# Машинное обучение

Пензин М.С.

penzin.ml.tsu@gmail.com (mailto:penzin.ml.tsu@gmail.com)

In [55]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
import warnings
warnings.filterwarnings('ignore')
from pylab import rcParams
rcParams['figure.figsize'] = 9, 6
```

- Numpy Tutorial (https://numpy.org/doc/stable/user/quickstart.html)
- Pandas Tutorial (https://pandas.pydata.org/pandas-docs/stable/getting_started/10min.html#min)
- scikit-learn (https://scikit-learn.org/stable/tutorial/index.html)

# 1. Задачи машинного обучения

## Машинное обучение

**Машинное обучение (Machine Learning)** - это обширный подраздел искусственного интеллекта, изучающий методы построения алгоритмов, способных обучаться.

Данная дисциплина находится на стыке нескольких дисциплин, таких как математическая статистика, теория оптимизации, теории вероятности, теории графов, программирования и т.д.

## Задачи

1. Обучение с учителем:
   - Классификация
   - Регрессия
2. Обучение без учителя:
   - кластеризация
3. Обучение с подкреплением
4. И другие

## Признаки

Пусть $\mathbf{X} = \{\vec{x}_i\}_{i=1}^N$ - набор объектов, где $\vec{x}_i$ - вектор признаков (feature)

- бинарные
- категориальные
- количественные

## Обучение с учителем

При обучении с учителем имеется набор целевых исходов $Y = \{t_i\}_{i=1}^N$.

**Задача:** при заданном прецеденте получить целевой исход

## Классификация

- $\mathbf{X}$ - набор прецедентов
- $Y$ - набор меток класса

