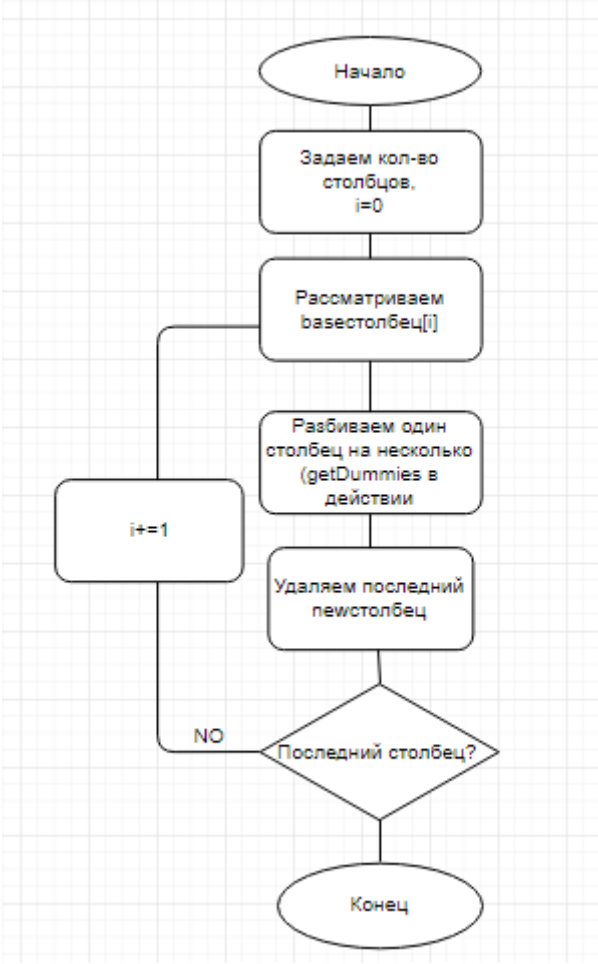
Таким же образом убираем из данных всё про «класс»

5)

```
In [5]: col = mush.columns.tolist()
        for i in range (len(col)):
            dum = pd.get_dummies(mush[col[i]],prefix=col[i])
            merged =  pd.concat([mush,dum],axis='columns')
            mush = merged.drop([col[i],merged.columns[-1]],axis='columns')
```

Экспериментальные фишки. Сначала мы в col заносим названия всех столбцов (что важно – на текущий момент), а потом каждый столбец мы разделяем.

Абстрактная блок-схемка:



ФИШЕЧКИ: удаляем последний, потому что он просто не нужен (банальная логика в действии).

6)

```
In [6]: X=mush.values
```

Задаём X, который будем обрабатывать в тестировании. «Тогда зачем нужно было задавать X в пункте 4?» - спросите вы. Он не нужен. Просто забыл удалить. Но не перекраивать же теперь из-за этого отчёт? Идём дальше.

7)

```
In [7]: y = pd.get_dummies(y)
        y = y.drop(columns='e',axis='columns')
        y = y.values.tolist()
```

Формируем y. Тактика схожа с п.5, только одна итерация. Ну и в лист перегоняем ещё.

Дальше особо комментировать нечего:

8)

```
In [6]: X_train, X_test, y_train, y_test = train_test_split(X,y, shuffle=True, stratify=y, test_size=0.2)
```

9)

```
In [7]: log_full = linear_model.LogisticRegression()
        log_full.fit(X_train, y_train)
```

```
C:\Users\Leonid\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Default solver will be changed
to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
C:\Users\Leonid\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:460: FutureWarning: Default multi_class will be ch
anged to 'auto' in 0.22. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)
```

```
Out[7]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

10)

```
In [10]: full_predict = log_full.predict(X_test)
         #print(y_test)
         #print(full_predict)
         print("Accuracy", metrics.accuracy_score(y_test, full_predict))
```

