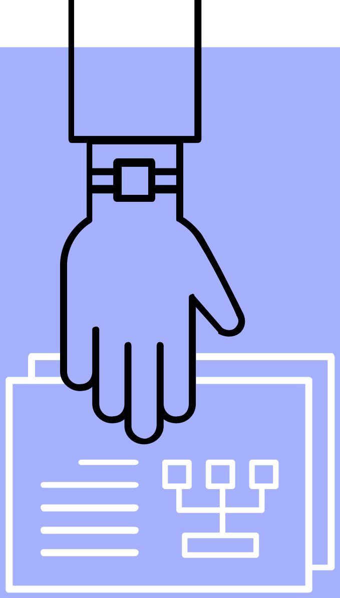
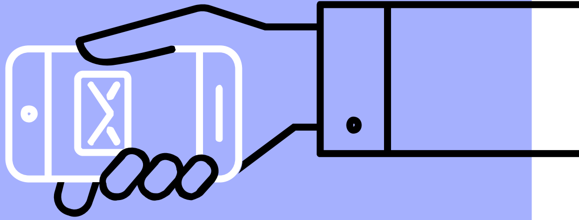
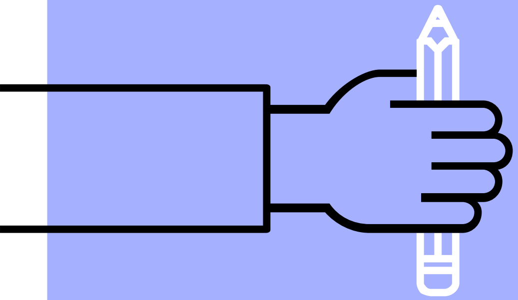
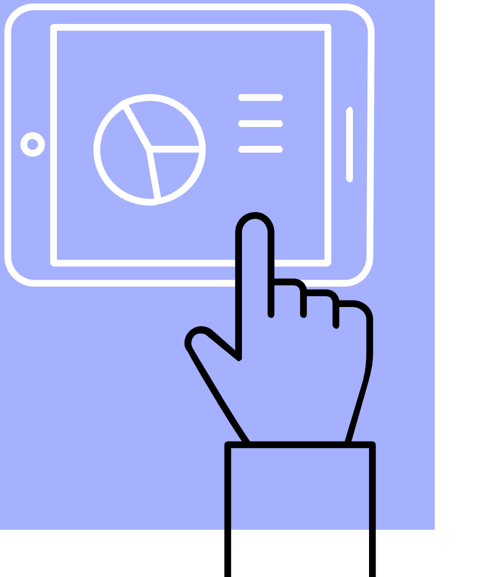




Support Vector Machine





Добрый день!

Весь используемый в проекте код:https://github.com/EvgeniaKomleva/ippi_svm

Работу выполнила Комлева Евгения для представления в Институте проблем передачи информации имени А. А. Харкевича РАН
По всем вопросам :

komleva.1999@inbox.ru

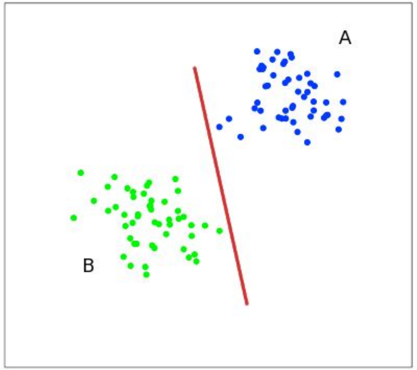
“

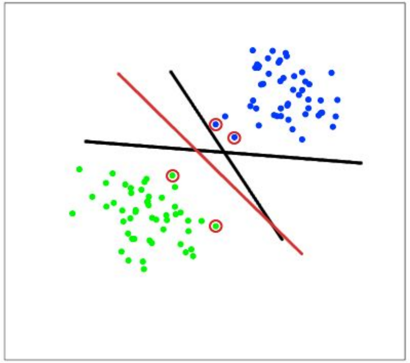
Метод опорных векторов один из самых известных методов в машинном обучении. Точно входит в топ-3

Воронцов Константин Вячеславович

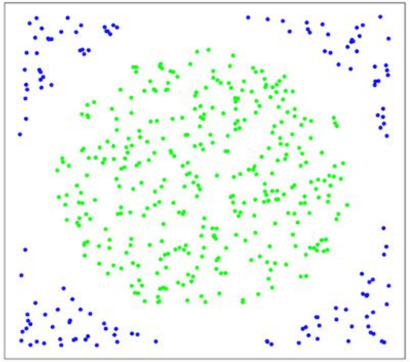
Типы данных

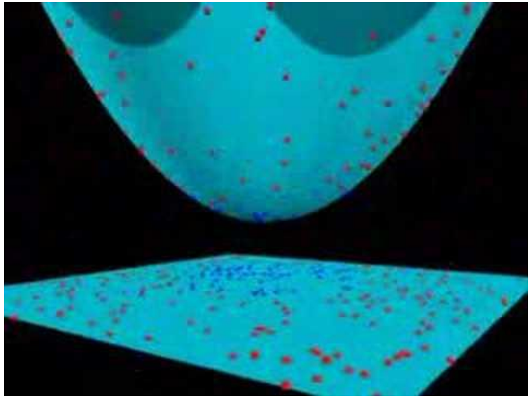
Линейно разделимые

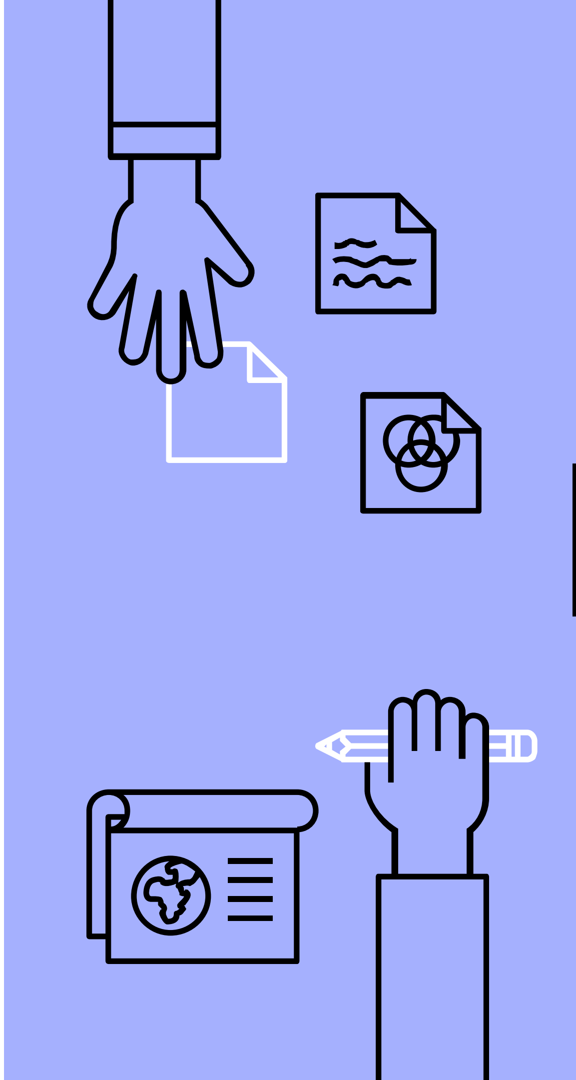


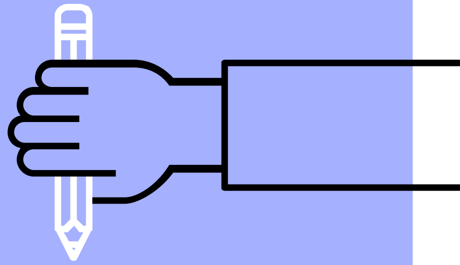


Не линейно разделимые

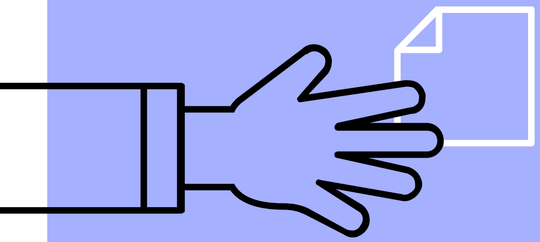








1.
Линейно разделимая выборка



Задача обучения линейного классификатора

Дано:
Обучающая выборка $X^\ell = (x_i, y_i)_{i=1}^{\ell}$,
x_i — объекты, векторы из множества $X = \mathbb{R}^n$,
y_i — метки классов, элементы множества $Y = \{-1, +1\}$.