**Задача:** построить модель $f(\vec{x})$ на основе $\mathbf{X}$ и $Y$, такую что на основе вектора признаков $\vec{x}$ данная модель возвращала признак класса **y** или вектор оценок принадлежности к какому-то классу.

## Регрессия

- $\mathbf{X}$ - набор прецедентов
- $Y$ - набор значений некой величины

**Задача:** построить модель $f(\vec{x})$ на основе $\mathbf{X}$ и $Y$, такую что на основе вектора признаков $\vec{x}$ данная модель возвращала значение некой величины **y**, либо апостериорное распределение этой величины.

## Функция потерь

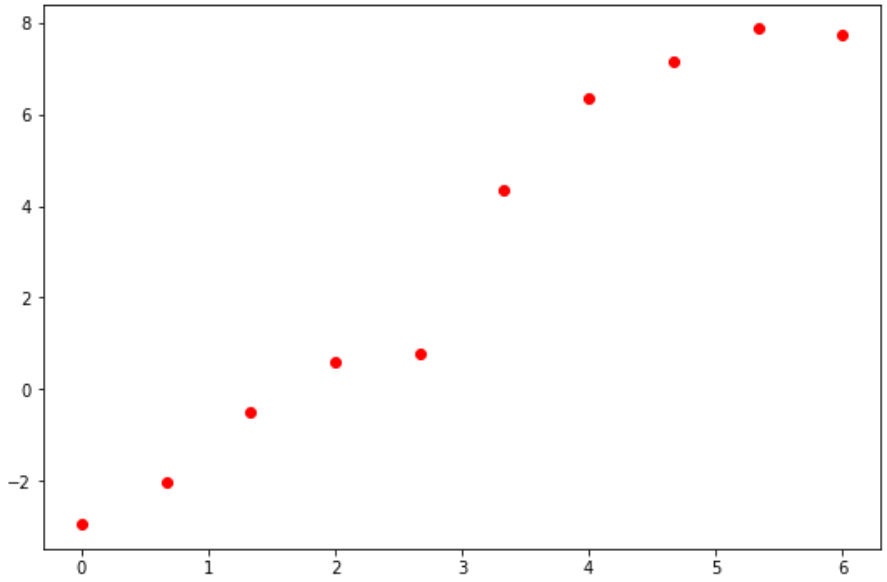
Для подавляющего числа моделей, необходимо определить способ, с помощью которого можно будет определить то, на сколько модель хорошо справляется со своей задачей. Для этого обычно вводят **функцию потерь (Loss function)**, которая используется при обучении модели

$$L(f) = \sum_{n=0}^{N-1} \frac{1}{N} \left( f(\vec{x}_n) - y_i \right)^2$$

## Переобучение

Одной из специфик машинного обучения является проблема переобучения модели.

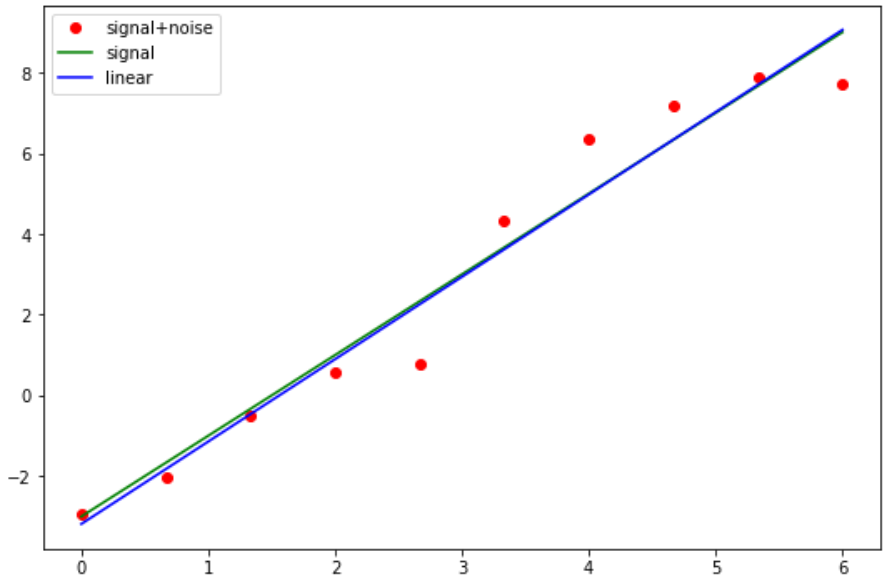
In [56]:
```
f = lambda x: 2 * x - 3
x = np.linspace(0, 6, 10)
y = f(x)
noise = np.random.normal(0, 1, y.shape)
yn = y + noise
plt.plot(x, yn, "ro", label="signal+noise")
plt.show()
```




In [57]:
```
# Фит полиномом первой степени
z = np.polyfit(x, yn, 1)
p_lin = np.poly1d(z)
z
```

Out[57]:
```
array([ 2.04050599, -3.18431024])
```

In [58]:
```
plt.plot(x, yn, "ro", label="signal+noise")
plt.plot(x, y, "g-", label="signal")
plt.plot(x, p_lin(x), "b-", label="linear")
plt.legend()
plt.show()
```

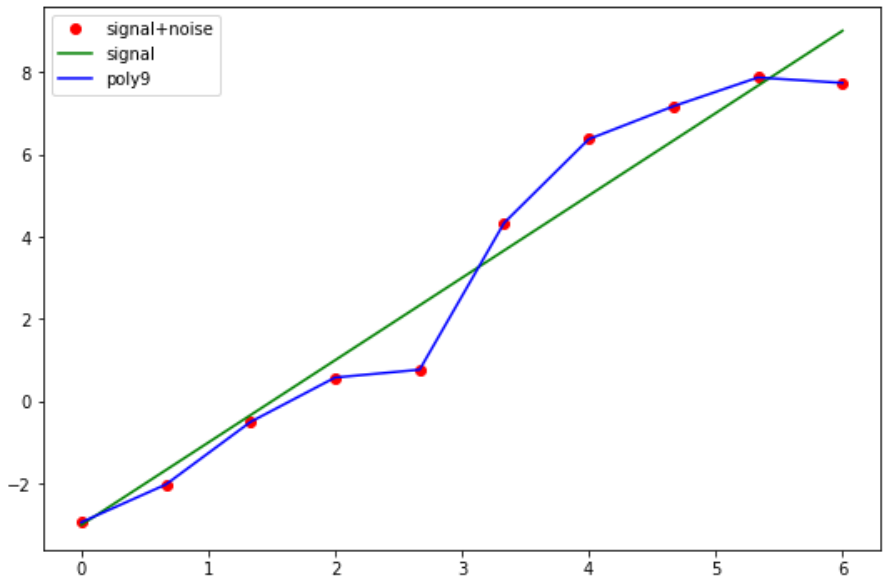



In [59]:
```
# Фит полиномом 9-й степени
z = np.polyfit(x, yn, 9)
p_9 = np.poly1d(z)
```

