Обучение с учителем

```
from sklearn.neighbors import KNeighborsClassifier
X = churn_df[["total_day_charge", "total_eve_charge"]].values
y = churn_df["churn"].values
print(X.shape, y.shape)
knn = KNeighborsClassifier(n_neighbors=15)
knn.fit(X, y)
```

```
X_new = np.array([[56.8, 17.5], [24.4, 24.1], [50.1, 10.9]])
print(X_new.shape)
predictions = knn.predict(X_new)
print('Predictions: {}'.format(predictions))
```

```
#разедение на test и train
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=21, stratify=y)
knn = KNeighborsClassifier(n_neighbors=6)
knn.fit(X_train, y_train)
print(knn.score(X_test, y_test))
```

```
#построение графика - точность на тесте и обучении
train_accuracies = {}
test_accuracies = {}
neighbors = np.arange(1, 26)
for neighbor in neighbors:
    knn = KNeighborsClassifier(n_neighbors=neighbor)
    knn.fit(X_train, y_train)
    train_accuracies[neighbor] = knn.score(X_train, y_train)
    test_accuracies[neighbor] = knn.score(X_test, y_test)
plt.figure(figsize=(8, 6))
plt.title("KNN: Varying Number of Neighbors")
plt.plot(neighbors, train_accuracies.values(), label="Training Accuracy")
plt.plot(neighbors, test_accuracies.values(), label="Testing Accuracy")
plt.legend()  plt.xlabel("Number of Neighbors")  plt.ylabel("Accuracy")
plt.show()
```

```
#регрессия
X = diabetes_df.drop("glucose", axis=1).values
y = diabetes_df["glucose"].values
X_bmi = X[:, 3]
print(y.shape, X_bmi.shape) #(752, ) (752,)
X_bmi = X_bmi.reshape(-1, 1)
print(X_bmi.shape) #для X надо (752, 1)

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
reg_all = LinearRegression()
reg_all.fit(X_train, y_train)
y_pred = reg_all.predict(X_test)
reg_all.score(X_test, y_test) #R2 score
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred, squared=False) #RMSE
```

```
#cross-validation
from sklearn.model_selection import cross_val_score, KFold
kf = KFold(n_splits=6, shuffle=True, random_state=42)
reg = LinearRegression()
cv_results = cross_val_score(reg, X, y, cv=kf)
```

```
#Lasso регрессия
from sklearn.linear_model import Lasso,  Ridge
scores = []
for alpha in [0.01, 1.0, 10.0, 20.0, 50.0]:
    lasso = Lasso(alpha=alpha)
    lasso.fit(X_train, y_train)
    lasso_pred = lasso.predict(X_test)
    scores.append(lasso.score(X_test, y_test))
print(scores)

X = diabetes_df.drop("glucose", axis=1).values
y = diabetes_df["glucose"].values
names = diabetes_df.drop("glucose", axis=1).columns
lasso = Lasso(alpha=0.1)
lasso_coef = lasso.fit(X, y).coef_
```

```
plt.bar(names, lasso_coef)
plt.xticks(rotation=45)
plt.show() #график - значение коэффициента и название


#мера точности для классификации
from sklearn.metrics import classification_report, confusion_matrix
knn = KNeighborsClassifier(n_neighbors=7)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print(confusion_matrix(y_test, y_pred)) #вложенный список
print(classification_report(y_test, y_pred))


#логистическая регрессия
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
y_pred_probs = logreg.predict_proba(X_test)[:, 1]
print(y_pred_probs[0])


#отрисовка ROC кривой
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_probs)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Logistic Regression ROC Curve')
plt.show()

from sklearn.metrics import roc_auc_score
print(roc_auc_score(y_test, y_pred_probs)) #площадь под кривой


#GridSearchCV
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
kf = KFold(n_splits=5, shuffle=True, random_state=42)
param_grid = {"alpha": np.arange(0.0001, 1, 10), "solver": ["sag", "lsqr"]}
ridge = Ridge()
ridge_cv = GridSearchCV(ridge, param_grid, cv=kf) #RandomizedSearchCV(n_iter = 2) кол-во пар
ridge_cv.fit(X_train, y_train)
print(ridge_cv.best_params_, ridge_cv.best_score_)

test_score = ridge_cv.score(X_test, y_test) #0.7564
print(test_score)


#get_dummies
import pandas as pd
music_df = pd.read_csv('music.csv')
music_dummies = pd.get_dummies(music_df["genre"], drop_first=True)
print(music_dummies.head())
music_dummies = pd.concat([music_df, music_dummies], axis=1)
music_dummies = music_dummies.drop("genre", axis=1)

music_dummies = pd.get_dummies(music_df, drop_first=True)
print(music_dummies.columns)


#пример
from sklearn.model_selection import cross_val_score, KFold
from sklearn.linear_model import LinearRegression
X = music_dummies.drop("popularity", axis=1).values
y = music_dummies["popularity"].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
kf = KFold(n_splits=5, shuffle=True, random_state=42)
linreg = LinearRegression()
linreg_cv = cross_val_score(linreg, X_train, y_train, cv=kf, scoring="neg_mean_squared_error")
print(np.sqrt(-linreg_cv))
```