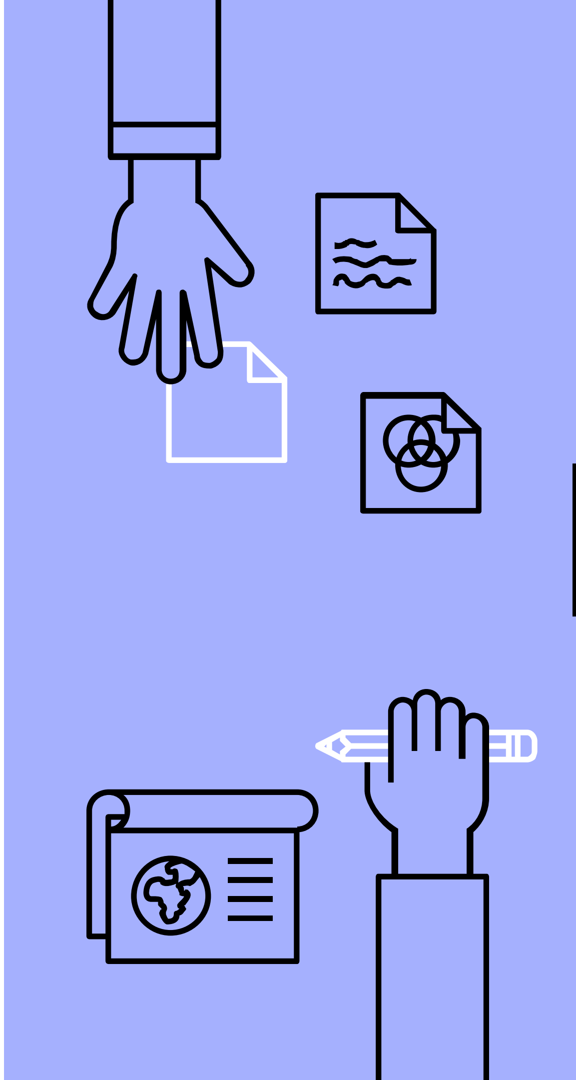
Найти:
Параметры $w \in \mathbb{R}^n$, $w_0 \in \mathbb{R}$ линейной модели классификации

$$a(x; w, w_0) = \operatorname{sign}\bigl(\langle x, w\rangle - w_0\bigr).$$

Критерий — минимизация эмпирического риска:

$$\sum_{i=1}^{\ell} \bigl[a(x_i; w, w_0) \neq y_i\bigr] = \sum_{i=1}^{\ell} \bigl[M_i(w, w_0) < 0\bigr] \to \min_{w, w_0}.$$

где $M_i(w, w_0) = \bigl(\langle x_i, w\rangle - w_0\bigr) y_i$ — *отступ* (margin) объекта x_i,
$b(x) = \langle x, w\rangle - w_0$ — *дискриминантная функция*.



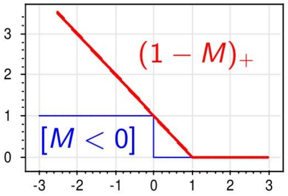
Аппроксимация и регуляризация эмпирического риска

Эмпирический риск — это кусочно-постоянная функция.
Заменим его оценкой сверху, непрерывной по параметрам:

$$Q(w, w_0) = \sum_{i=1}^{\ell} \left[M_i(w, w_0) < 0\right] \leqslant$$

$$\leqslant \sum_{i=1}^{\ell} \left(1 - M_i(w, w_0)\right)_+ + \frac{1}{2C}\|w\|^2 \;\to\; \min_{w, w_0}.$$

- *Аппроксимация* штрафует объекты за приближение к границе классов, увеличивая зазор между классами
- *Регуляризация* штрафует неустойчивые решения в случае мультиколлинеарности



Оптимальная разделяющая гиперплоскость

Линейный классификатор:

$$a(x, w) = \text{sign}\big(\langle w, x\rangle - w_0\big), \quad w, x \in \mathbb{R}^n, \; w_0 \in \mathbb{R}.$$

Пусть выборка $X^\ell = (x_i, y_i)_{i=1}^\ell$ линейно разделима:

$$\exists w, w_0 : \quad M_i(w, w_0) = y_i\big(\langle w, x_i\rangle - w_0\big) > 0, \quad i = 1, \dots, \ell$$

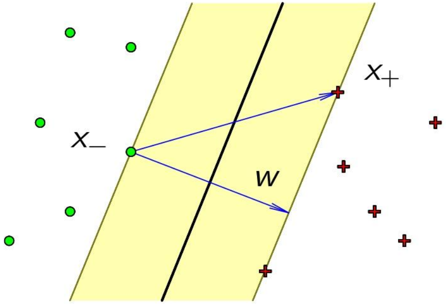
Нормировка: $\min\limits_{i=1,\dots,\ell} M_i(w, w_0) = 1.$

Разделяющая полоса:

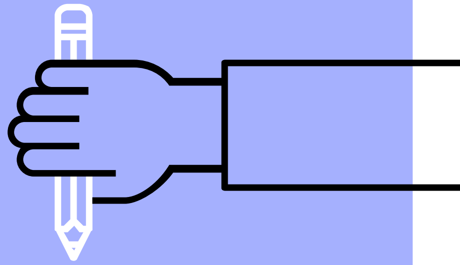
$$\big\{x\colon \; -1 \leqslant \langle w, x\rangle - w_0 \leqslant 1\big\}.$$

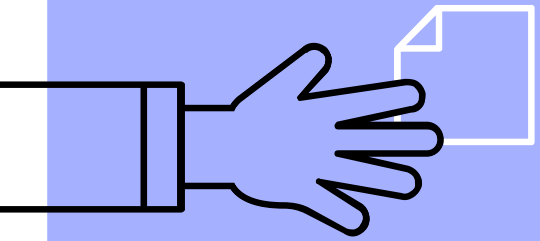
Ширина полосы:

$$\frac{\langle x_+ - x_-, w\rangle}{\|w\|} = \frac{2}{\|w\|} \to \max.$$



2.
Линейно неразделимая выборка





Переход к линейно неразделимой выборке

Постановка задачи в случае линейно разделимой выборки:

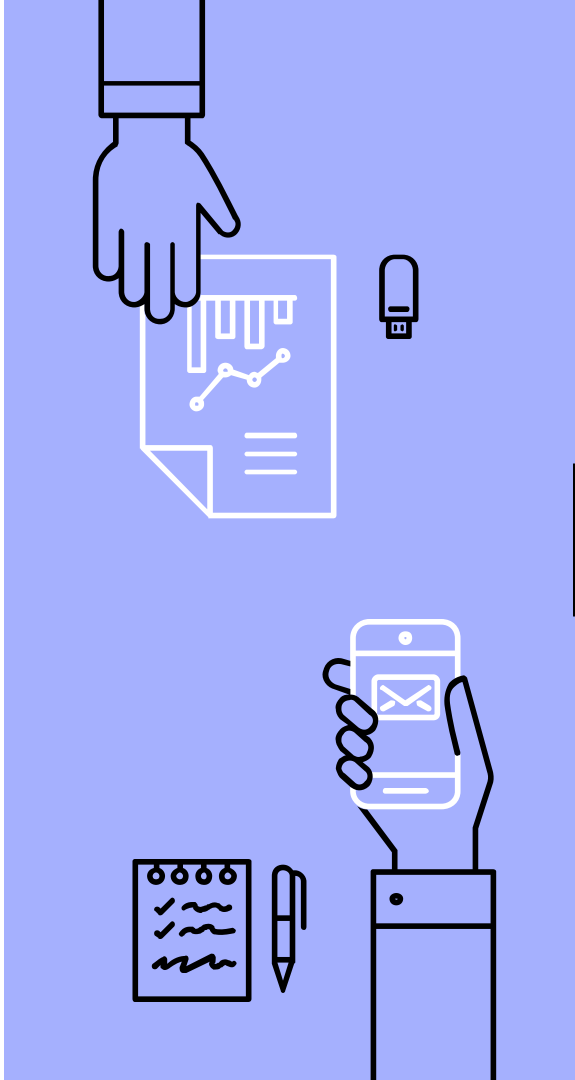
$$\begin{cases} \frac{1}{2}\|w\|^2 \to \min\limits_{w,w_0}; \\ M_i(w, w_0) \geqslant 1, \quad i = 1, \dots, \ell. \end{cases}$$