```
Accuracy 0.9993846153846154
```

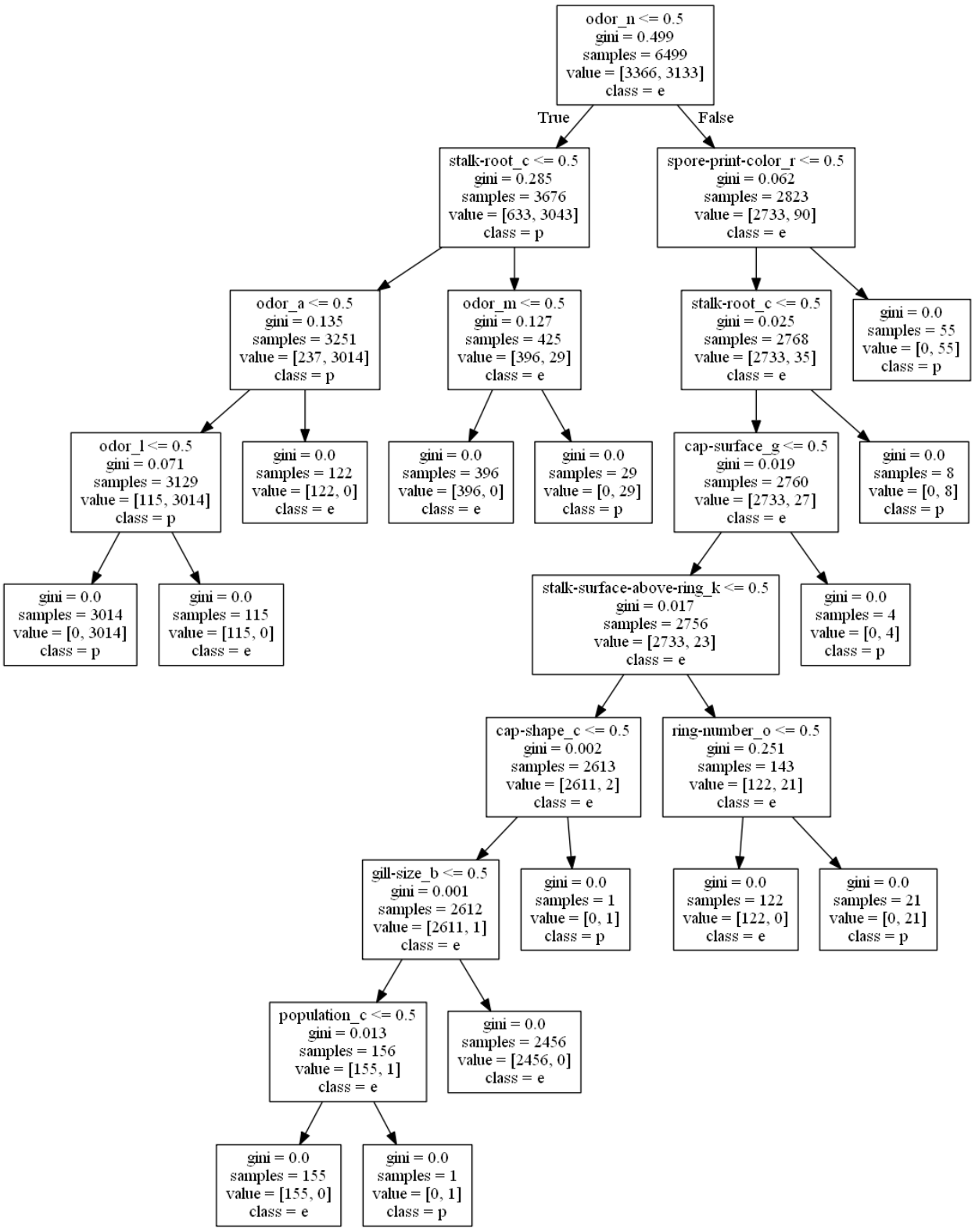
Пришлось продать душу Дьяволу и сломать кнопку перезапуска, чтобы получился ответ, не равный единице.

11)

In [11]:
```
dt_pruned = tree.DecisionTreeClassifier()
dt_pruned.fit(X_train, y_train)
dt_pred = dt_pruned.predict(X_test)
```

12)

In [12]:
```
col = mush.columns.tolist()
dot = graphviz.Source(tree.export_graphviz(dt_pruned, feature_names=col, class_names=Y_names))
svg_bytes = dot.pipe(format='svg')
no_wrap_div = '<div style="width:80% ;height:80% ;">{}</div>'
HTML(no_wrap_div.format(svg_bytes.decode()))
```



А тут вставил. ГАЙД КАК ЧИТАТЬ: первую строчку можно интерпретировать как «еслиВЭтомКлассеебуква_х <= нет», последнюю – как класс принадлежности

13)

