https://www.youtube.com/watch?v=Ca4V80jHW-c

Обзор лекции «Как решать и побеждать в соревнованиях по анализу данных»

# Понимание задачи

- Определение и формулировании бизнес задачи
- Оценка рисков, затрат, общего профита
- Постановка DM целей
- Определение критериев успешности
- Выработка плана решения задач
- Четкое понимание того что надо предсказать

# Понимание данных: Первый шаг

- Сбор данных
- Какие данные есть: сколько примеров, сколько признаков, какие признаки по природе
- Достаточно ли данных для решения задачи, есть ли необходимость собирать дополнительные данные

# Понимание данных: Второй шаг

❖Описательные статистики

$$\mu_x = \frac{\sum_{i=1}^{n} x_i}{n} \qquad s = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \mu_x)^2}{n-1}}$$

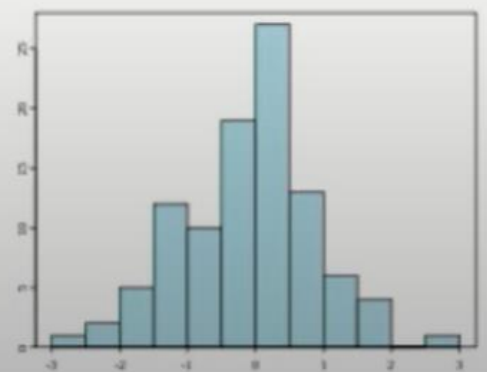
❖Корреляции Пирсона, Спирмена

$$r_{xy} = \frac{\sum_{i=1}^{m} (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{m} (x_i - \bar{x})^2 \sum_{i=1}^{m} (y_i - \bar{y})^2}} = \frac{cov(x,y)}{\sqrt{s_x^2 s_y^2}},$$

❖Проверка статистических гипотез (нормальность, проверку на распределение)

❖Исследование распределений: гистограммы признаков, таргетов

$$F = \frac{D_1}{D_2} = \frac{\sigma_1^2}{\sigma_2^2}$$



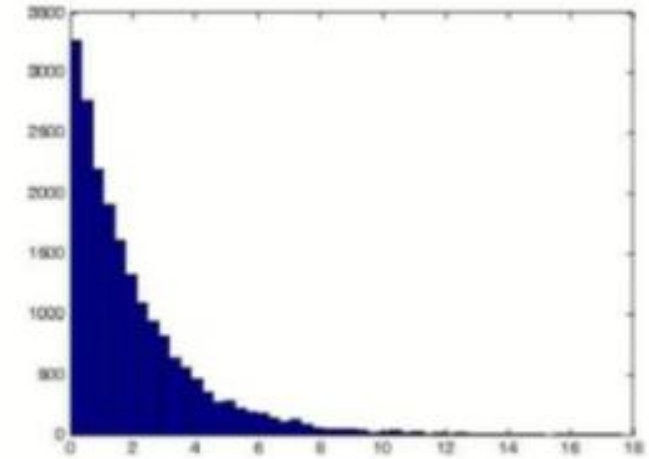
# Подготовка данных

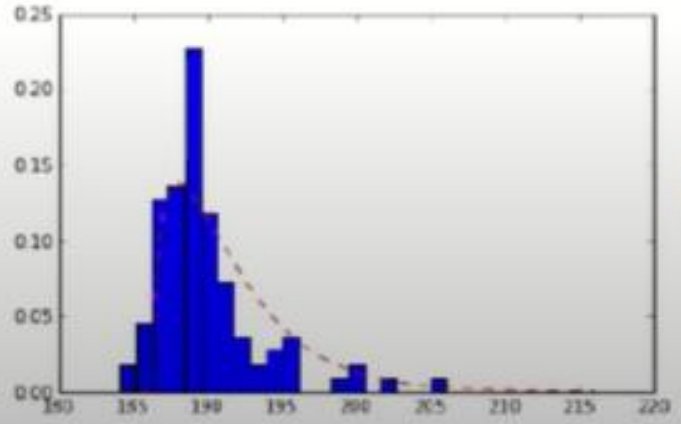
- Выбор и интеграция данных
- Форматирование данных
- Предобработка данных: заполнение пропусков, определение выбросов, нормализация данных, т.д.
- Выбор/экстракция признаков, сокращение размерности
- Инженерия признаков
- Разбиение на тренировочное, тестовое множества

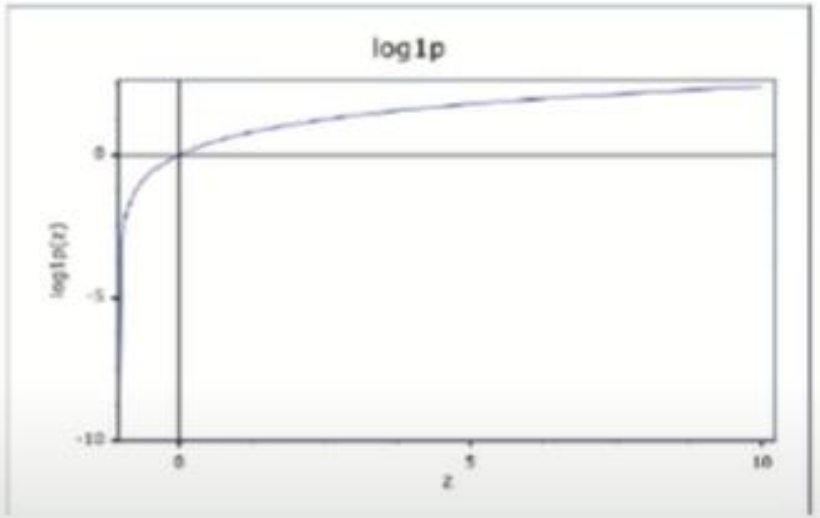
# Оценка качества моделей

- Анализ эффективности моделей на тестовом множестве: статистические гипотезы, корреляции
- Вычисление метрик оценки качества моделей
- Проверка overfitting/underfitting
- Постпроцессинг
- Достигнут ли желаемый результат?