Работа с пропущенными данными

```
print(music_df.isna().sum().sort_values())
music_df = music_df.dropna(subset=["genre", "popularity", "loudness", "liveness", "tempo"])
print(music_df.isna().sum().sort_values())
```

```
#imputation (помни о разделении выборки)
from sklearn.impute import SimpleImputer
X_cat = music_df["genre"].values.reshape(-1, 1)
X_num = music_df.drop(["genre","popularity"], axis=1).values
y = music_df["popularity"].values
X_train_cat, X_test_cat, y_train, y_test = train_test_split(X_cat, y, test_size=0.2, randm_state=12)
X_train_num, X_test_num, y_train, y_test = train_test_split(X_num, y, test_size=0.2, random_state=12)
imp_cat = SimpleImputer(strategy="most_frequent")
X_train_cat = imp_cat.fit_transform(X_train_cat)
X_test_cat = imp_cat.transform(X_test_cat)

imp_num = SimpleImputer()
X_train_num = imp_num.fit_transform(X_train_num)
X_test_num = imp_num.transform(X_test_num)
X_train = np.append(X_train_num, X_train_cat, axis=1)
X_test = np.append(X_test_num, X_test_cat, axis=1)


#Imputing within a pipeline
from sklearn.pipeline import Pipeline
music_df = music_df.dropna(subset=["genre", "popularity", "loudness", "liveness", "tempo"])
music_df["genre"] = np.where(music_df["genre"] == "Rock", 1, 0)
X = music_df.drop("genre", axis=1).values
y = music_df["genre"].values

steps = [("imputation", SimpleImputer()), ("logistic_regression", LogisticRegression())]
pipeline = Pipeline(steps)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
pipeline.fit(X_train, y_train)
pipeline.score(X_test, y_test)
```

Центрирование и масштабирование

```
print(music_df[["duration_ms", "loudness", "speechiness"]].describe())
from sklearn.preprocessing import StandardScaler
X = music_df.drop("genre", axis=1).values
y = music_df["genre"].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
print(np.mean(X), np.std(X))
print(np.mean(X_train_scaled), np.std(X_train_scaled))

#scaling in pipeline
steps = [('scaler', StandardScaler()), ('knn', KNeighborsClassifier(n_neighbors=6))]
pipeline = Pipeline(steps)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=21)
knn_scaled = pipeline.fit(X_train, y_train)
y_pred = knn_scaled.predict(X_test)
print(knn_scaled.score(X_test, y_test))

#CV и scaling в pipeline
from sklearn.model_selection import GridSearchCV
steps = [('scaler', StandardScaler()), ('knn', KNeighborsClassifier())]
pipeline = Pipeline(steps)
parameters = {"knn__n_neighbors": np.arange(1, 50)}
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=21)
cv = GridSearchCV(pipeline, param_grid=parameters)
cv.fit(X_train, y_train)
y_pred = cv.predict(X_test)

print(cv.best_score_)
print(cv.best_params_)


#Evaluating classification models
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score, KFold, train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
X = music.drop("genre", axis=1).values
y = music["genre"].values
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

models = {"Logistic Regression": LogisticRegression(), "KNN": KNeighborsClassifier(), "Decision Tree": DecisionTreeClassifier()}
```

```
models = {'Logistic Regression': LogisticRegression(), 'KNN': KNeighborsClassifier(), 'Decision Tree': DecisionTreeClassifier()}
results = []
for model in models.values():
    kf = KFold(n_splits=6, random_state=42, shuffle=True)
    cv_results = cross_val_score(model, X_train_scaled, y_train, cv=kf)
    results.append(cv_results)
plt.boxplot(results, labels=models.keys()) #список списков
plt.show()

for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    test_score = model.score(X_test_scaled, y_test)
    print("{} Test Set Accuracy: {}".format(name, test_score))
```