Общий случай — линейно неразделимая выборка:

$$\begin{cases} \frac{1}{2}\|w\|^2 + C \sum\limits_{i=1}^{\ell} \xi_i \to \min\limits_{w,w_0,\xi}; \\ M_i(w, w_0) \geqslant 1 - \xi_i, \quad i = 1, \dots, \ell; \\ \xi_i \geqslant 0, \quad i = 1, \dots, \ell. \end{cases}$$

Исключая ξ_i, получаем задачу безусловной минимизации:

$$C \sum_{i=1}^{\ell} \bigl(1 - M_i(w, w_0)\bigr)_+ + \frac{1}{2}\|w\|^2 \to \min_{w,w_0}.$$



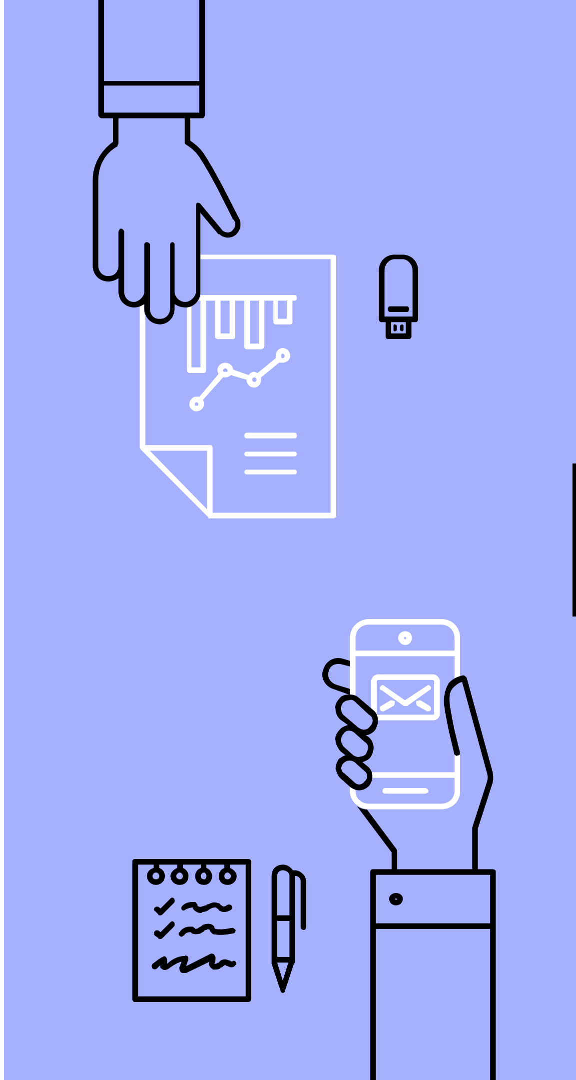
Условие Каруша-Куна-Таккера

Задача математического программирования:

$$\begin{cases} f(x) \to \min\limits_{x}; \\ g_i(x) \leqslant 0, \quad i = 1, \dots, m; \\ h_j(x) = 0, \quad j = 1, \dots, k. \end{cases}$$

Необходимые условия. Если x — точка локального минимума, то существуют множители μ_i, $i = 1, \dots, m$, λ_j, $j = 1, \dots, k$:

$$\begin{cases} \dfrac{\partial \mathscr{L}}{\partial x} = 0, \quad \mathscr{L}(x; \mu, \lambda) = f(x) + \sum\limits_{i=1}^{m} \mu_i g_i(x) + \sum\limits_{j=1}^{k} \lambda_j h_j(x); \\ g_i(x) \leqslant 0; \; h_j(x) = 0; \; (\text{исходные ограничения}) \\ \mu_i \geqslant 0; \; (\text{двойственные ограничения}) \\ \mu_i g_i(x) = 0; \; (\text{условие дополняющей нежёсткости}) \end{cases}$$



Применение условий ККТ в задаче SVM

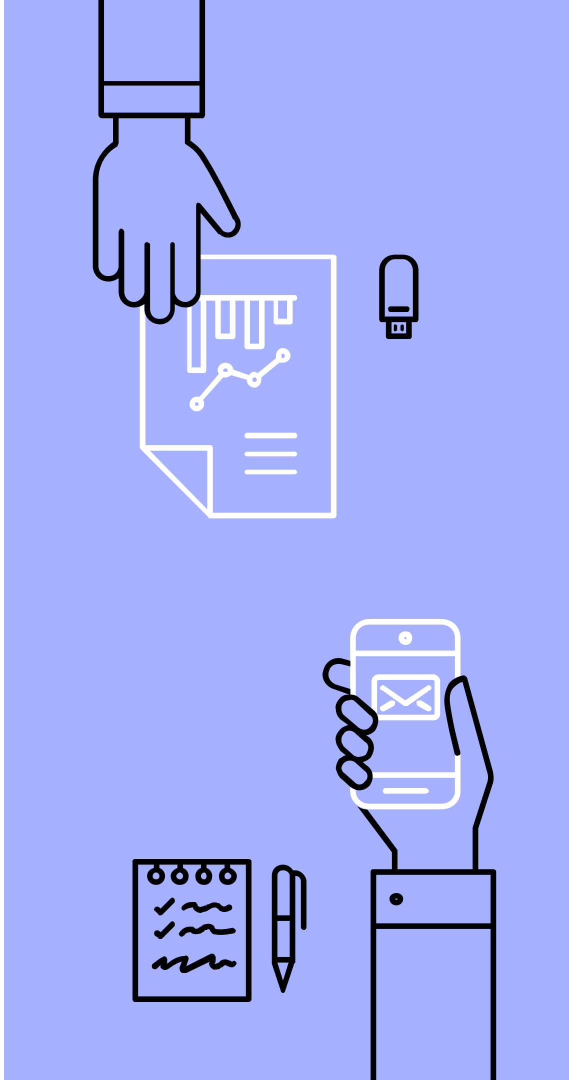
Функция Лагранжа:

$$\mathscr{L}(w, w_0, \xi; \lambda, \eta) =$$

$$= \frac{1}{2}\|w\|^2 - \sum_{i=1}^{\ell} \lambda_i \bigl(M_i(w, w_0) - 1\bigr) - \sum_{i=1}^{\ell} \xi_i \bigl(\lambda_i + \eta_i - C\bigr),$$

λ_i — переменные, двойственные к ограничениям $M_i \geqslant 1 - \xi_i$;
η_i — переменные, двойственные к ограничениям $\xi_i \geqslant 0$.

$$\begin{cases} \dfrac{\partial \mathscr{L}}{\partial w} = 0, \quad \dfrac{\partial \mathscr{L}}{\partial w_0} = 0, \quad \dfrac{\partial \mathscr{L}}{\partial \xi} = 0; \\ \xi_i \geqslant 0, \quad \lambda_i \geqslant 0, \quad \eta_i \geqslant 0, \quad i = 1, \ldots, \ell; \\ \lambda_i = 0 \ \text{либо} \ M_i(w, w_0) = 1 - \xi_i, \quad i = 1, \ldots, \ell; \\ \eta_i = 0 \ \text{либо} \ \xi_i = 0, \quad i = 1, \ldots, \ell; \end{cases}$$



Двойственная задача

$$
\begin{cases}
-\sum\limits_{i=1}^{\ell}\lambda_i + \frac{1}{2}\sum\limits_{i=1}^{\ell}\sum\limits_{j=1}^{\ell}\lambda_i\lambda_j y_i y_j \langle x_i, x_j\rangle \;\to\; \min\limits_{\lambda};\\
0 \leqslant \lambda_i \leqslant C, \quad i = 1,\dots,\ell;\\
\sum\limits_{i=1}^{\ell}\lambda_i y_i = 0.
\end{cases}
$$

Решив эту задачу численно относительно λ_i,
получаем линейный классификатор:

$$a(x) = \operatorname{sign}\Bigl(\sum_{i=1}^{\ell}\lambda_i y_i\langle x_i, x\rangle - w_0\Bigr),$$

где $w_0 = \sum\limits_{i=1}^{\ell}\lambda_i y_i\langle x_i, x_j\rangle - y_j$ для такого j, что $\lambda_j > 0$, $M_j = 1$

Определение

Объект x_i называется *опорным*, если $\lambda_i \neq 0$.