```
In [13]: dot.format = 'png'
         dot.render('dtree_render',view=True)

Out[13]: 'dtree_render.png'
```

14)

In [14]:
```
dt_pruned.feature_importances_
importants = {}
for i in range (len(dt_pruned.feature_importances_)):
    if dt_pruned.feature_importances_[i]>0:
        importants[col[i]]=dt_pruned.feature_importances_[i]
from collections import OrderedDict
importants = OrderedDict(sorted(importants.items(), key=lambda x: x[1],reverse=True))
importants
```

Out[14]:
```
OrderedDict([('odor_n', 0.6233784439870285),
             ('stalk-root_c', 0.1756844939242213),
             ('odor_l', 0.06826648716360124),
             ('odor_a', 0.06714223974416834),
             ('spore-print-color_r', 0.03239940386080367),
             ('odor_m', 0.01665237890653905),
             ('ring-number_o', 0.011041170550687155),
             ('cap-surface_g', 0.002420599782103595),
             ('stalk-surface-above-ring_k', 0.0017831826254065882),
             ('cap-shape_c', 0.0006155639697847466),
             ('population_c', 0.0006123209663323426),
             ('gill-size_b', 3.7145193234867967e-06)])
```

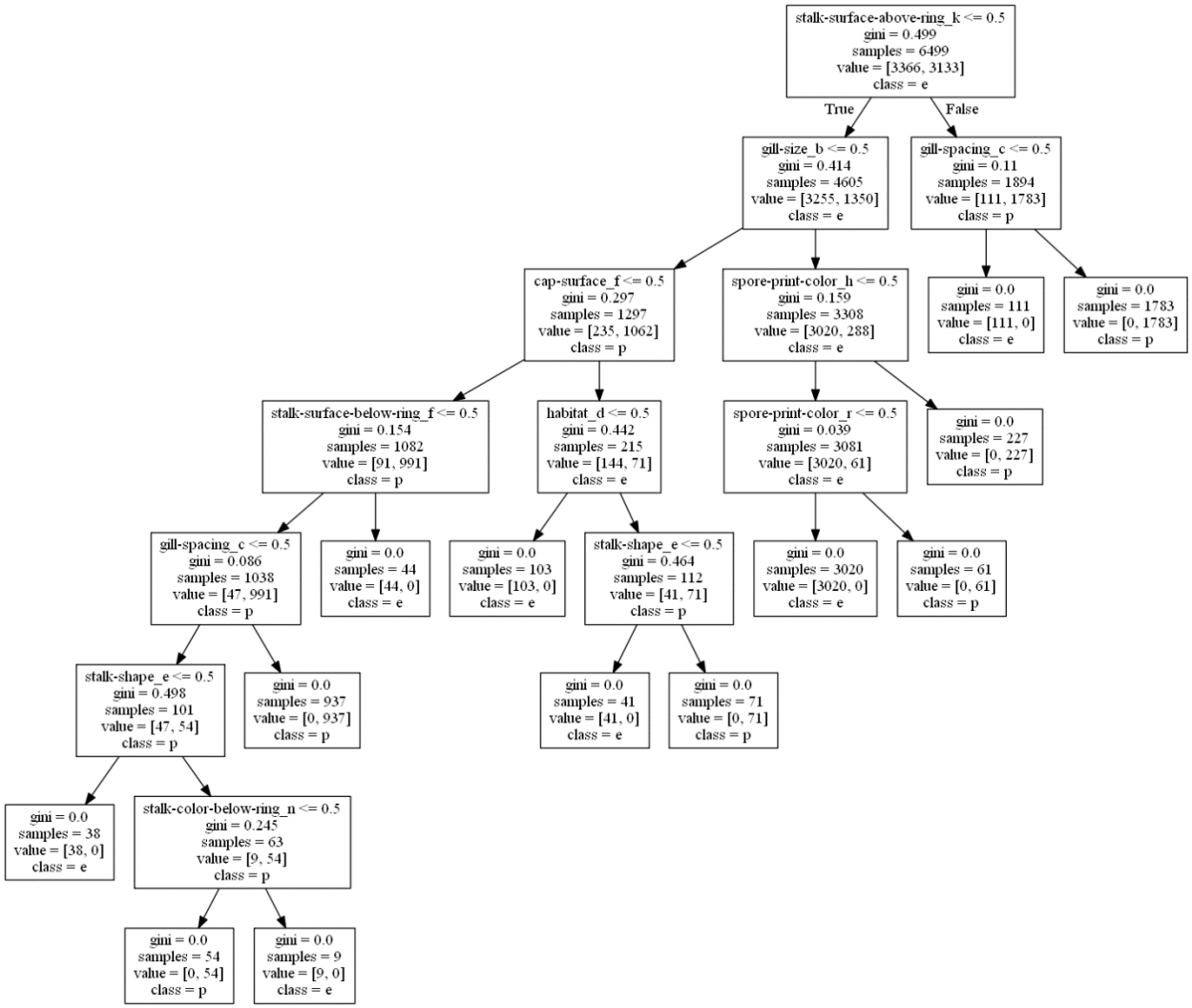
Переходим ко второй части Марлезонского балета. На основе полученного дерева, при помощи невероятного метода feature_importances_, мы получаем список всех частот использования каждого параметра. Но т.к. неиспользуемые параметры нам не интересны, мы создаём словарь и помещаем туда параметры с ненулевой вероятностью появления. Да, словарь. И потом сортируем его. Да, сортируем словарь. А почему бы и нет?

Этап 3: к сожалению, он ручной. Мы видим, что наиболее важным свойством является «odor_n». Что нам это даёт? Во-первых, важно понимать, что мы должны читать это как важное свойство «odor».

И вот теперь мы вернёмся к п.2, к закомментированной части. Мы удаляем этот параметр и проделываем АБСОЛЮТНО те же операции, ничего не изменяя. Поэтому ниже будут только интересующие нас результаты:

In [10]:
```
full_predict = log_full.predict(X_test)
#print(y_test)
#print(full_predict)
print("Accuracy", metrics.accuracy_score(y_test, full_predict))
```