# Исследование распределений: Линеаризация



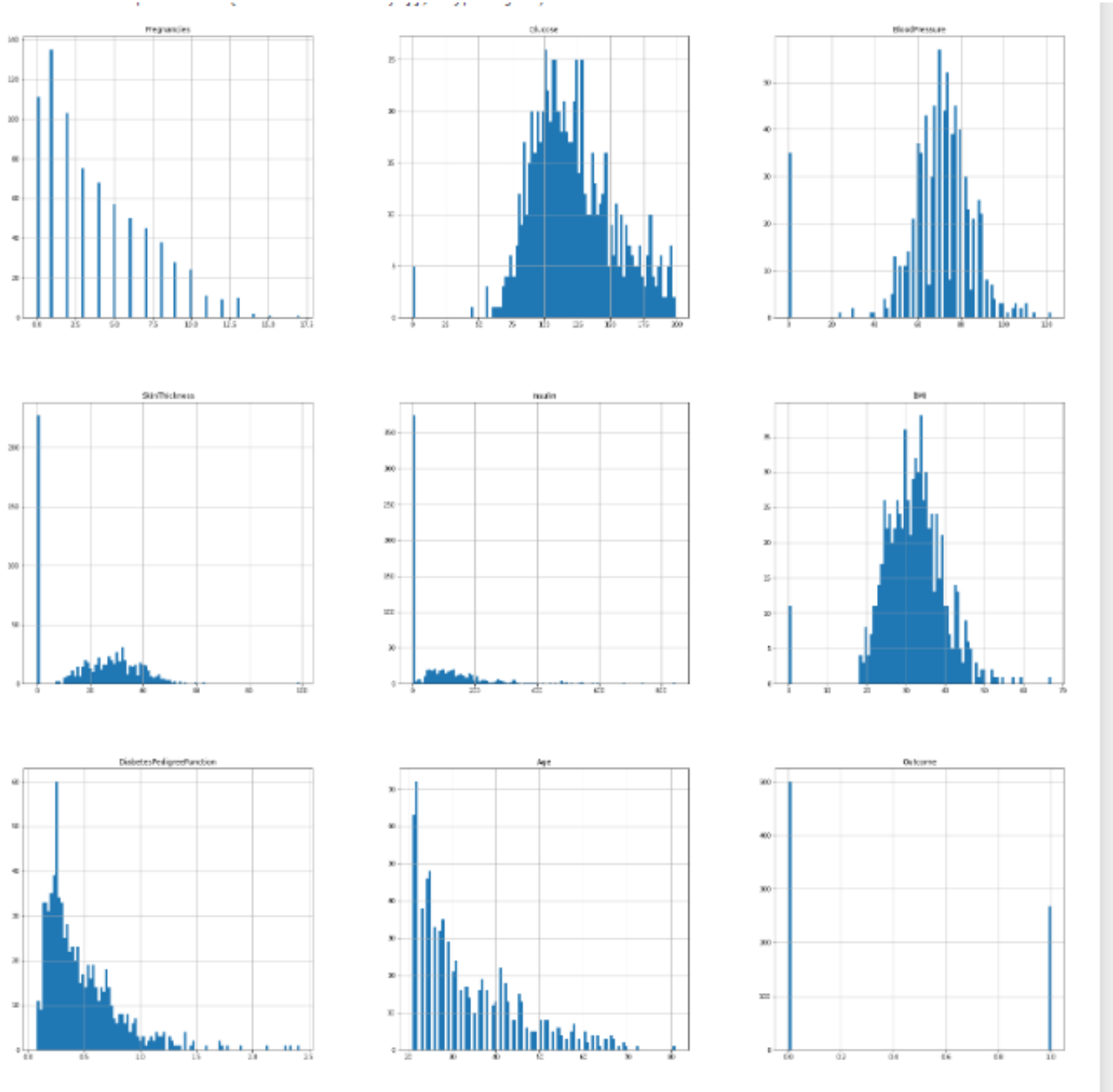





Если распределение экпоненциальное или вейбула то обязательно линеаризовать (через логарифм)

# Histogram of features

```
data.hist(bins=100,figsize=(30,30))
```



# Предобработка данных

❖Преобразование категориальных переменных:
sklearn.preprocessing.OneHotEncoder

| ID | Gender |
|---|---|
| 1 | Male |
| 2 | Female |
| 3 | Not Specified |
| 4 | Not Specified |
| 5 | Female |

| ID | Male | Female | Not Specified |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 |
| 4 | 0 | 0 | 1 |
| 5 | 0 | 1 | 0 |

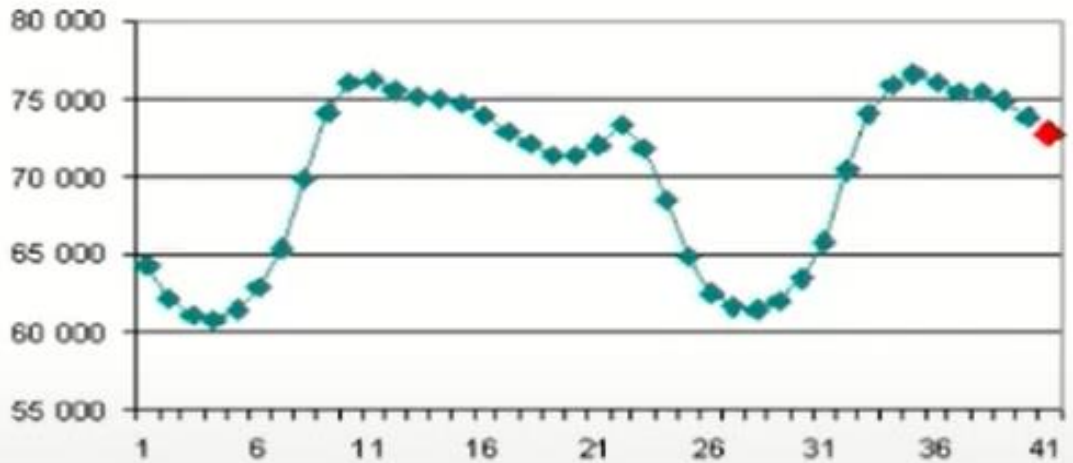
На основе словарей

❖Преобразование категориальных переменных: hashing trick
❖Преобразование дат: pandas.TimeStamp, pandas.to_datetime, т.д.
❖Преобразование строк: regular expressions, т.д.

```
>>> from sklearn.feature_extraction import FeatureHasher
>>> h = FeatureHasher(n_features=10)
>>> D = [{'dog': 1, 'cat':2, 'elephant':4},{'dog': 2, 'run': 5}]
>>> f = h.transform(D)
>>> f.toarray()
array([[ 0.,  0., -4., -1.,  0.,  0.,  0.,  0.,  0.,  2.],
       [ 0.,  0.,  0., -2., -5.,  0.,  0.,  0.,  0.,  0.]])
```