Двойственная задача нелинейное обобщение SVM

$$
\begin{cases}
-\sum\limits_{i=1}^{\ell}\lambda_i + \frac{1}{2}\sum\limits_{i=1}^{\ell}\sum\limits_{j=1}^{\ell}\lambda_i\lambda_j y_i y_j K(x_i, x_j) \;\to\; \min\limits_{\lambda};\\
0 \leqslant \lambda_i \leqslant C, \quad i = 1, \dots, \ell;\\
\sum\limits_{i=1}^{\ell}\lambda_i y_i = 0.
\end{cases}
$$

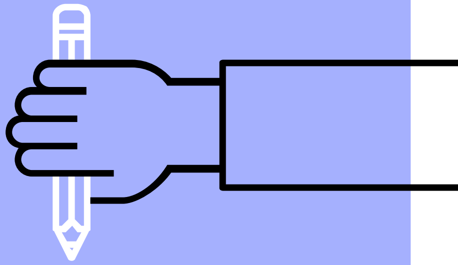
Решив эту задачу численно относительно λ_i, получаем линейный классификатор:

$$a(x) = \mathrm{sign}\Bigl(\sum_{i=1}^{\ell}\lambda_i y_i K(x_i, x) - w_0\Bigr),$$

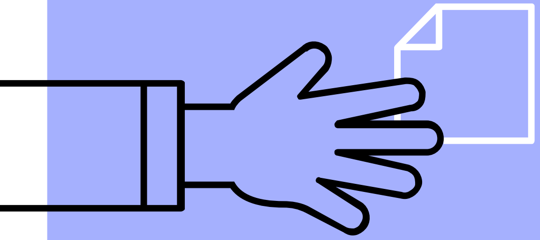
где $w_0 = \sum\limits_{i=1}^{\ell}\lambda_i y_i K(x_i, x_j) - y_j$ для такого j, что $\lambda_j > 0$, $M_j = 1$

Определение

Объект x_i называется *опорным*, если $\lambda_i \neq 0$.



3.
Ядра



Ядра для нелинейного обобщения SVM

Определение

Функция от пары объектов $K(x, x')$ называется *ядром*, если она представима в виде скалярного произведения

$$K(x, x') = \langle \psi(x), \psi(x') \rangle$$

при некотором преобразовании $\psi\colon X \to H$ из пространства признаков X в новое *спрямляющее* пространство H.

Возможная интерпретация:
признак $f_i(x) = K(x_i, x)$ — это оценка близости объекта x к опорному объекту x_i. Выбирая опорные объектов, SVM осуществляет отбор признаков в линейном классификаторе

$$a(x) = \mathrm{sign}\Bigl(\sum_{i=1}^{\ell} \lambda_i y_i K(x_i, x) - w_0 \Bigr).$$

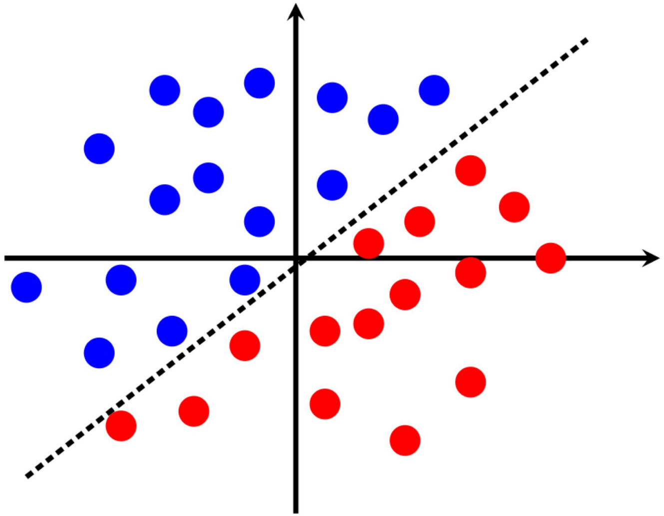
Примеры ядер

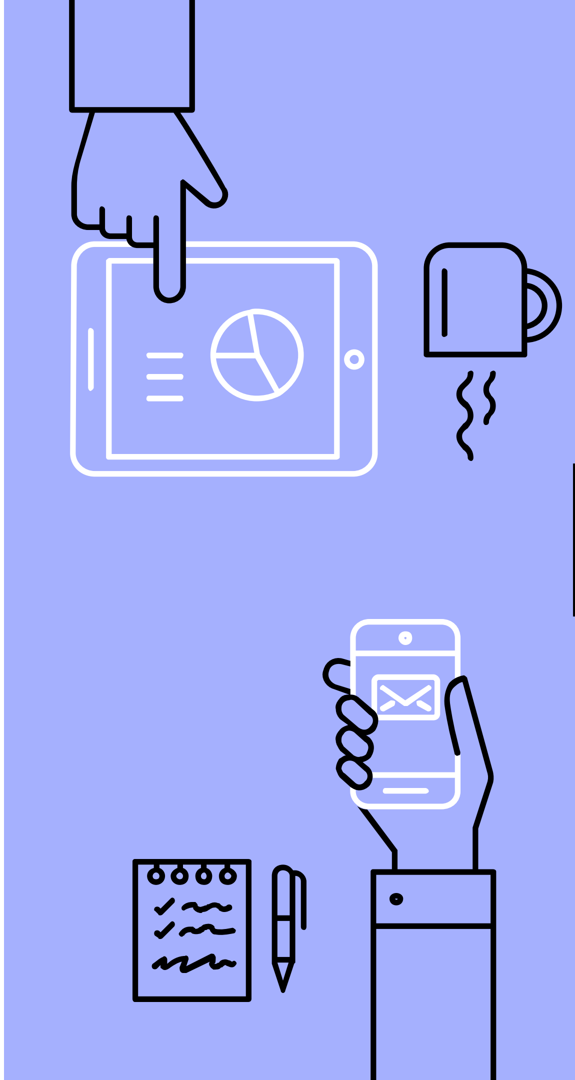
Ядра в SVM расширяют линейную модель классификации:

1. $K(x,x') = \big(\langle x,x'\rangle + 1\big)^d$
   — полиномиальная разделяющая поверхность степени $\leqslant d$;
2. $K(x,x') = \sigma\big(\langle x,x'\rangle\big)$
   — нейронная сеть с заданной функцией активации $\sigma(z)$
   (K не при всех σ является ядром);
3. $K(x,x') = \mathrm{th}\big(k_1\langle x,x'\rangle - k_0\big),\;\; k_0, k_1 \geqslant 0$
   — нейросеть с сигмоидными функциями активации;
4. $K(x,x') = \exp\big(-\gamma\|x - x'\|^2\big)$
   — сеть радиальных базисных функций (RBF ядро);