```
z
```

Out[59]:
```
array([ 1.70521382e-02, -4.68607332e-01,  5.40292284e+00, -3.38949291e+01,
        1.25403853e+02, -2.77012794e+02,  3.51797099e+02, -2.30751611e+02,
        6.01800537e+01, -2.93509476e+00])
```

In [60]:
```
plt.plot(x, yn, "ro", label="signal+noise")
plt.plot(x, y, "g-", label="signal")
plt.plot(x, p_9(x), "b-", label="poly9")
plt.legend()
plt.show()
```




In [61]:
```
# Посмотрим экстраполяцию данных

f(10), p_lin(10), p_9(10)
```

Out[61]:
```
(17, 17.220749661761232, 425282.49214725837)
```

## Соревнование

В рамках данного курса лекций мы будем использовать данные, которые представлены в соревновании Higgs Boson Machine Learning Challenge (https://www.kaggle.com/c/higgs-boson/overview). Эти же данные вы будете использовать в своей самостоятельной работе после этого курса лекций.

In [63]:
```
def loadTrain():
    import zipfile
    z = zipfile.ZipFile("data/training.zip")
    df = pd.read_csv(z.open("training.csv"))
    return df

df = loadTrain()
df.head()
```

Out[63]:

| | EventId | DER_mass_MMC | DER_mass_transverse_met_lep | DER_mass_vis | DER_pt_h | DER_deltaeta_jet_jet |
|---|---|---|---|---|---|---|
| **0** | 100000 | 138.470 | 51.655 | 97.827 | 27.980 | 0.91 |
| **1** | 100001 | 160.937 | 68.768 | 103.235 | 48.146 | -999.00 |
| **2** | 100002 | -999.000 | 162.172 | 125.953 | 35.635 | -999.00 |
| **3** | 100003 | 143.905 | 81.417 | 80.943 | 0.414 | -999.00 |
| **4** | 100004 | 175.864 | 16.915 | 134.805 | 16.405 | -999.00 |

5 rows × 33 columns

In [65]:
```
df.columns
```

Out[65]:
```
Index(['EventId', 'DER_mass_MMC', 'DER_mass_transverse_met_lep',
       'DER_mass_vis', 'DER_pt_h', 'DER_deltaeta_jet_jet', 'DER_mass_jet_jet',
       'DER_prodeta_jet_jet', 'DER_deltar_tau_lep', 'DER_pt_tot', 'DER_sum_pt
',
       'DER_pt_ratio_lep_tau', 'DER_met_phi_centrality',
       'DER_lep_eta_centrality', 'PRI_tau_pt', 'PRI_tau_eta', 'PRI_tau_phi',
       'PRI_lep_pt', 'PRI_lep_eta', 'PRI_lep_phi', 'PRI_met', 'PRI_met_phi',
       'PRI_met_sumet', 'PRI_jet_num', 'PRI_jet_leading_pt',
       'PRI_jet_leading_eta', 'PRI_jet_leading_phi', 'PRI_jet_subleading_pt',
       'PRI_jet_subleading_eta', 'PRI_jet_subleading_phi', 'PRI_jet_all_pt',
       'Weight', 'Label'],
      dtype='object')
```

In [66]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250000 entries, 0 to 249999
Data columns (total 33 columns):
 #   Column                      Non-Null Count   Dtype  
---  ------                      --------------   -----  
 0   EventId                     250000 non-null  int64  
 1   DER_mass_MMC                250000 non-null  float64
 2   DER_mass_transverse_met_lep 250000 non-null  float64
 3   DER_mass_vis                250000 non-null  float64
 4   DER_pt_h                    250000 non-null  float64
 5   DER_deltaeta_jet_jet        250000 non-null  float64
 6   DER_mass_jet_jet            250000 non-null  float64
 7   DER_prodeta_jet_jet         250000 non-null  float64
 8   DER_deltar_tau_lep          250000 non-null  float64
 9   DER_pt_tot                  250000 non-null  float64
 10  DER_sum_pt                  250000 non-null  float64
 11  DER_pt_ratio_lep_tau        250000 non-null  float64
 12  DER_met_phi_centrality      250000 non-null  float64
 13  DER_lep_eta_centrality      250000 non-null  float64
 14  PRI_tau_pt                  250000 non-null  float64
 15  PRI_tau_eta                 250000 non-null  float64
 16  PRI_tau_phi                 250000 non-null  float64
 17  PRI_lep_pt                  250000 non-null  float64
 18  PRI_lep_eta                 250000 non-null  float64
 19  PRI_lep_phi                 250000 non-null  float64
 20  PRI_met                     250000 non-null  float64
 21  PRI_met_phi                 250000 non-null  float64
 22  PRI_met_sumet               250000 non-null  float64
 23  PRI_jet_num                 250000 non-null  int64  
 24  PRI_jet_leading_pt          250000 non-null  float64
 25  PRI_jet_leading_eta         250000 non-null  float64
 26  PRI_jet_leading_phi         250000 non-null  float64
 27  PRI_jet_subleading_pt       250000 non-null  float64
 28  PRI_jet_subleading_eta      250000 non-null  float64
 29  PRI_jet_subleading_phi      250000 non-null  float64
 30  PRI_jet_all_pt              250000 non-null  float64
 31  Weight                      250000 non-null  float64
 32  Label                       250000 non-null  object 
dtypes: float64(30), int64(2), object(1)
memory usage: 62.9+ MB
```

# 2. Вероятностный подход

## Теорема Байеса

Пусть $A$ - это наблюдаемые классы событий: фон(b) и сигнал(s), а $B$ - это наблюдаемые данные.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

- **P(A)** - априорная вероятность наблюдения конкретного класса
- **P(B)** - полная вероятность наблюдения имеющихся данных
- **P(A|B)** - вероятность наблюдения некого класса при наблюдении конкретных данных
- **P(B|A)** - вероятность наблюдения имеющихся данных для конкретного класса

## Наивный байесовский классификатор

Таким образом, используя теорему Байеса мы сразу же можем записать

$$P(s|\vec{x}) = \frac{P(\vec{x}|s)P(s)}{P(\vec{x})}$$
$$P(b|\vec{x}) = \frac{P(\vec{x}|b)P(b)}{P(\vec{x})}$$

## Что мы имеем?

- $P(s)$, $P(b)$ - это априорная информация, которая у нас есть еще до начала наблюдения
- $P(\vec{x})$ - полная вероятность наблюдения конкретных данных
- $P(\vec{x}|s)$, $P(\vec{x}|b)$ - условные вероятности, определяемые из наблюдаемых данных

**Проблема:** Если у нас очень много признаком, то для того, чтобы построить распределения $P(\vec{x}|s)$ и $P(\vec{x}|b)$ нужно очень много данных.