# Заполнение пропусков

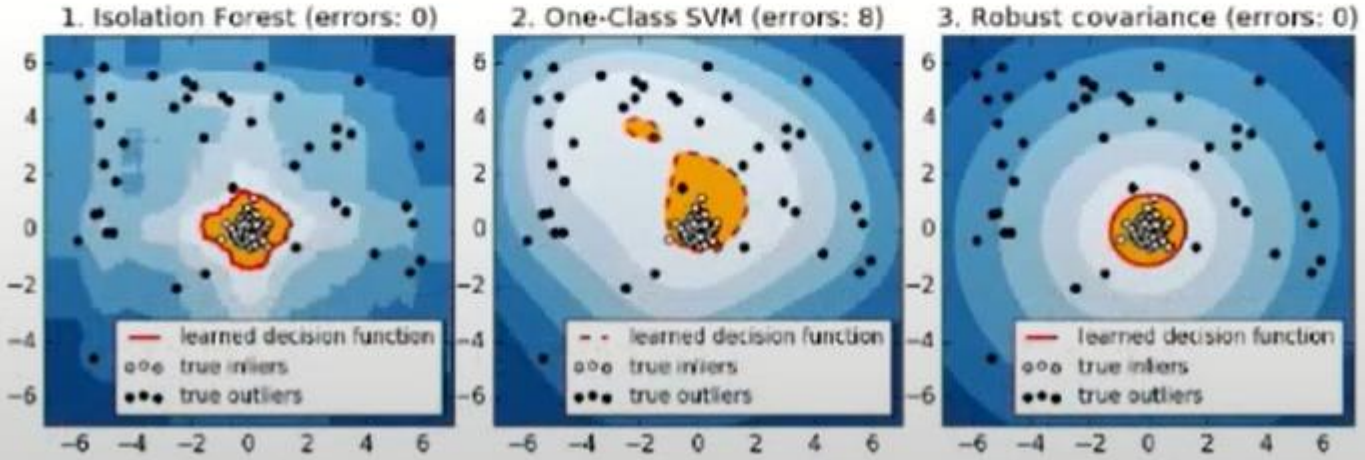
- Заполнение нулями
- Заполнение следующими, предыдущими значениями (pandas.fillna)
- Заполнение средними, модами, медианами (sklearn.preprocessing.Imputer)
- Заполнение с использованием (авто)регрессии/сезонных моделей




- Заполнение с использованием специальных адаптированных алгоритмов: MLE, kNN, EM, SVM, т.д.

# Определение выбросов

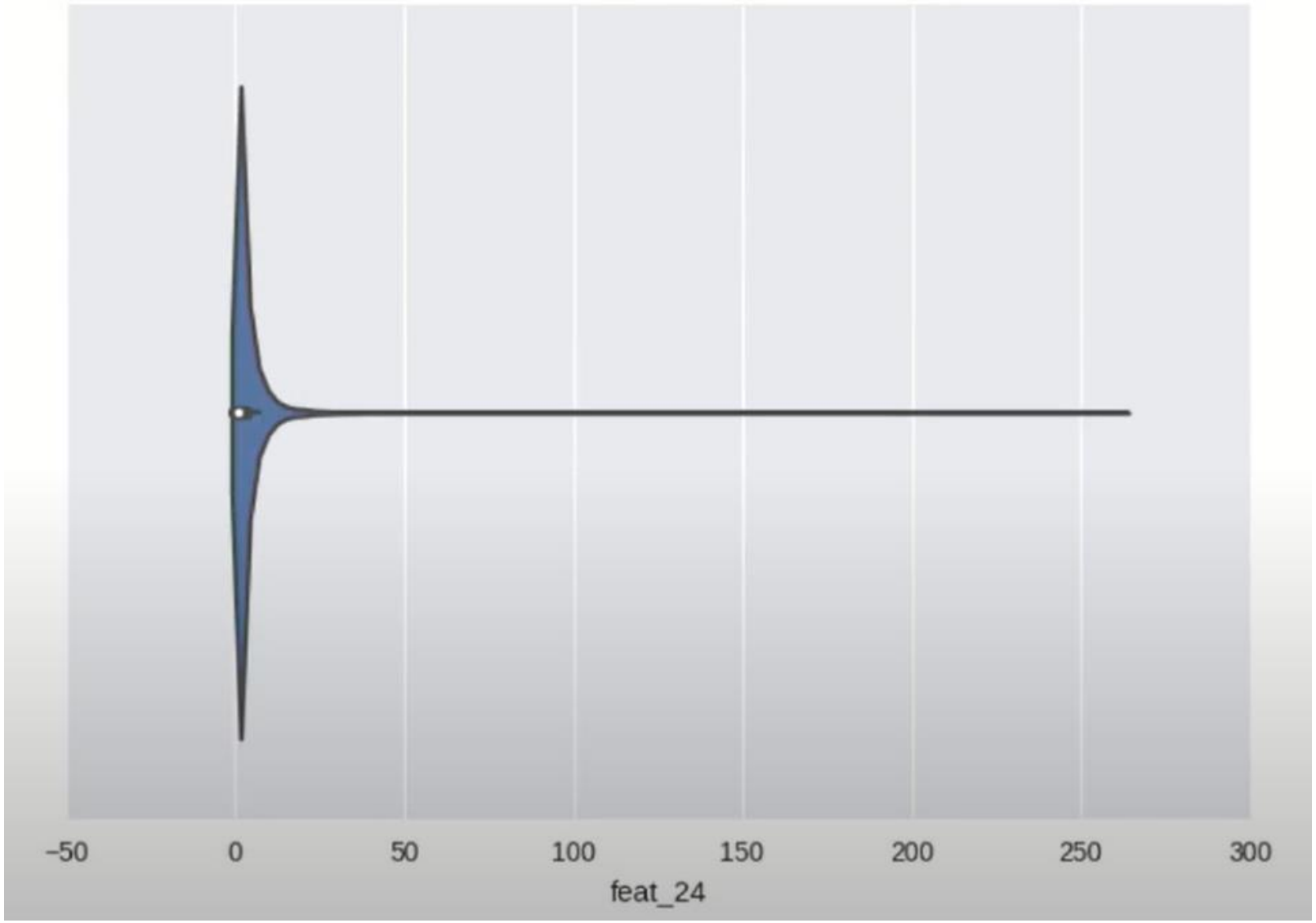
- Определение через распределения: по квантилям, перцентилям, по другим правилам пальца
- Определение через визуализацию, использую алгоритмы кластеризации (k-means, affinity propagation, ..), сокращения размерности (PCA, t-SNE, ..)
- Определение через анализ построенных моделей, постпроцессинг
- Определение через специальные адаптированные алгоритмы: OneClassSVM, EllipticEnvelope, IsolationForest




Отброс выбросов значительно улучшает результирующие метрики модели

```
sns.violinplot(train[train.columns[23]])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2a54f51978>
```




Можно предположить что все данные >30 можно отбросить, потому что их мало)

# Нормализация данных

- Стандартная нормализация
- Нормализация в 0-1 или в -1, 1(для нейроных сетей)
- Стемминг, лемматизация, TF-IDF и другие методы для текста
- Нормализация для звука
- Вычитание среднего по всем пикселям, нормализация цветов для картинок

Стандартная - приведение к нулевому мат ожиданию и единичной дисперсии. **Это рекомендовано сразу применять к числовым параметрам.**