```
Accuracy 0.9950769230769231
```



In [14]:
```
dt_pruned.feature_importances_
importants = {}
for i in range (len(dt_pruned.feature_importances_)):
    if dt_pruned.feature_importances_[i]>0:
        importants[col[i]]=dt_pruned.feature_importances_[i]
from collections import OrderedDict
importants = OrderedDict(sorted(importants.items(), key=lambda x: x[1],reverse=True))
importants
```

Out[14]:
```
OrderedDict([('stalk-surface-above-ring_k', 0.3475354186029787 6),
             ('gill-size_b', 0.3074499972264671),
             ('spore-print-color_h', 0.1251856574071015),
             ('gill-spacing_c', 0.0765642529499571),
             ('cap-surface_f', 0.03791357431997616),
             ('spore-print-color_r', 0.0368482706142081),
             ('stalk-shape_e', 0.02674956843055875),
             ('stalk-surface-below-ring_f', 0.02371086520654151),
             ('habitat_d', 0.01328830139853946),
             ('stalk-color-below-ring_n', 0.004754093843671585)])
```

Вывод: его нет. Вроде сделано, вроде работает. Постарался максимально объяснить, что к чему и почему. Ну ладно, отвечу, почему выбрал словарь: потому что это довольно логично использовать название столбика как ключ. А чтобы так делать, я использовал префикс названия класса в столбиках.