Сделаем очень сильное заявление. Пусть все признаки являются **независимыми** случайными величинами. В этом случае, мы можем записать $P(\vec{x}|s)$ как

$$P(\vec{x}|s) = \prod_i P(x_i|s)$$

Подставим в теорему Байеса

$$P(s|\vec{x}) = \frac{P(s)}{P(\vec{x})} \prod_i P(x_i|s)$$

Так как $P(\vec{x})$ не зависит от выбора модели данных, то его можно не писать

$$P(s|\vec{x}) \propto P(s) \prod_i P(x_i|s)$$

Наиболее удобно работать с логарифмом этой величины

$$\ln P(s|\vec{x}) \propto \ln P(s) + \sum_i \ln P(x_i|s)$$

## Итого

Чтобы определить фон это или нет, нам достаточно посчитать два выражения

$$\ln P(s|\vec{x}) \propto \ln P(s) + \sum_i \ln P(x_i|s)$$
$$\ln P(b|\vec{x}) \propto \ln P(b) + \sum_i \ln P(x_i|b)$$

На практике, при наличие двух классов просто рассматривают отношение апостериорных вероятностей

$$\frac{P(s|\vec{x})}{P(b|\vec{x})} = \frac{P(s)\prod_i P(x_i|s)}{P(b)\prod_i P(x_i|b)}$$

Фактически, это является обычным статистическим критерием.

## Scikit-learn

Мощная библиотека, включающая в себя огромное число современных алгоритмов и инструментов для анализа данных.

```
python -m pip install sklearn
```

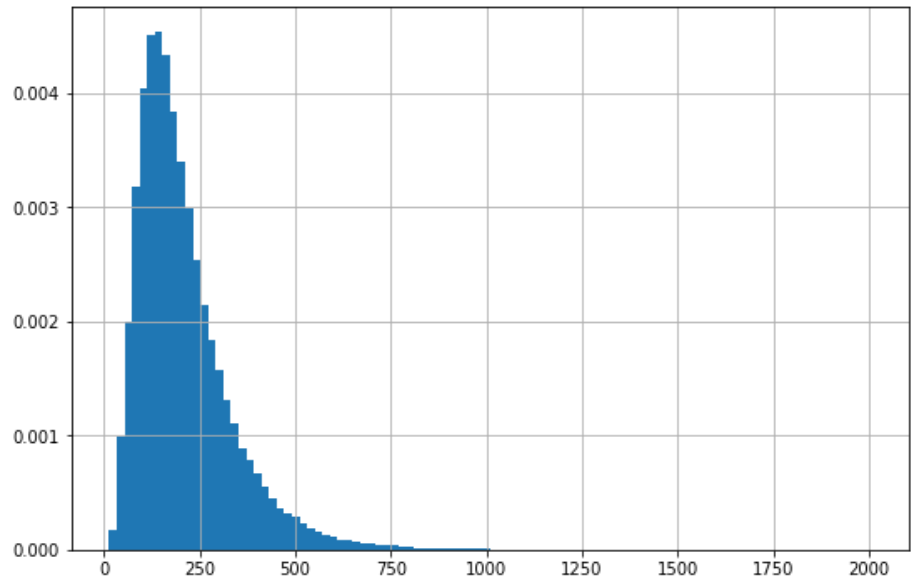
## Простая реализация

В **sklearn** уже реализована поддержка наивного байесовского классификатора, использующего нормальное распределение, распределение Бернулли и мультиномиальное распределение.

Мы воспользуемся классификатором, который предполагает, что все признаки распределены по нормальному закону

$$P(x|C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(x-\mu_k)^2}{2\sigma_k^2}}$$

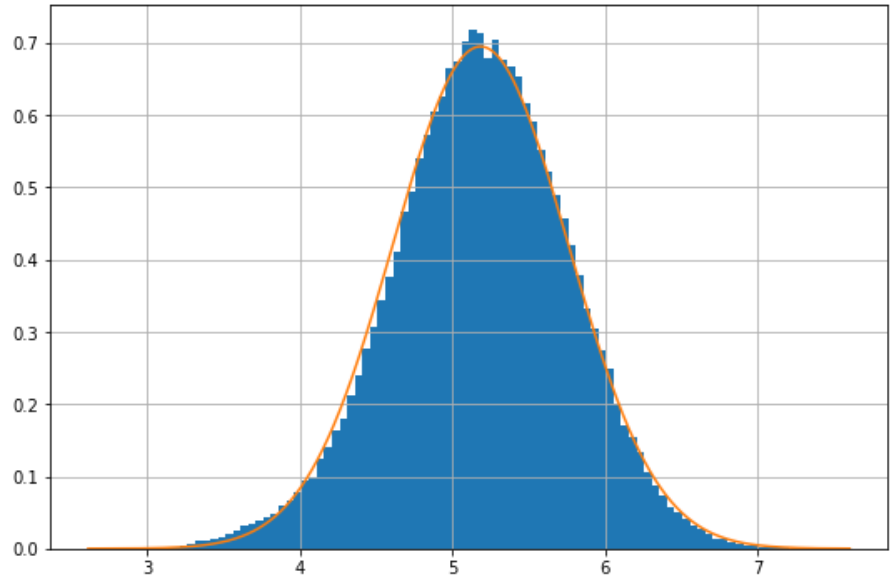
In [37]:
```
field = "PRI_met_sumet"
(df[field]).hist(bins=100, density=True);
```




Распределение не очень похоже на нормальное. Нередко его можно привести к нормальному просто прологарифмировав или использовав преобразование Бокса-Кокса/Йео-Джонсона.