# Несбалансированные данные

- Использовать методы балансировки данных: imbalanced-learn
  - Undersampling (SVM, kNN, NN)
  - Oversampling (SVM, kNN, NN)
- Использовать методы генерирующие данные в процессе своего обучения (NN)
- Использовать метрики для несбалансированных данных (F1-score, Matthews correlation cofficient)

## Balancing data

```
sm = SMOTE()
X_train_augmented, y_train_augmented = sm.fit_sample(X_train, y_train)
print('Original data: {0}'.format(X_train.shape))
print('Augmented data: {0}'.format(X_train_augmented.shape))
```

```
Original data: (49502, 93)
Augmented data: (60875, 93)
```

Алгоритм SMOTE() из библ **imblearn** делает оверсэмплинг, генерирует объекты меньшего класса

# Выбор значимых признаков

- Выбор через model-free методы: scikit-feature
  - Статистики (sklearn.feature_selection.SelectKBest)
  - Корреляции Пирсона, Спирмена
- Выбор через model-based методы:
  - RandomForest (rf.feature_importances_, PFI)
  - Lasso, ElasticNet (lr.coeffs_)
  - NN (DFS, HVS, PFI)
  - RFE (SVM, kNN, т.д.)

В 90% случаев основные результаты модели достигаются именно на этом этапе

Алгоритмы выбора признаков реализованы в модуле **sklearn.feature_selection**

**1. Выбор функций (Feature Selection).** Сохранение только наиболее релевантных переменных из исходного набора данных.

model-free методы

**Вот лишь некоторые примеры данных методов:**

- *Фильтр с низкой дисперсией* (VarianceThreshold)

Например, если переменные какого-либо параметра не изменяются (т.е. параметр имеет нулевую дисперсию), то этот параметр никак не влияет на модель. Поэтому, можно рассчитать дисперсию каждого параметра в выборке, а затем отбросить параметры, имеющие низкую дисперсию по сравнению с другими параметрами в наборе данных. Например, по данным титаника видно, что наименьшую дисперсию дает параметр Parch:

| | |
|---|---|
| Pclass | 0.699015 |
| Age | 211.019125 |
| SibSp | 1.216043 |
| Parch | 0.649728 |
| Fare | 2469.436846 |

| | |
|---|---|
| Age | Age in years |
| sibsp | # of siblings / spouses aboard the Titanic |
| parch | # of parents / children aboard the Titanic |
| ticket | Ticket number |
| fare | Passenger fare |

model-free методы

## 2. Фильтр высокой корреляции

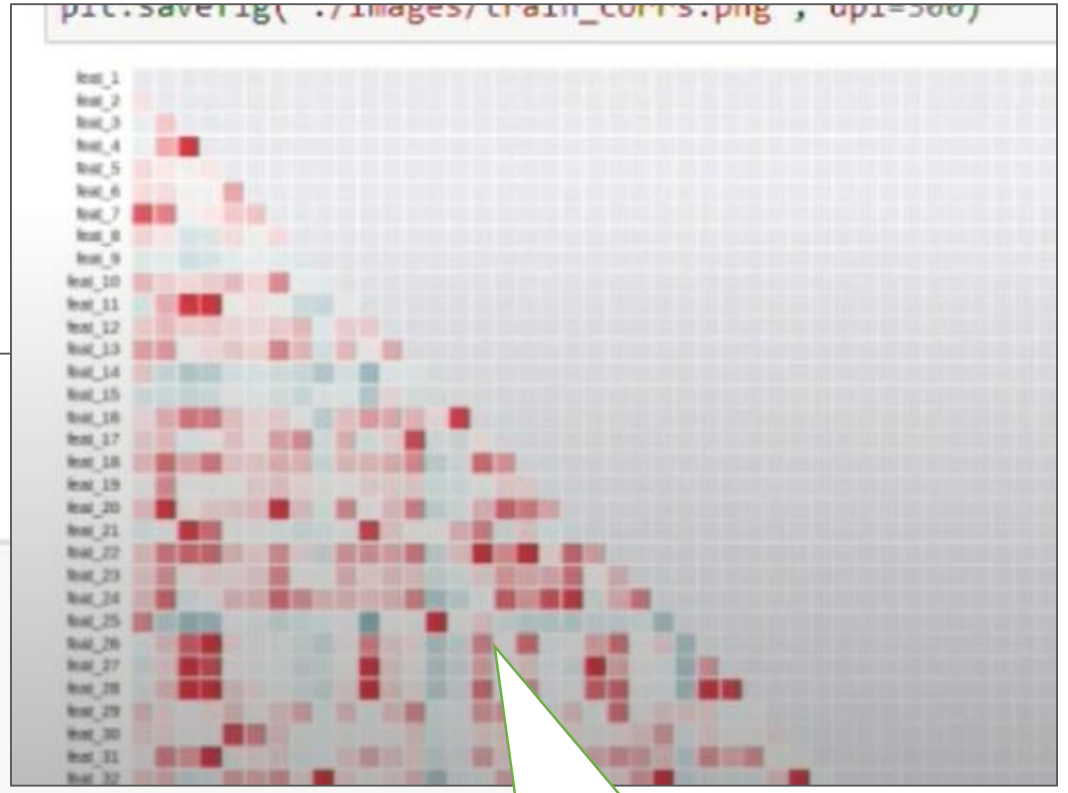
Высокая корреляция между двумя переменными означает, что они имеют схожие тенденции и, вероятно, несут схожую информацию. Это может резко снизить производительность некоторых моделей (например, моделей линейной и логистической регрессии). Можно вычислить корреляцию между независимыми числовыми переменными. *Если коэффициент корреляции пересекает определенное пороговое значение (как правило 0,5–0,6), мы можем отбросить одну из переменных (удаление переменной очень субъективно и всегда должно производиться с учетом предметной области).*

В качестве общего правила мы должны оставить те переменные, которые показывают приличную или высокую корреляцию с целевой переменной.