Примеры ядер

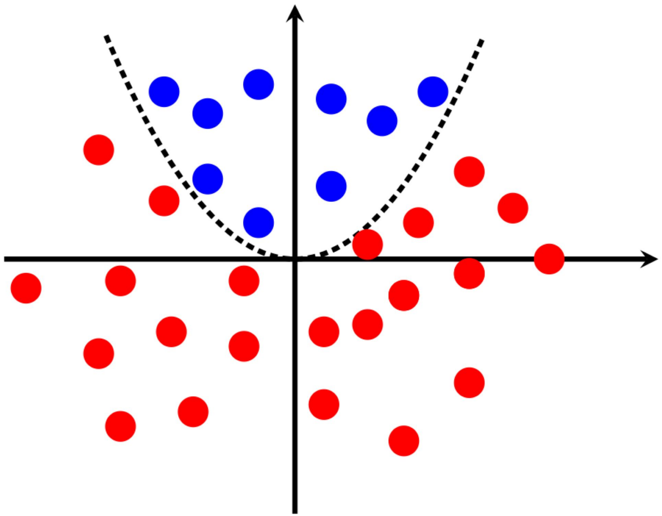
Линейное

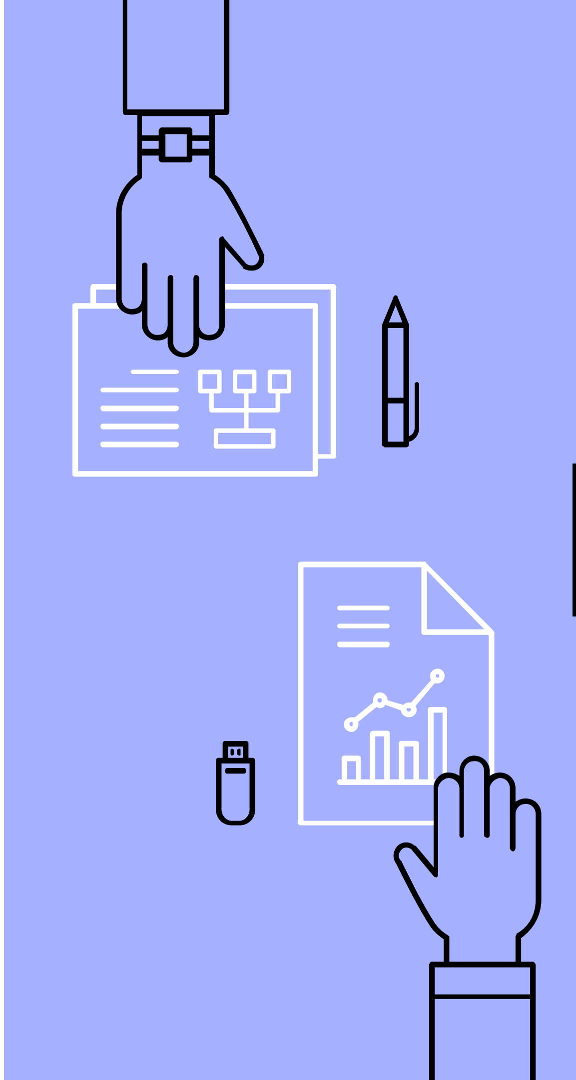




Примеры ядер

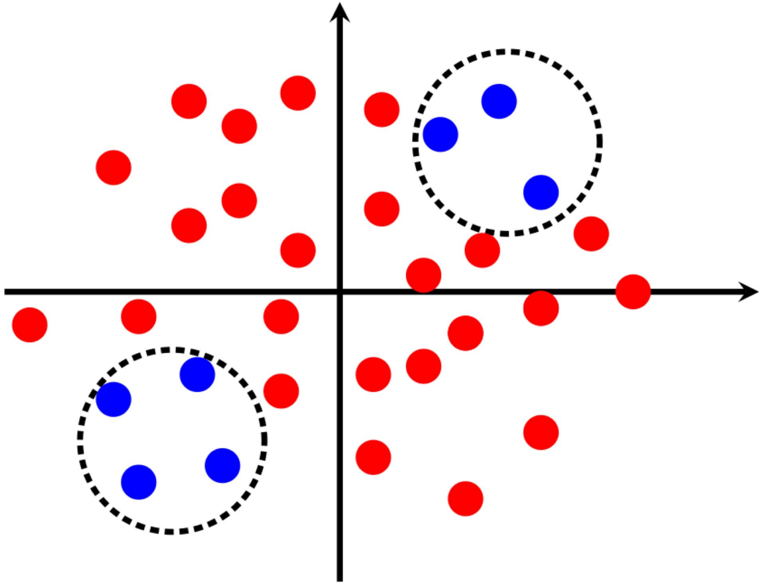
Полиномиальное

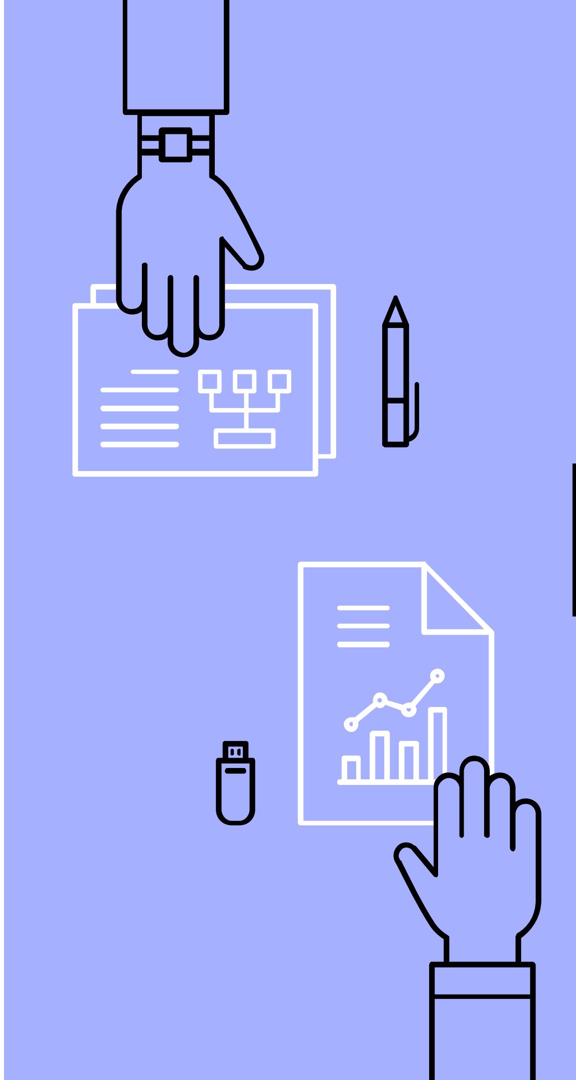




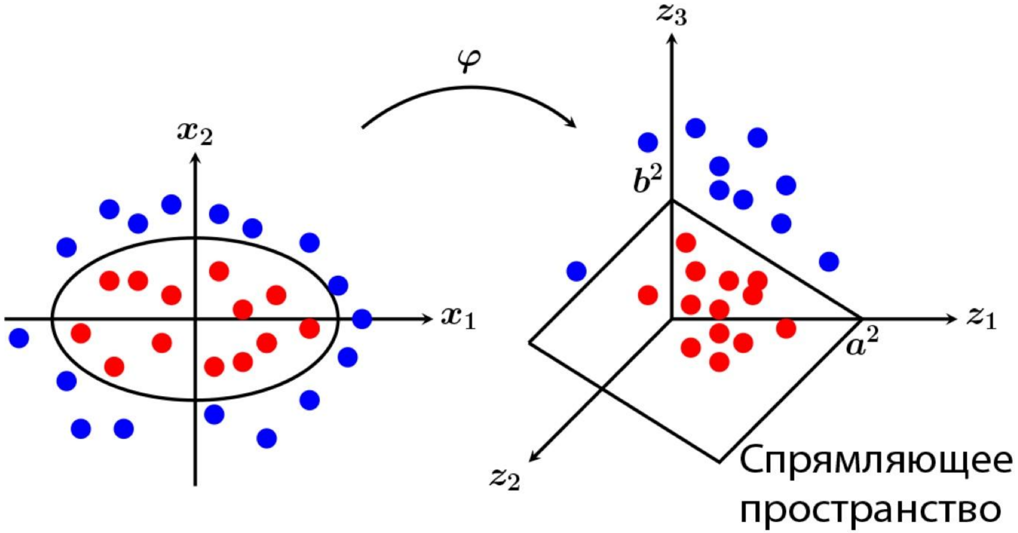
Примеры ядер

Радиальное





Пример



$$\varphi : (x_1, x_2) \to (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

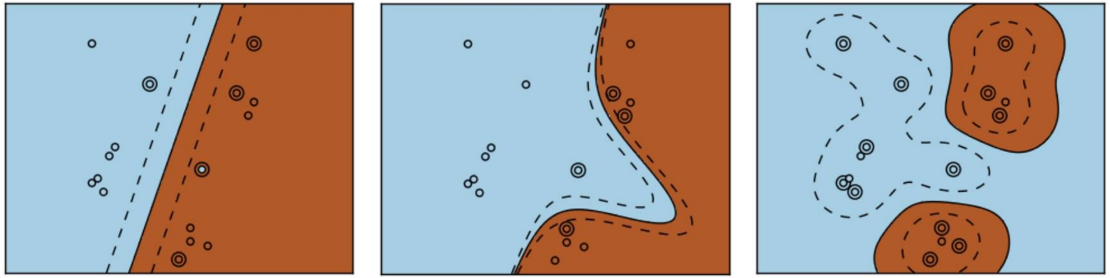
$$\left(\frac{x_1}{a}\right)^2 + \left(\frac{x_2}{b}\right)^2 = 1 \;\to\; \frac{z_1}{a^2} + \frac{z_3}{b^2} = 1$$

Классификация с различными ядрами

Гиперплоскость в спрямляющем пространстве соответствует нелинейной разделяющей поверхности в исходном.