In [38]:
```
field = "PRI_met_sumet"
np.log(df[field]).hist(bins=100, density=True);
from scipy.stats import norm
```

```
mu, sigma = np.log(df[field]).mean(), np.log(df[field]).std()
x = np.linspace(np.log(df[field]).min(), np.log(df[field]).max(), 1000)
plt.plot(x, norm.pdf(x, mu, sigma));
```




In [39]:
```
df[["log_PRI_met_sumet", "log_PRI_met", "log_DER_pt_ratio_lep_tau"]] = np.log(
    df[["PRI_met_sumet", "PRI_met", "DER_pt_ratio_lep_tau"]]
)

columns = ["log_PRI_met_sumet", "log_PRI_met", "log_DER_pt_ratio_lep_tau"]

df["Y"] = df["Label"].map({
    "s": 1,
    "b": 0,
})
```

In [40]:
```
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()

train = df[columns]
target = df["Y"]

model = gnb.fit(train, target)
predict = model.predict(train)
predict_proba = model.predict_proba(train)

print("Accuracy =", (target == predict).sum() / df.shape[0])
```

```
Accuracy = 0.683396
```

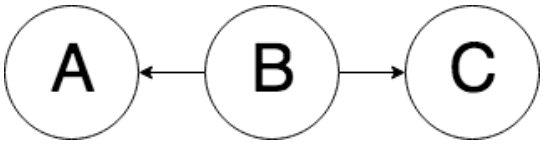
In [41]:
```
model.predict(train[1:2]), model.predict_proba(train[1:2])
```

Out[41]:
```
(array([0]), array([[0.65415632, 0.34584368]]))
```

## Факторизация

Подробнее можно почитать здесь (https://habr.com/ru/company/surfingbird/blog/176461/)

Если кратко, то мы можем простроить менее наивный классификатор.



$$P(A, B, C) = P(A|B)P(B)P(C|B)$$

## Что посмотреть дальше?

- скрытые марковские модели
- смеси распределений
- гауссовы процессы

## Что почитать?

- K.P. Murphy Machine Learning. A Probabilistic Perspective.
- C.E. Rasmussen and C.K.I. Williams. Gaussian Processes for Machine Learning

# 3. Метрики качества классификации

У нас довольно много признаков, мы можем выбрать любую их комбинацию, мы можем выбрать любые модели. Как определить какие лучше? Для сравнения вводят метрики качества классификации.

Мы уже знакомы с метрикой *accuracy_score* (доля правильных ответов), которая просто показывает процент правильных ответов. Казалось бы этого нам должно с лихвой хватить.

**Например:** Пусть у нас есть 100 хороших писем и 10 спам-писем. Наш классификатор определил правильно 90 хороших и 5 спам-писем.

$$m = \frac{90 + 5}{100 + 10} \approx 0.86$$

Но если мы просто будем говорить, что все письма не спам

$$m = \frac{100 + 0}{100 + 10} \approx 0.91$$

## Матрица неточностей

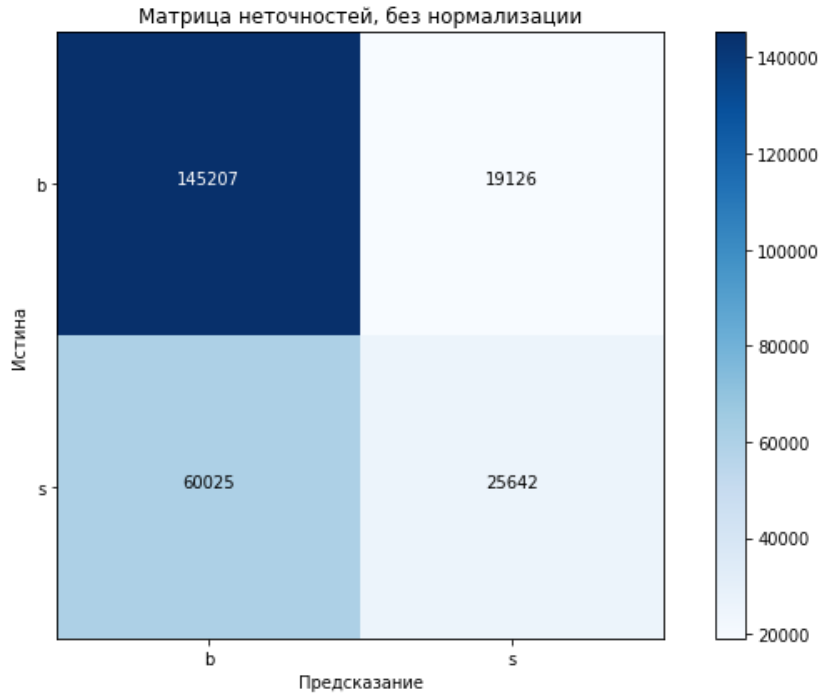
In [42]:
```
from sklearn.metrics import confusion_matrix

cnf_matrix = confusion_matrix(target, predict)
```