Если обнаружены сильные корреляции, то коррелирующие признаки можно записать через функции и сократить их количество

## Correlations of features

```
corr = train.corr()

mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

f, ax = plt.subplots(figsize=(30, 30))

cmap = sns.diverging_palette(220, 10, as_cmap=True)

sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
plt.savefig("./images/train_corrs.png", dpi=300)
```



Тепловая карта позволяет визуализировать числовые значения коэф кореляции, где темнее, там и корреляция больше

# Выбор через model-based методы

**RandomForest for feature selection**

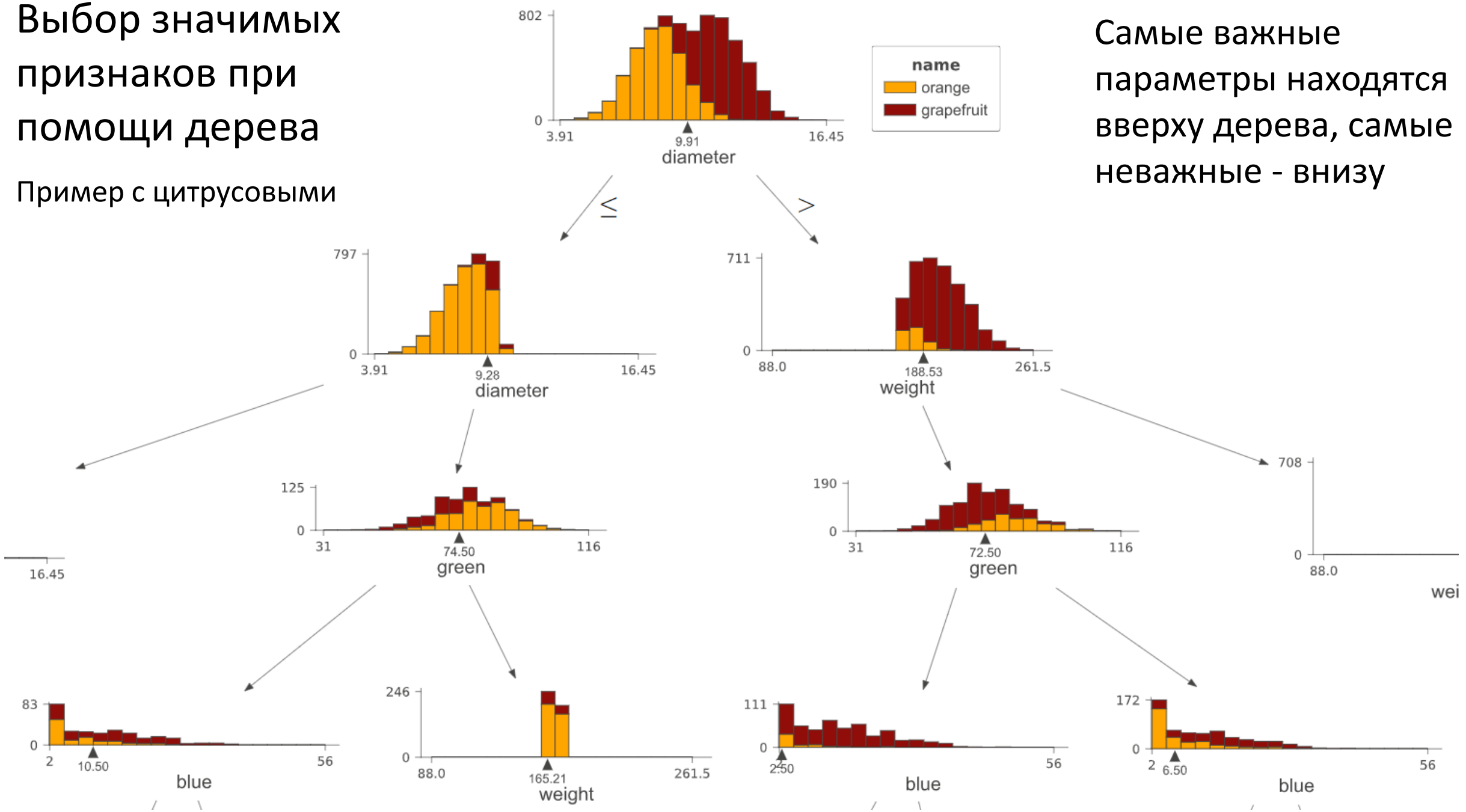
```
num_of_most_important_features = 20
forest = RandomForestClassifier(n_estimators=100)
forest.fit(X_train, y_train)

importances = forest.feature_importances
indices = np.argsort(importances)[::-1]

five_most_important_features = [X_train.columns[indices[f]] for f in range(num_of_most_important_featu
res)]
five_most_important_features.append('target')

selected_train = train.ix[:,five_most_important_features]
print(selected_train.shape) # after selection we can train new model on the selected features
```

Выделяем важные признаки при помощи случайного леса

Тренируем модель случайного леса, затем сортируем массив признаков и выбираем первые  f признаков

# Выбор значимых признаков при помощи дерева

Пример с цитрусовыми




Самые важные параметры находятся вверху дерева, самые неважные - внизу

## Рекурсивное устранение признаков

При наличии внешнего оценщика, который присваивает веса характеристикам (например, коэффициентам линейной модели), цель исключения рекурсивных признаков (*RFE*) состоит в том, чтобы выбрать признаки путем рекурсивного рассмотрения все меньших и меньших наборов признаков. Оценщик обучается на начальном наборе признаков, и определяется важность каждого признака. Затем наименее важные признаки удаляются из текущего набора признаков. Эта процедура рекурсивно повторяется для сокращенного набора, пока в конечном итоге не будет достигнуто желаемое количество признаков.

# Permutation Feature Importance

```
1.  def shuffle(array):
2.      return np.array(sorted(array, key=lambda *args: np.random.random()))
3.
4.  def pfi(model, data, target, metric, bootstrap=10):
5.      perfomance =    metric(target, model.predict(data))
6.      feature_importances = []
7.      for feature in range(num_features):
8.          feature_score = []
9.          clone = np.copy(data)
10.         for step in range(0, bootstrap):
11.             clone[:,feature] = shuffle(clone[:,feature])
12.             pred = model.predict(clone)
13.             feature_score.append(metric(target, pred))
14.         feature_importances.append(perfomance-np.mean(np.array(r2_feature_score)))
15.
16.     return feature_importances
```

Важность признаков через перестановки

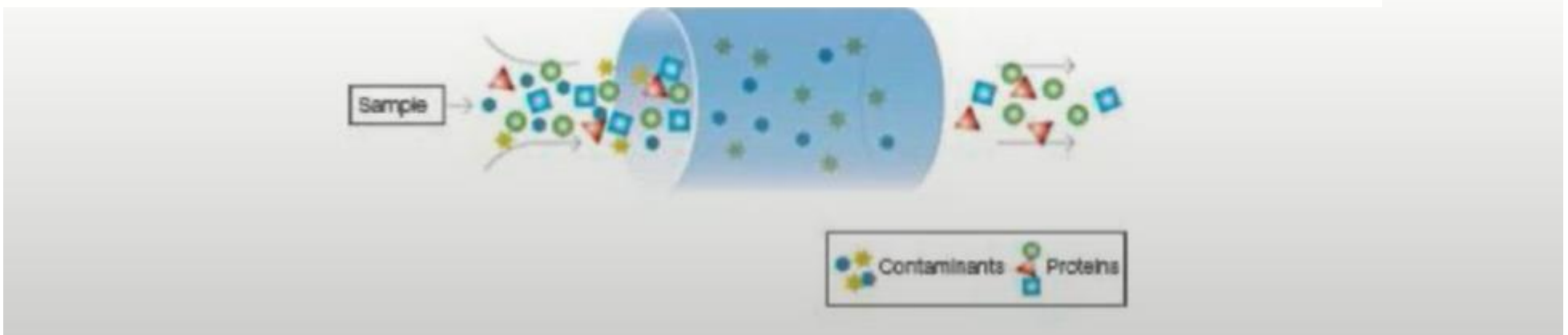
Суть
Рассчитываем метрику на начальных данных а потом на перемешенных, если точность упала то признак – значим, если не упала – то незначим