Кластеризация

```
from sklearn.cluster import KMeans
model = KMeans(n_clusters=3)
model.fit(samples)
labels = model.predict(samples)  print(labels) #выход список 0 1 0 0 1
new_labels = model.predict(new_samples) #кластеризуя по ближайшей центройде (среднее каждого кластера)


import matplotlib.pyplot as plt
xs, ys = samples[:,0], samples[:,2]
plt.scatter(xs, ys, c=labels)
plt.show()


import pandas as pd
df = pd.DataFrame({'labels': labels, 'species': species}) # pd.crosstab любит df-формат
ct = pd.crosstab(df['labels'], df['species'])  # если известны заранее кластеры


# оценка качества кластреризации по inertial
model = KMeans(n_clusters=3)
model.fit(samples)  print(model.inertia_) #берем локоть - точка после которой инерация уменьшается меделнее


from sklearn.preprocessing import StandardScaler #как преобразование влияет на результат
scaler = StandardScaler()
scaler.fit(samples)
StandardScaler(copy=True, with_mean=True, with_std=True)
samples_scaled = scaler.transform(samples)


#засовываем в pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
scaler = StandardScaler()  kmeans = KMeans(n_clusters=3)
from sklearn.pipeline import make_pipeline
pipeline = make_pipeline(scaler, kmeans)  pipeline.fit(samples)
labels = pipeline.predict(samples)
```

Иерархическая кластеризация

```
import matplotlib.pyplot as plt #отдельно имеем признаки и лейблы
from scipy.cluster.hierarchy import linkage, dendrogram # используем scipy
mergings = linkage(samples, method='complete') #complete - расстояние между двумя кластерами является максимальным
dendrogram(mergings, labels=country_names, leaf_rotation=90, leaf_font_size=6)  plt.show()
```

Отрисовка кластеризации

```
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import linkage, dendrogram
mergings = linkage(samples, method='complete')
dendrogram(mergings,labels=country_names, leaf_rotation=90,leaf_font_size=6)
plt.show()


from scipy.cluster.hierarchy import linkage
mergings = linkage(samples, method='complete')
from scipy.cluster.hierarchy import fcluster #указываем высоту 15 и отсекаем кластеры
labels = fcluster(mergings, 15, criterion='distance') print(labels) # начало от 1, а не от 0

import pandas as pd
pairs = pd.DataFrame({'labels': labels, 'countries': country_names})
print(pairs.sort_values('labels'))
```

t-SNE - t-распределенное стохастическое соседское встраивание

```
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
model = TSNE(learning_rate=100)
transformed = model.fit_transform(samples)
xs, ys = transformed[:,0], transformed[:,1]
plt.scatter(xs, ys, c=species)  plt.show() #species - кластер точки
```

PCA

```
from sklearn.decomposition import PCA
model = PCA()
model.fit(samples)
transformed = model.transform(samples) #выход имеет ту же размернрость что и исходное признаковое пространство
#PCA называется «анализом главных компонентов», потому что он изучает «основные компоненты» данных - это направления, в которых объекты
```

```
print(model.components_) #это 2D массив, где строка - один основной компонент
```

```
import matplotlib.pyplot as plt   from sklearn.decomposition import PCA
pca = PCA()  pca.fit(samples)
features = range(pca.n_components_)
plt.bar(features, pca.explained_variance_) #график оцененной дисперсии новых признаков
plt.xticks(features) plt.ylabel('variance') plt.xlabel('PCA feature') plt.show()
```

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(samples)
transformed = pca.transform(samples)
print(transformed.shape) #выход размерность (150, 2)
```

Non-negative matrix factorization (NMF)

```
from sklearn.decomposition import NMF
model = NMF(n_components=2) # компоненты всегда указываются
model.fit(samples)

nmf_features = model.transform(samples)
print(model.components_) # 2D матрица (2 копоненты в R^n), размер 2 на 4, т.к 2 - стало, 4 - было
print(nmf_features)
```

```
# применение NMF для изображений
bitmap = sample.reshape((2, 3))
from matplotlib import pyplot as plt
plt.imshow(bitmap, cmap='gray', interpolation='nearest')
plt.show()
```

```
#Построение рекомендательной системы
from sklearn.decomposition import NMF
nmf = NMF(n_components=6)
nmf_features = nmf.fit_transform(articles)

#  оценка косинусного сходства
from sklearn.preprocessing import normalize
norm_features = normalize(nmf_features) # ко всем признакам NMF
# if has index 23
current_article = norm_features[23,:]
similarities = norm_features.dot(current_article)
print(similarities)

import pandas as pd
norm_features = normalize(nmf_features)
df = pd.DataFrame(norm_features, index=titles)
current_article = df.loc['Dog bites man']
similarities = df.dot(current_article)

print(similarities.nlargest())
```

SVM и Logistic regression

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
```