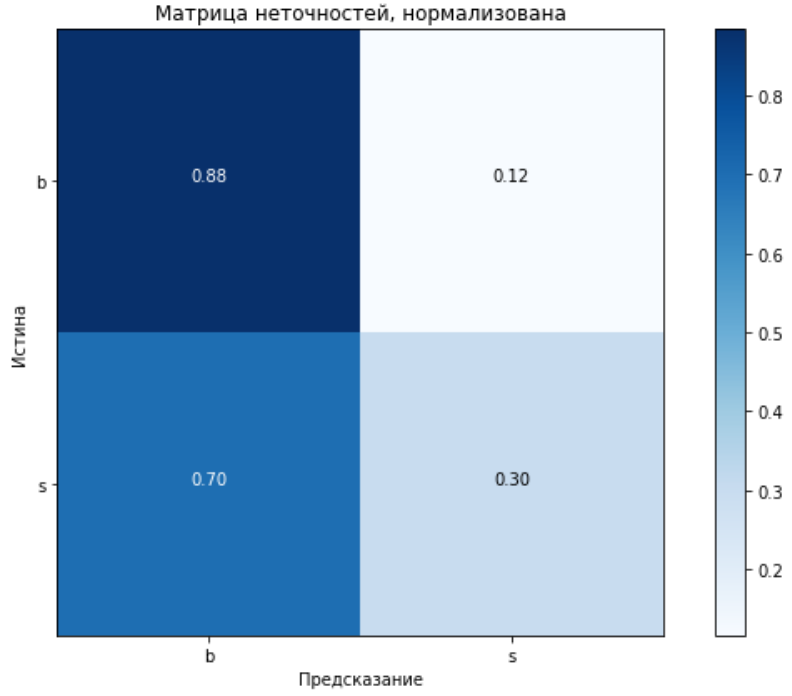
In [17]:
```
import itertools
class_names = ["b", "s"]
def plot_confusion_matrix(cm, classes, normalize=False, title='Матрица неточнос
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=0)
    plt.yticks(tick_marks, classes)
    fmt = '.2f' if normalize else 'd'
    thresh = cm.min() + (cm.max() - cm.min()) * 2 / 3.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
```

```
                 color="white" if cm[i, j] > thresh else "black")
    plt.ylabel('Истина')
    plt.xlabel('Предсказание')
    plt.tight_layout()
```

In [43]:
```
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Матрица неточностей, без нормализации')
plt.show()
```




In [44]:
```
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                      title='Матрица неточностей, нормализована')
plt.show()
```




- *True Positive(TP)* - истинно-положительный
- *False Positive(FP)* - ложно-положительный

- *True Negative(TN)* - истинно-отрицательный
- *False Negative(FN)* - ложно-отрицательный

## Accuracy

Доля правильных ответов

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

## Precision & Recall

Точность (доля правильных ответов среди ответов с этим классом)

$$precision = \frac{TP}{TP + FP}$$

Полнота (доля правильно найденных объектов данного класса)

$$recall = \frac{TP}{TP + FN}$$

Данные метрики работают в случае несбалансированности классов.

## F-мера

Есть несколько способов объединения точности и полноты в единое значение. Например, т.н. F-мера (среднее гармоническое)

$$F_\beta = (1 + \beta^2)\frac{precision \cdot recall}{\beta^2 \cdot precision + recall}$$

In [20]:
```
from sklearn.metrics import classification_report

report = classification_report(target, predict, target_names=["b", "s"])
print(report)
```

```
              precision    recall  f1-score   support

           b       0.71      0.88      0.79    164333
           s       0.57      0.30      0.39     85667

    accuracy                           0.68    250000
   macro avg       0.64      0.59      0.59    250000
weighted avg       0.66      0.68      0.65    250000
```

## ROC AUC

При определении принадлежности прецедента к какому-нибудь классу в случае бинарной классификации, мы должны выбрать порог вероятности, с которого мы будет считать, что данный прецендент относится к этому классу.

Естественным является порог 0.5, но он не всегда оптимальный. Это наиболее важно при отсутствии баланса между классами.