Ф-я shuffle() перемешивает массив признаков

Данный код выдает ранжированные признаки и можно по присвоенному рангу выбрать незначимые

# Экстракция признаков

- ❖Экстракция через визуальный анализ (handcrafted признаки)
- ❖Экстракция через model-based методы (NN, RandomForest, т.д.)
- ❖Экстракция через автоэнкодеры (AE) (Stacked AE, Denoising AE, т.д.)
- ❖Экстракция через методы сокращения размерности (PCA, kernel PCA, t-SNE)
- ❖Экстракция через методы кластеризации (kNN, AffinityPropagation, DBSCAN, т.д.)



# Генерация признаков Handcrafted

- Простейшие handcrafted признаки: среднее, дисперсия и т.п. по примеру
- Исследование взаимодействия признаков между собой и признаков с таргетами
- Handcrafted признаки, основанные на знании области (формулы, т.д.)
- Введение зависимостей x1*x2, x1^2, sin(x1)*x2, log1p(x1), т.д.
- Handcrafted признаки, основанные на анализе временных рядов (сезонность, т.д.)
- Handcrafted признаки, основанные на правилах (правила переходов, т.д.)
- Handcrafted признаки, основанные на пространственной зависимости примеров (ближайшие соседи, т.д.)

## PCA for feature extraction

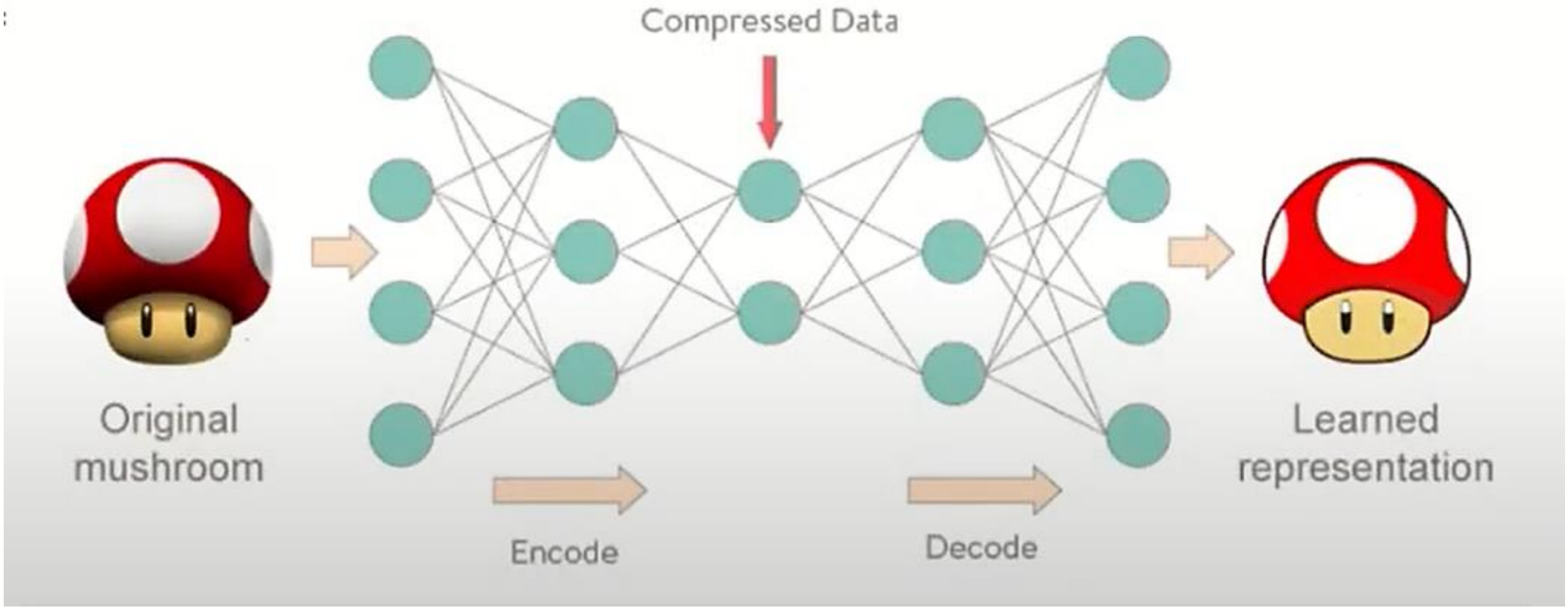
```
pca = PCA(n_components=0.95)
pca_train = pca.fit_transform(train[train.columns[:-1]])
print('{0} components preserve 95% of total variance'.format(pca.n_components_))
```

```
57 components preserve 95% of total variance
```

В примере желаемые 95 % точности при использовании метода PCA дали 57 компонент из 93-х исходного датасета