Примеры с различными ядрами $K(x, x')$

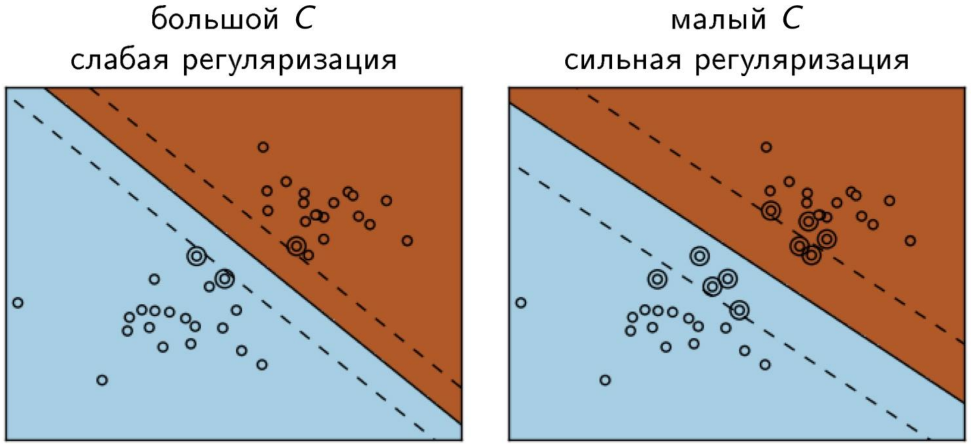
линейное	полиномиальное	гауссовское (RBF)
$\langle x, x' \rangle$	$(\langle x, x' \rangle + 1)^d, \quad d{=}3$	$\exp(-\gamma \lVert x - x' \rVert^2)$



Влияние C на решение SVM

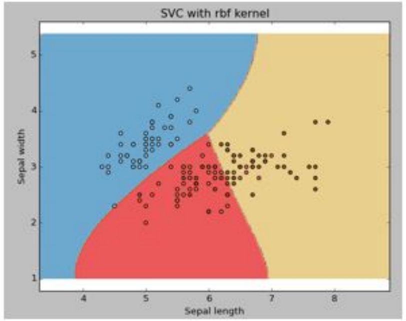
SVM — аппроксимация и регуляризация эмпирического риска:

$$\sum_{i=1}^{\ell} \left(1 - M_i(w, w_0)\right)_+ + \frac{1}{2C}\|w\|^2 \to \min_{w, w_0}.$$

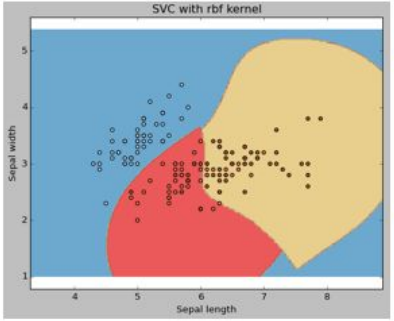


Подбор параметров SVM

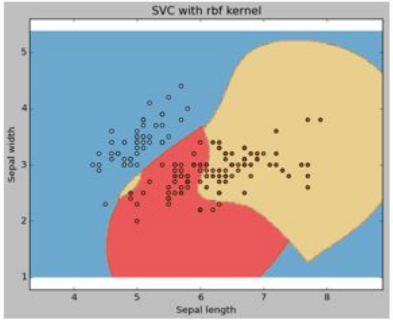
c =1



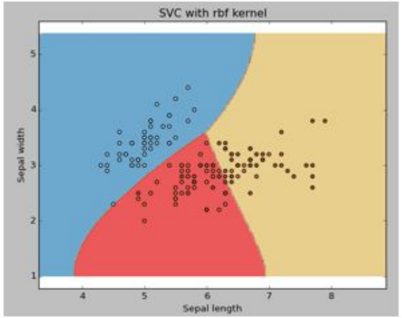
C =100



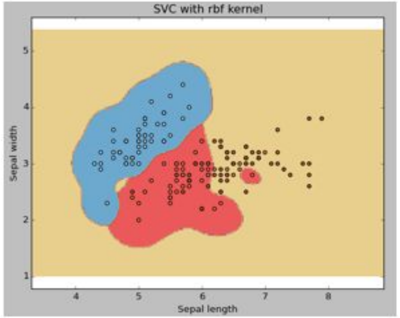
c =1000



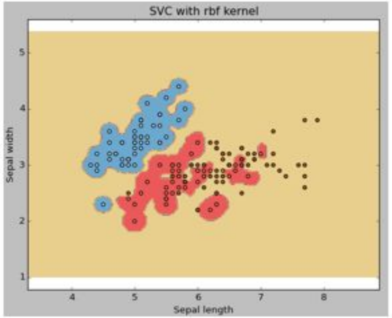
gamma =0



gamma =10



gamma =100



C: Предельный параметр C для ошибки. Он также контролирует расстояние между гладкой границей принятия решений и правильной классификацией точек тренировки.

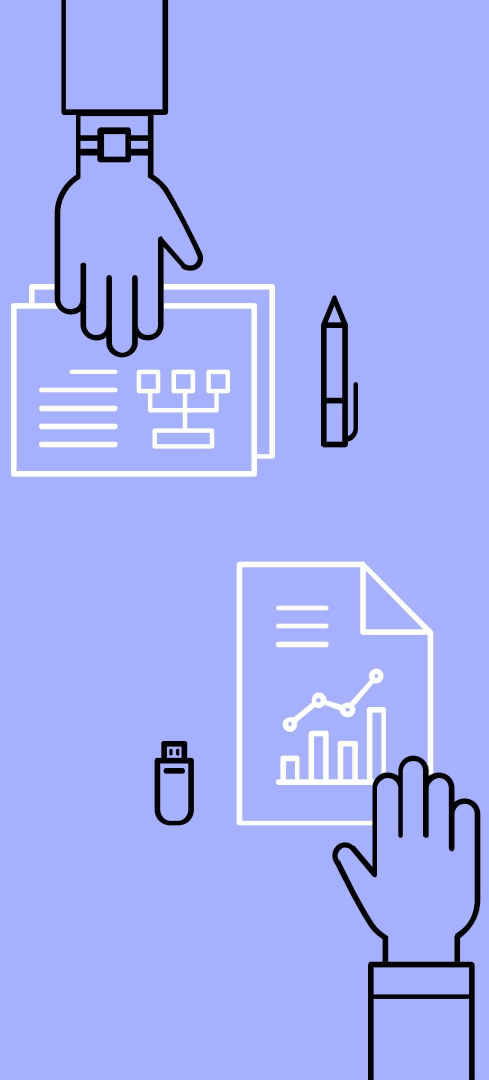
gamma : Коэффициент ядра для 'rbf', 'poly' и 'sigmoid'. Чем выше значение гаммы, тем лучше будет задаваться как набор данных тренировки.

Подбор параметров SVM: C параметр

Вопрос:

Допустим мы взяли большой параметр C, это значит что:

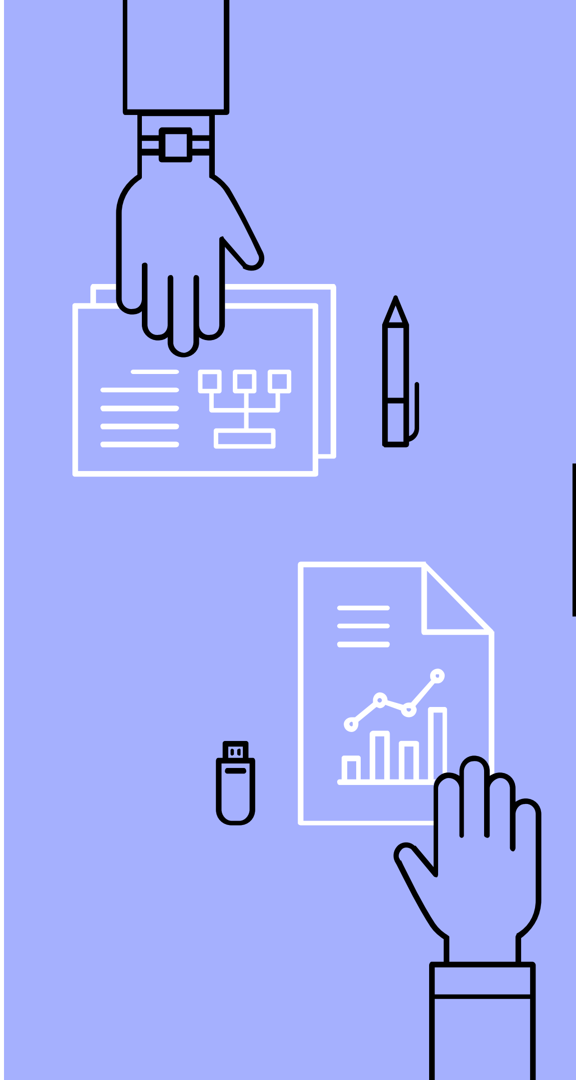
- Будет более изогнутая кривая, разделяющая классы
- Тренировочные данные будут лучше разбиты на классы