**ROC AUC** - это площадь (Area Under Curve) под кривой ошибок (Receiver Operating Characteristic curve).

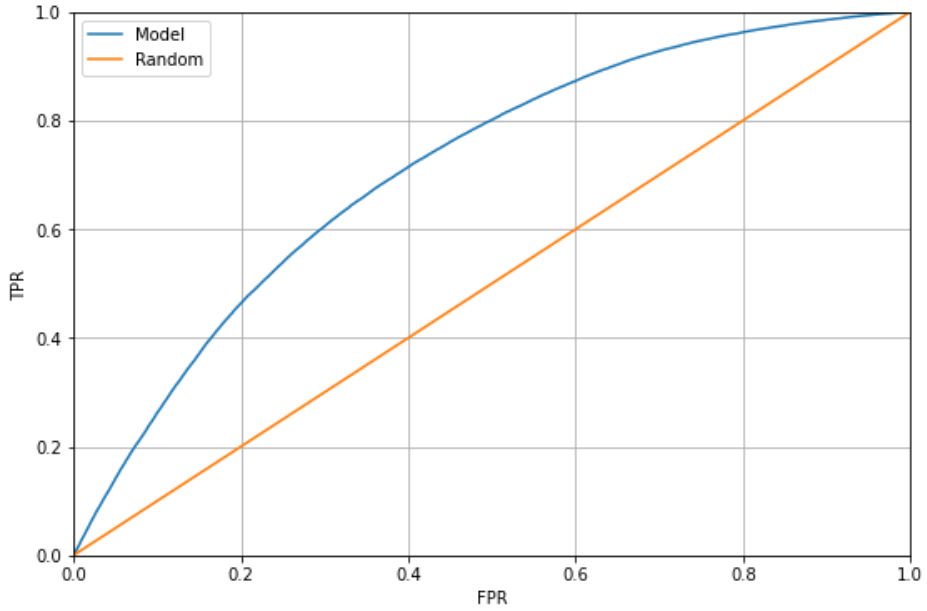
**Кривая ошибок** - это линия от (0,0) до (1,1) в координатах True Positive Rate(TPR) и False Positive

Rate(FRP) при изменении порога

$$TPR = \frac{TP}{TP + FN}$$
$$FPR = \frac{FP}{FP + TN}$$

In [45]:
```
from sklearn.metrics import roc_curve
fpr, tpr, thr = roc_curve(target, predict_proba[:,1], pos_label=1)
```

In [46]:
```
plt.xlim([0, 1]); plt.xlabel("FPR")
plt.ylim([0, 1]); plt.ylabel("TPR")
plt.plot(fpr, tpr, label="Model"); plt.plot([0, 1], [0, 1], label="Random")
plt.legend();plt.grid(); plt.show()
```



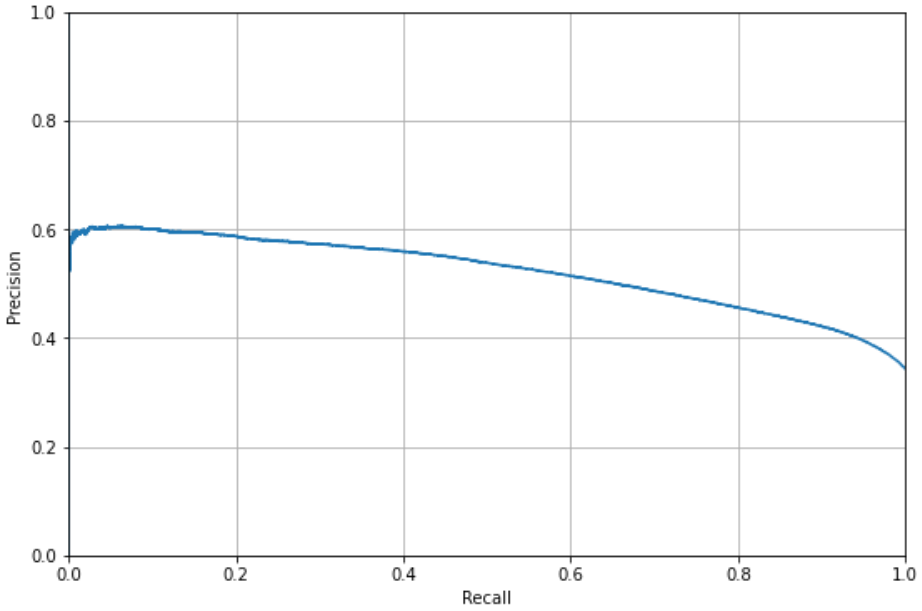

In [23]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(target, predict_proba[:,1])
```

Out[23]: 0.7119134182141041

Можно также построить кривую Precision-Recall и оценивать площадь под этой кривой (PR AUC).

In [47]:
```
from sklearn.metrics import precision_recall_curve
pr, rc, thr = precision_recall_curve(target, predict_proba[:,1], pos_label=1)
```

In [48]:
```
plt.xlim([pr.min(), pr.max()]); plt.xlabel("Recall")
plt.ylim([rc.min(), rc.max()]); plt.ylabel("Precision")
plt.plot(rc, pr); plt.grid(); plt.show()
```




## Логистическая функция потерь

одна из самых популярных метрик в соревнованиях.

$$L = -(y_i \log p_i + (1 - y_i) \log(1 - p_i))$$

где $p_i$ - это ответ классификатора для класса 0, $t_i$ - это истинное значение.

К особенности логистической функции потерь можно отнести очень сильный штраф за уверенность классификатора в правильности неправильного ответа.

## Соревнование

В представленном соревновании используется нестандартная метрика

$$AMS = \sqrt{2\left((s + b + b_r)\log\left(1 + \frac{s}{b + b_r}\right) - s\right)}$$

$s$ и $b$ - это ненормированные TPR и TFR

$$s = \sum_n w_n[y_n = s][\hat{y} = s]$$

$$b = \sum_n w_n[y_n = b][\hat{y} = s]$$

In [49]:
```
def AMS(w, y, y_pred):
    """
        Расчет метрики, работает только с Numpy-массивами
    """
    s = (w * (y == 1) * (y_pred == 1)).sum()
    b = (w * (y == 0) * (y_pred == 1)).sum()
    bReg = 10.
    return np.sqrt(2 * ((s + b + bReg) *
                        np.log(1 + s / (b + bReg)) - s))
```

In [27]:
```
from sklearn.metrics import accuracy_score

gnb = GaussianNB()

train = df[columns].to_numpy()
weight = df["Weight"].to_numpy()
target = df["Y"].to_numpy()

model = gnb.fit(train, target)
predict = model.predict(train)

print("Accuracy =", accuracy_score(target, predict))
print("AMS =", AMS(weight, target, predict))
```