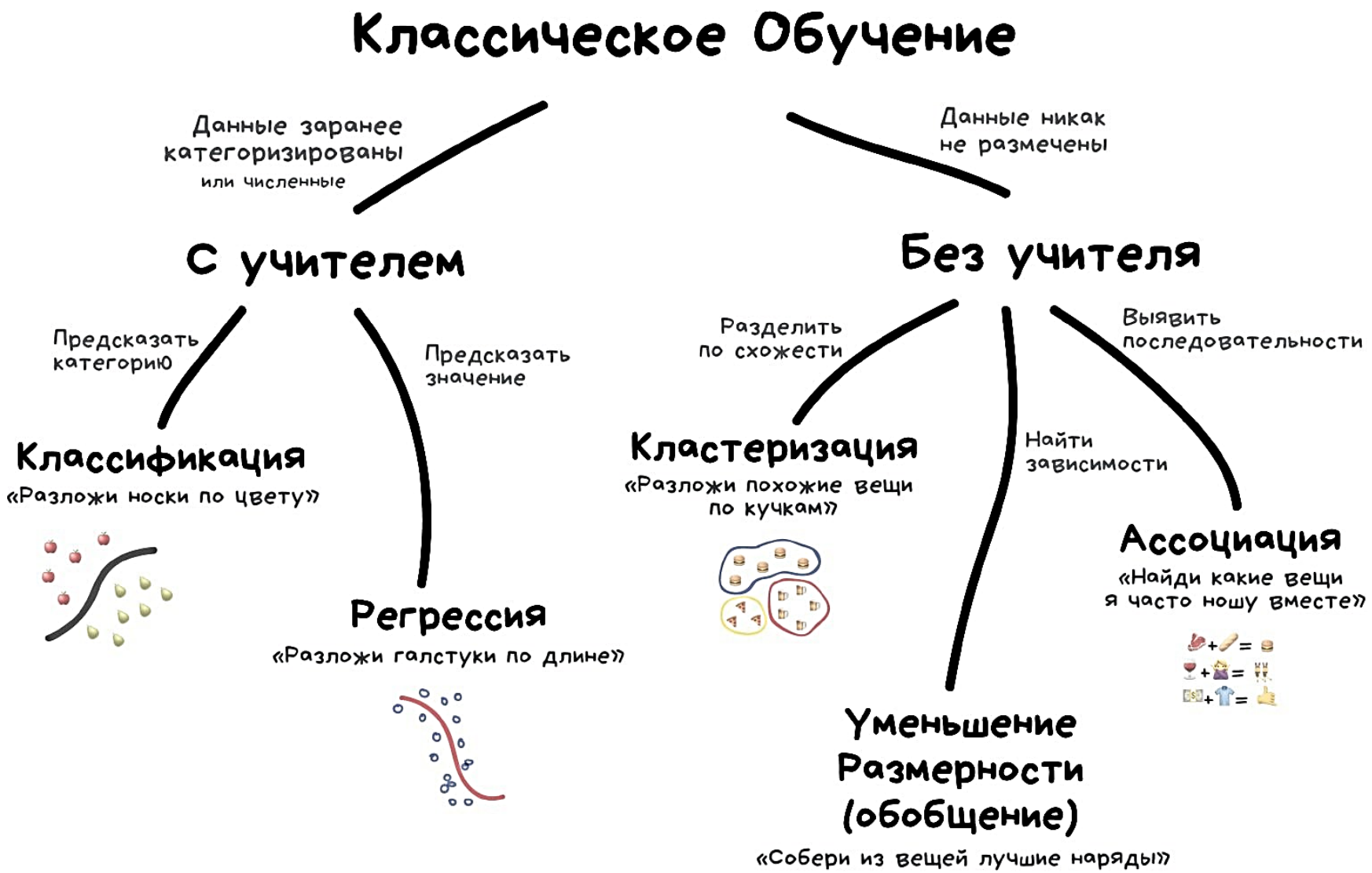
## Autoencoder for feature extraction

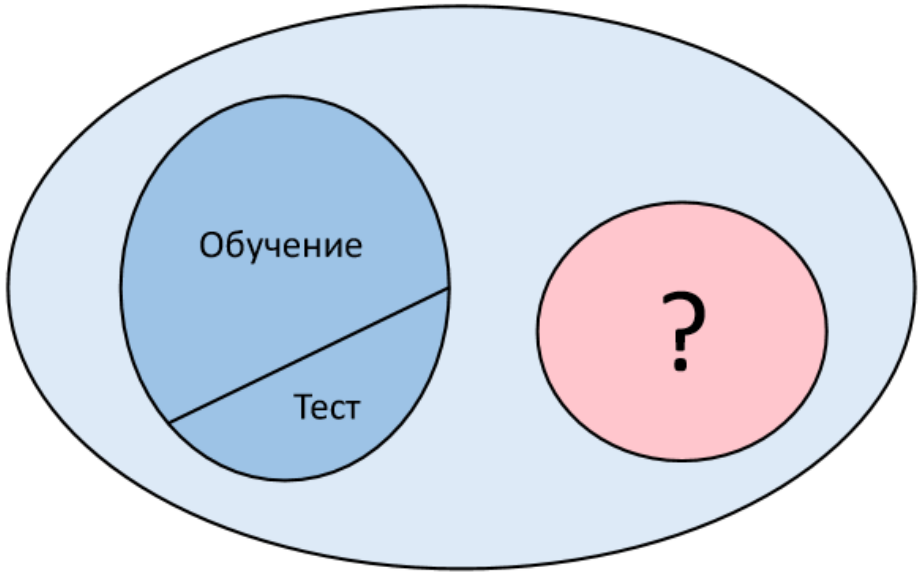
```
Image(filename="./addition/ae.png", width=800, height=800)
```




Идея обучения автоэнкодера получить на выходе то же, что подали на вход, т.е. слой с малым количеством нейронов содержит достаточную для восстановления информацию

# Выбор модели обучения






На практике все имеющиеся данные разбивают на обучающую и тестовую выборки (70% и 30%).

Обучение производится с использованием обучающей выборки, а оценка качества предсказания на основе данных тестовой выборки.

**Какие могут быть ошибки при таком разбиении** (например, выборка была отсортирована по какому либо признаку)

Оправдано, если очень много данных

Параметр **random_state** используется в качестве начального значения для генератора чисел random.
Это гарантирует, что наборы данных train и test не будут изменяться при каждом новом выполнении кода.

```
X_train, X_valid, y_train, y_valid = train_test_split(   # по умолчанию 75% и 25%
    X, y, test_size=0.3, random_state=17)
```

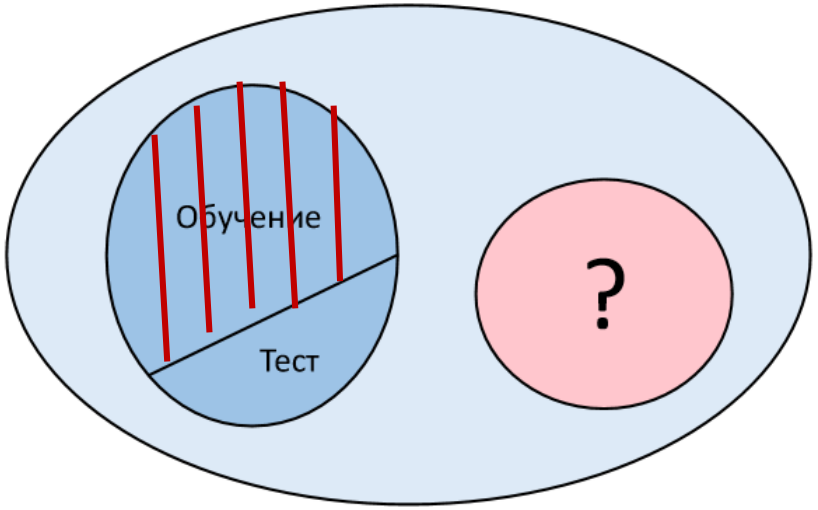
```
X_train.shape, X_valid.shape
```

```
((7000, 5), (3000, 5))
```