Подбор параметров SVM: С параметр

Ответ:

- Для этого нужно использовать ядра
- Верно:)



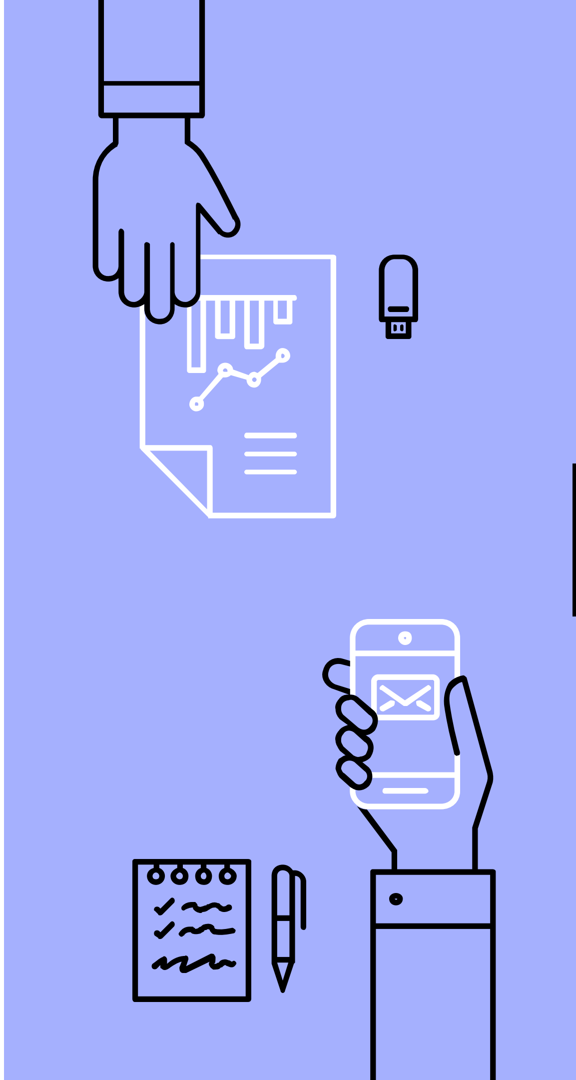
Преимущества и недостатки SVM

Преимущества:

- Задача выпуклого квадратичного программирования имеет единственное решение.
- Выделяется множество опорных объектов.
- Имеются эффективные численные методы для SVM.
- Изящное обобщение на нелинейные классификаторы.

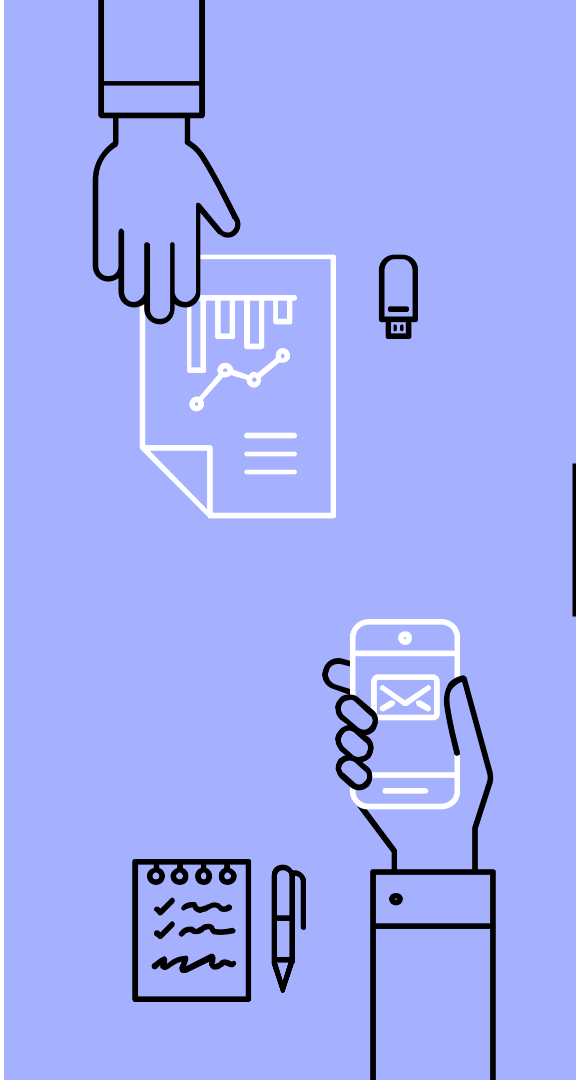
Недостатки:

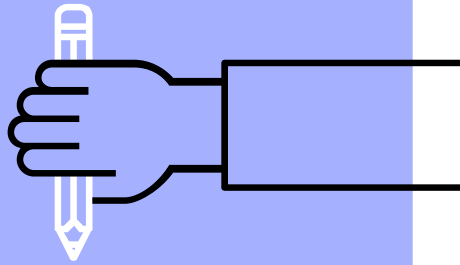
- Опорными объектами могут становиться выбросы.
- Нет отбора признаков в исходном пространстве X.
- Приходится подбирать константу C.



Выводы

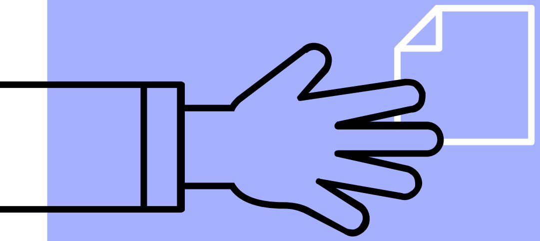
- Метод опорных векторов — линейный классификатор с кусочно-линейной функцией потерь (hinge loss) и $L2$ -регуляризатором
- Придуман метод был из соображений максимизации зазора между классами
- В случае линейно разделимой выборки это означает просто максимизацию ширины разделяющей полосы
- А в случае линейно неразделимой выборки просто добавляется возможность попадания объектов в полосу и штрафы за это



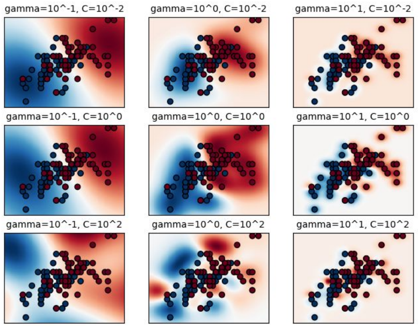


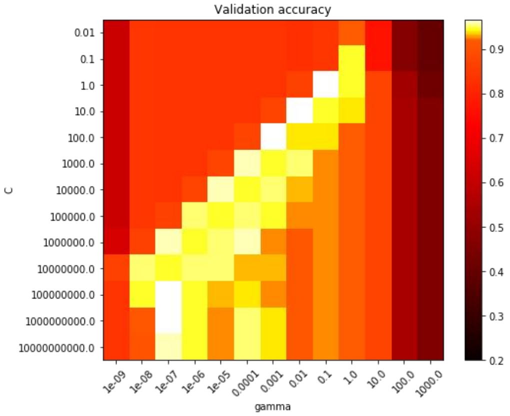
4.

