[3]

```python
#Практика 1
import pandas as pd
from sklearn.svm import SVC
data = pd.read_csv('svm-data.csv', header=None)
y = data.iloc[:, 0]
X = data.iloc[:, [1, 2]]
clf = SVC(kernel='linear', C=100000, random_state=241)
clf.fit(X, y)
support_numbers = sorted([i + 1 for i in clf.support_])
print(' '.join(map(str, support_numbers)))
```

```
4 5 10
```

[1]

```python
#Пратика 2
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.svm import SVC
import numpy as np

categories = ['alt.atheism', 'sci.space']
newsgroups = fetch_20newsgroups(subset='all', categories=categories, random_state=241)
X_text = newsgroups.data
y = newsgroups.target
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(X_text)
param_grid = {'C': [10**i for i in range(-5, 6)]}
kf = KFold(n_splits=5, shuffle=True, random_state=241)
clf = SVC(kernel='linear', random_state=241)
grid_search = GridSearchCV(clf, param_grid, cv=kf, scoring='accuracy')
grid_search.fit(X, y)
best_C = grid_search.best_params_['C']
best_svm = SVC(kernel='linear', C=best_C, random_state=241)
best_svm.fit(X, y)
coef = best_svm.coef_.toarray()[0]
top_indices = np.argsort(np.abs(coef))[::-1][:10]
feature_names = vectorizer.get_feature_names_out()
top_words = sorted([feature_names[i] for i in top_indices])
print(', '.join(top_words))
```

```
atheism, atheists, bible, god, keith, moon, nick, religion, sky, space
```

[2]

```python
#Практика 3
import pandas as pd
import numpy as np
from sklearn.metrics import roc_auc_score

def sigmoid(z):
    return 1.0 / (1.0 + np.exp(-z))
def logistic_gradient_descent(X, y, reg=0, k=0.1, eps=1e-5, max_iter=10000):
    m, n = X.shape
    X_with_bias = np.c_[np.ones(m), X]  # (m, n+1)
    w = np.zeros(n + 1)
    for i in range(max_iter):
        z = X_with_bias.dot(w)  # (m,)
        yz = y * z
        p = sigmoid(-yz)
        grad = -1.0 / m * (X_with_bias.T.dot(y * p))  # (n+1,)
        if reg > 0:
            grad[1:] += reg * w[1:]
        new_w = w - k * grad
        if np.linalg.norm(new_w - w) < eps:
            w = new_w
            break
        w = new_w
    return w
def predict_proba(X, w):
    X_with_bias = np.c_[np.ones(X.shape[0]), X]
    z = X_with_bias.dot(w)
    return sigmoid(z)

data = pd.read_csv('data-logistic.csv', header=None)
y = data.iloc[:, 0].values
X = data.iloc[:, [1, 2]].values
w1 = logistic_gradient_descent(X, y, reg=0)
w2 = logistic_gradient_descent(X, y, reg=10)
proba1 = predict_proba(X, w1)
proba2 = predict_proba(X, w2)
auc1 = roc_auc_score(y, proba1)
auc2 = roc_auc_score(y, proba2)
print(f"{auc1:.3f} {auc2:.3f}")
#Более длинные шаги: могут привести к расходимости.
#Меньшие шаги: увеличивают число итераций до сходимости.
#Изменение начального приближения: для выпуклой задачи (как логистическая регрессия) результат не должен зависеть от начального приближения, если сходится.
```

```
0.932 0.936
```

Как установить библиотеки Python? | Загрузи данные с Google Диска | Покажи пример маш

[3]

```python
#Патика 4
import pandas as pd
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    roc_auc_score, precision_recall_curve
)
data = pd.read_csv('classification.csv')
y_true = data['true'].values
y_pred = data['pred'].values
TP = ((y_true == 1) & (y_pred == 1)).sum()
FP = ((y_true == 0) & (y_pred == 1)).sum()
FN = ((y_true == 1) & (y_pred == 0)).sum()
TN = ((y_true == 0) & (y_pred == 0)).sum()
print(TP, FP, FN, TN)
acc = accuracy_score(y_true, y_pred)
prec = precision_score(y_true, y_pred)
rec = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)
print(f"{acc:.2f} {prec:.2f} {rec:.2f} {f1:.2f}")
scores = pd.read_csv('scores.csv')
y_true_scores = scores['true'].values
auc_logreg = roc_auc_score(y_true_scores, scores['score_logreg'])
auc_svm = roc_auc_score(y_true_scores, scores['score_svm'])
auc_knn = roc_auc_score(y_true_scores, scores['score_knn'])
auc_tree = roc_auc_score(y_true_scores, scores['score_tree'])
max_auc_name = max(
    ('score_logreg', auc_logreg),
    ('score_svm', auc_svm),
    ('score_knn', auc_knn),
    ('score_tree', auc_tree),
    key=lambda x: x[1]
)[0]
print(max_auc_name)
def max_precision_at_recall_70(y_true, y_scores):
    prec, rec, _ = precision_recall_curve(y_true, y_scores)
    valid = rec >= 0.7
    if not valid.any():
        return 0.0
    return prec[valid].max()
prec_at_rec_70_logreg = max_precision_at_recall_70(y_true_scores, scores['score_logreg'])
prec_at_rec_70_svm = max_precision_at_recall_70(y_true_scores, scores['score_svm'])
prec_at_rec_70_knn = max_precision_at_recall_70(y_true_scores, scores['score_knn'])
prec_at_rec_70_tree = max_precision_at_recall_70(y_true_scores, scores['score_tree'])
best_prec_at_70 = max(
    ('score_logreg', prec_at_rec_70_logreg),
    ('score_svm', prec_at_rec_70_svm),
    ('score_knn', prec_at_rec_70_knn),
    ('score_tree', prec_at_rec_70_tree),
    key=lambda x: x[1]
)[1]
print(f"{best_prec_at_70:.2f}")
```

```
43 34 59 64
0.54 0.56 0.42 0.48
score_logreg
0.65
```