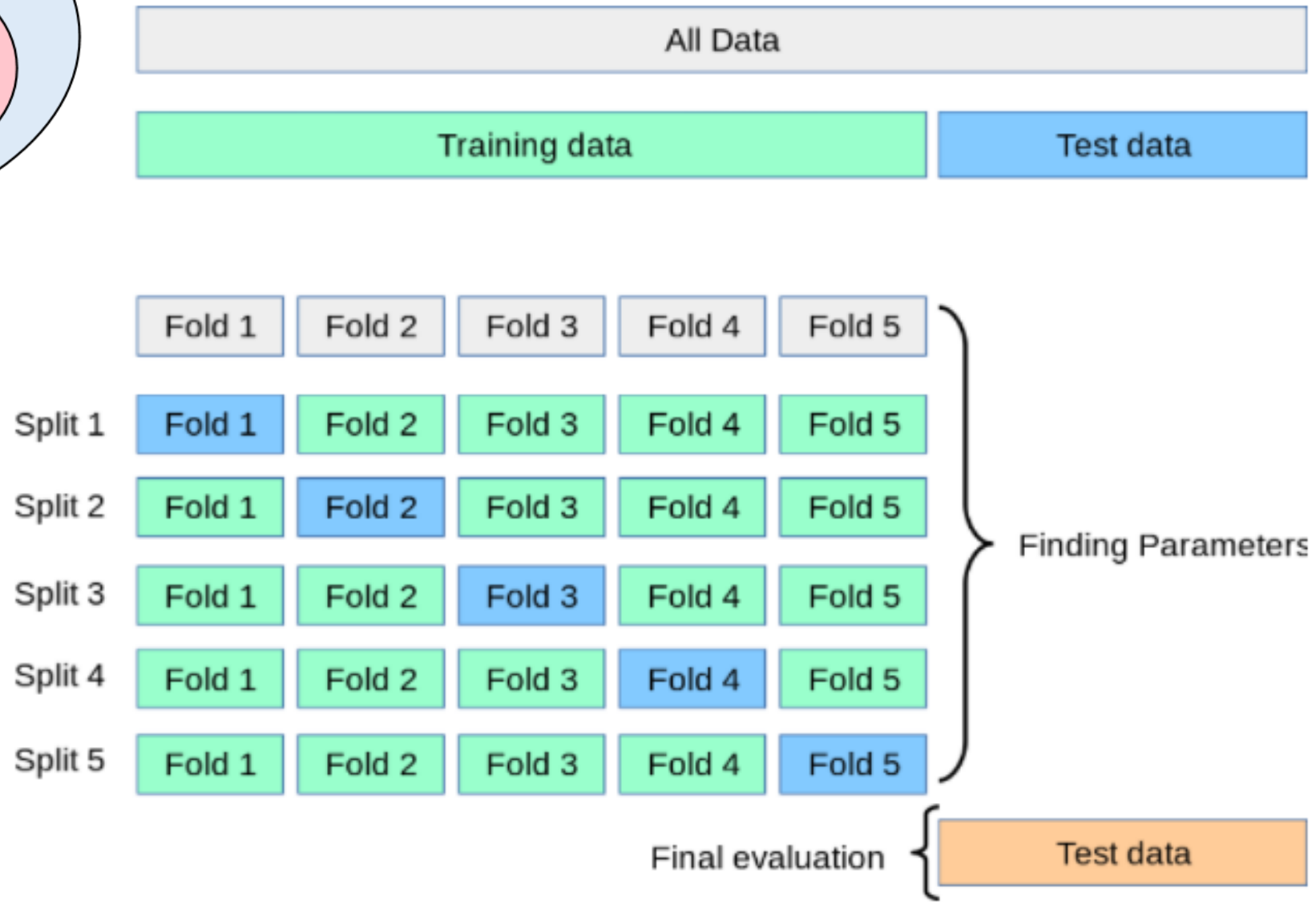
```
y_train.shape, y_valid.shape
```

```
((7000,), (3000,))
```

Данный код разбивает данные на обучающую и тестовую выборки




На практике: выборку разбивают на обучающую и тестовую, а затем к обучающей применяют метод кросс валидации:



# Оценка модели с помощью кросс-валидации

```
first_tree = DecisionTreeClassifier(random_state=17)
```

```
cross_val_score(first_tree, X_train, y_train, cv=5) # оценка модели с помощью кросс-в
```

```
array([0.93214286, 0.94357143, 0.94071429, 0.93785714, 0.94357143])
```

```
np.mean(cross_val_score(first_tree, X_train, y_train, cv=5)) #среднее по пяти оценкам
```

```
0.9395714285714286
```

# Настройка гиперпараметров

- Найти оптимальное подмножество данных на котором стоит обучаться и настраивать гиперпараметры моделей
- Использовать следующую схему GridSearch, RandomSearch:
  - На первом шаге используем попараметровый grid search для большого количества значений
  - На втором шаге для лучших значений используем обычный GridSearch, RandomSearch
- Использовать методы байесовской оптимизации:
  - baeys_opt для моделей sklearn
  - hyperopt для моделей нейронных сетей на Theano

# Настройка гиперпараметров через GridSearch

```
from sklearn.model_selection import GridSearchCV
```

```
tree_params = {"max_depth": np.arange(1, 11), "max_features": [0.5, 0.7, 1]}
```

Задаем диапазон изменения гипперпараметров для дерева max_depth и max_features

**GridSearchCV** оптимизация путем перекрестного поиска по сетке параметров:

```
tree_grid = GridSearchCV(first_tree, tree_params, cv=5, n_jobs=-1)
```

Обсчитываем все комбинации заданных гиперпараметров

```
%%time
tree_grid.fit(X_train, y_train);
```

```
Wall time: 3.14 s

GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=17), n_jobs=-1,
             param_grid={'max_depth': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
                         'max_features': [0.5, 0.7, 1]})
```

```
tree_grid.best_score_, tree_grid.best_params_
```

```
(0.9397142857142857, {'max_depth': 8, 'max_features': 0.7})
```

Вывод комбинации гиперпараметров, которые обеспечивают лучшую точность модели

# Технические Tips & Tricks

- Делать верную предобработку данных
- Правильно работать с нормализацией/выбросами/пропусками
- Проводить визуальный анализ данных, чтобы лучше понять данные, извлечь высокоуровневые признаки
- Делать сокращение размерности, выбор/экстрацию признаков model-free & model-based методами
- Строить разные модели на разных данных
- Выбрать правильные метрики
- Выбрать верную кроссвалидацию (leave-one-out vs cross validation)

- Правильно подбирать модель и ее гиперпараметры
- Подбирать оптимальный порог предсказаний
- Делать калибровку вероятностей моделей
- Правильно работать с несбалансированными данными
- Использовать аугментацию данных
- Строить ансамбли моделей, т.е. делать стекинг (Stacking)

На следующей лекции:

Оценка модели
Порог предсказаний
Калибровка вероятностей
. . . .

Через лекцию:

Ваши доклады с применением тем из 1-3 лекций