```
Accuracy = 0.683396
AMS = 1.2750309370283022
```

# 4. Проблема переобучения

Ранее, мы уже видели, что увеличение сложности модели позволяет получить практически любую наперед заданную ошибку на данных, используемых для обучения.

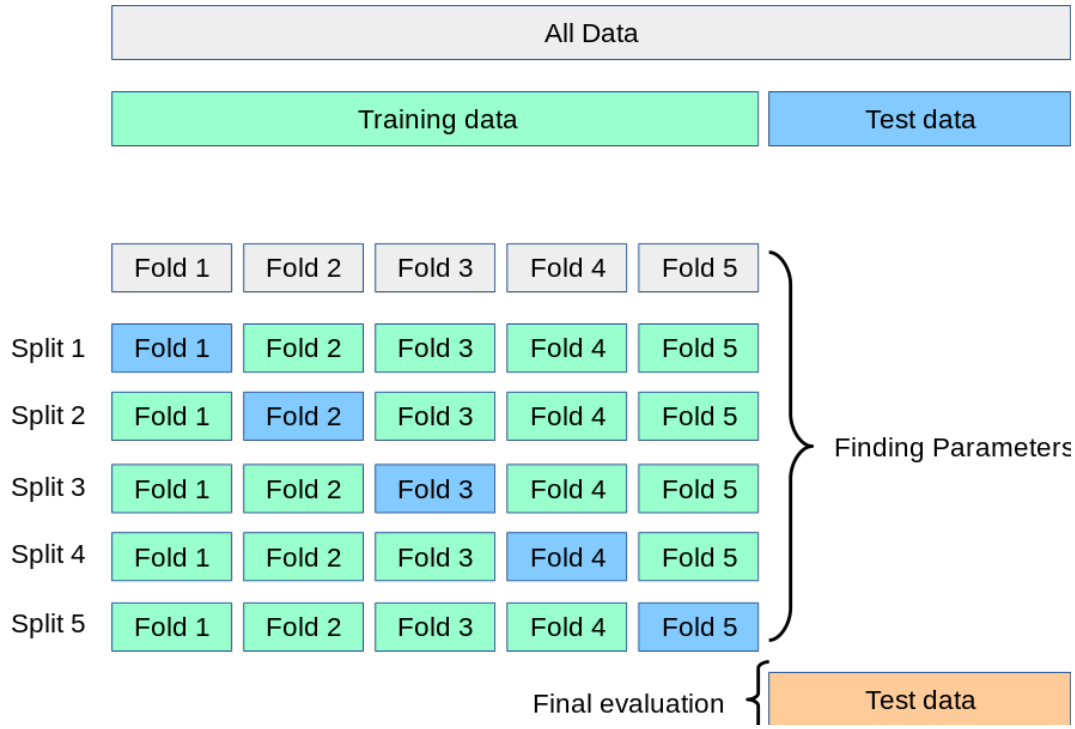
## Гиперпараметры

**Гиперпараметр модели** - некий параметр алгоритма, используемый для анализа данных. Не меняется при обучении.

## Борьба с переобучением

Для того, чтобы контролировать проблему переобучения, используют два подхода

- отложенная выборка - при этом подходе, данные разбиваются на две части: обучающая выборку и тестовую выборку. Стоит использовать тогда, когда данных очень много.
- кросс-валидация (cross-validation), наиболее популярен подход K-fold



## Пример

In [50]:
```
from sklearn.model_selection import train_test_split

# Разобьем наши данные
X_train, X_test, y_train, y_test = train_test_split(
    df[columns + ["Weight"]].to_numpy(), df["Y"].to_numpy(),
    test_size=0.3, random_state=13)
```

In [51]:
```
# Отщепляем последний столбец

w_train = X_train[:, -1]
w_test = X_test[:, -1]

X_train = X_train[:, :-1]
X_test = X_test[:, :-1]
```

In [52]:
```
gnb = GaussianNB()

model = gnb.fit(X_train, y_train)
predict = model.predict(X_test)

print("Accuracy =", accuracy_score(y_test, predict))
print("AMS =", AMS(w_test, y_test, predict))
```

```
Accuracy = 0.6832133333333333
AMS = 0.706294962879287
```

In [53]:
```
from sklearn.model_selection import cross_validate, cross_val_score

gnb = GaussianNB()

cv = cross_val_score(gnb, X_train, y_train,
                     n_jobs=2,
                     scoring='accuracy',
                     cv=3)
cv, cv.mean()
```

Out[53]:
```
(array([0.68330648, 0.68367819, 0.68342105]), 0.6834685723548334)
```