Примеры кода



Подбор параметров SVM






```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize

from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import GridSearchCV

# Загрузите и подготовьте набор данных
iris = load_iris()
X = iris.data
y = iris.target

X_2d = X[:, :2]
X_2d = X_2d[y > 0]
y_2d = y[y > 0]
y_2d -= 1

scaler = StandardScaler()
X = scaler.fit_transform(X)
X_2d = scaler.fit_transform(X_2d)

C_range = np.logspace(-2, 10, 13)
gamma_range = np.logspace(-9, 3, 13)
param_grid = dict(gamma=gamma_range, C=C_range)
cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, random_state=42)
grid = GridSearchCV(SVC(), param_grid=param_grid, cv=cv)
grid.fit(X, y)

print("The best parameters are %s with a score of %0.2f"
      % (grid.best_params_, grid.best_score_))

C_2d_range = [1e-2, 1, 1e2]
gamma_2d_range = [1e-1, 1, 1e1]
classifiers = []
for C in C_2d_range:
    for gamma in gamma_2d_range:
        clf = SVC(C=C, gamma=gamma)
        clf.fit(X_2d, y_2d)
        classifiers.append((C, gamma, clf))

plt.figure(figsize=(8, 6))
xx, yy = np.meshgrid(np.linspace(-3, 3, 200), np.linspace(-3, 3, 200))
for (k, (C, gamma, clf)) in enumerate(classifiers):
    # оценивать функцию решения в сетке
    Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # визуализировать функцию решения для этих параметров
    plt.subplot(len(C_2d_range), len(gamma_2d_range), k + 1)
    plt.title("gamma=10^%d, C=10^%d" % (np.log10(gamma), np.log10(C)),
              size='medium')

    # визуализировать влияние параметра на функцию принятия решения
    plt.pcolormesh(xx, yy, -Z, cmap=plt.cm.RdBu)
    plt.scatter(X_2d[:, 0], X_2d[:, 1], c=y_2d, cmap=plt.cm.RdBu_r,
                edgecolors='k')
    plt.xticks(())
    plt.yticks(())
    plt.axis('tight')

scores = grid.cv_results_['mean_test_score'].reshape(len(C_range),
                                                     len(gamma_range))
plt.figure(figsize=(8, 6))
plt.subplots_adjust(left=.2, right=0.95, bottom=0.15, top=0.95)
plt.imshow(scores, interpolation='nearest', cmap=plt.cm.hot,
           norm=MidpointNormalize(vmin=0.2, midpoint=0.92))
plt.xlabel('gamma')
plt.ylabel('C')
plt.colorbar()
plt.xticks(np.arange(len(gamma_range)), gamma_range, rotation=45)
plt.yticks(np.arange(len(C_range)), C_range)
plt.title('Validation accuracy')
plt.show()
```

```
The best parameters are {'C': 1.0, 'gamma': 0.1} with a score of 0.97
```

Библиотечная реализация SVM

```python
from sklearn import datasets

cancer = datasets.load_breast_cancer()
```

```python
from sklearn.model_selection import train_test_split

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, test_size=0.3,random_state=109) # 70
```

```python
from sklearn import svm

#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

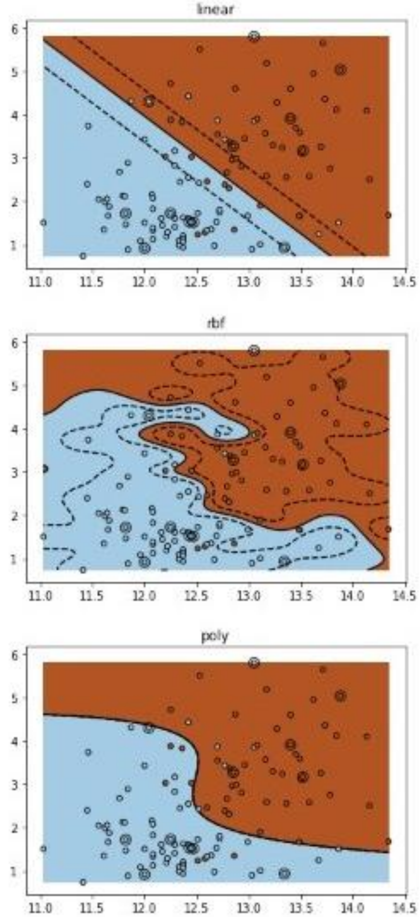
```python
from sklearn import metrics

# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9649122807017544
```

```python
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Precision: 0.9811320754716981
Recall: 0.9629629629629629
```

SVM с разными ядрами




```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets, svm
from sklearn import metrics

#datasets для классификации : wine, iris

iris = datasets.load_wine()
X = iris.data
y = iris.target

X = X[y != 0, :2]
y = y[y != 0]
n_sample = len(X)

np.random.seed(0)
order = np.random.permutation(n_sample)
X = X[order]
y = y[order].astype(np.float)

X_train = X[:int(.9 * n_sample)]
y_train = y[:int(.9 * n_sample)]
X_test = X[int(.9 * n_sample):]
y_test = y[int(.9 * n_sample):]

# fit the model
for fig_num, kernel in enumerate(('linear', 'rbf', 'poly')):
    clf = svm.SVC(kernel=kernel, gamma=10)
    clf.fit(X_train, y_train)

    plt.figure(fig_num)
    plt.clf()
    plt.scatter(X[:, 0], X[:, 1], c=y, zorder=10, cmap=plt.cm.Paired,
                edgecolor='k', s=20)
    y_pred = clf.predict(X_test)
    # Model Accuracy: how often is the classifier correct?
    print("Accuracy:", kernel,metrics.accuracy_score(y_test, y_pred))

    # Model Precision: what percentage of positive tuples are labeled as such?
    print("Precision:",metrics.precision_score(y_test, y_pred))

    # Model Recall: what percentage of positive tuples are labelled as such?
    print("Recall:",metrics.recall_score(y_test, y_pred))

    # Circle out the test data
    plt.scatter(X_test[:, 0], X_test[:, 1], s=80, facecolors='none',
                zorder=10, edgecolor='k')

    plt.axis('tight')
    x_min = X[:, 0].min()
    x_max = X[:, 0].max()
    y_min = X[:, 1].min()
    y_max = X[:, 1].max()

    XX, YY = np.mgrid[x_min:x_max:200j, y_min:y_max:200j]
    Z = clf.decision_function(np.c_[XX.ravel(), YY.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(XX.shape)
    plt.pcolormesh(XX, YY, Z > 0, cmap=plt.cm.Paired)
    plt.contour(XX, YY, Z, colors=['k', 'k', 'k'],
                linestyles=['--', '-', '--'], levels=[-.5, 0, .5])

    plt.title(kernel)
plt.show()
```

```
Accuracy: linear 0.8333333333333334
Precision: 1.0
Recall: 0.75
Accuracy: rbf 0.9166666666666666
Precision: 1.0
Recall: 0.875
Accuracy: poly 0.9166666666666666
Precision: 1.0
Recall: 0.875
```

Support Vector Machine без использования библиотечной реализации

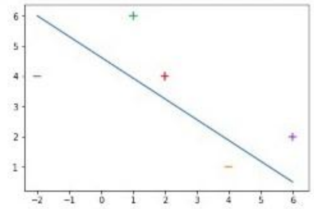
```
import numpy as np

X = np.array([
    [-2,4,-1],
    [4,1,-1],
    [1, 6, -1],
    [2, 4, -1],
    [6, 2, -1],

])

y = np.array([-1,-1,1,1,1])

def svm_sgd(X, Y):

    w = np.zeros(len(X[0]))
    eta = 1
    epochs = 100000


    for epoch in range(1,epochs):
        for i, x in enumerate(X):
            if (Y[i]*np.dot(X[i], w)) < 1:
                w = w + eta * ( (X[i] * Y[i]) + (-2  *(1/epoch)* w) )
            else:
                w = w + eta * (-2  *(1/epoch)* w)

    return w

w = svm_sgd(X,y)
print(w)
```

```
[ 1.58876117  3.17458055 11.11863105]
```

```
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
```

```
for d, sample in enumerate(X):
    # Plot the negative samples
    if d < 2:
        plt.scatter(sample[0], sample[1], s=120, marker='_', linewidths=2)
    # Plot the positive samples
    else:
        plt.scatter(sample[0], sample[1], s=120, marker='+', linewidths=2)

# Print a possible hyperplane, that is seperating the two classes.
plt.plot([-2,6],[6,0.5])
```

```
[<matplotlib.lines.Line2D at 0x7fac85f58550>]
```



Мы будем запускать обучение 100000 раз. Наш параметр обучения eta равен 1. В качестве регулирующего параметра мы выбираем 1 / t, поэтому этот параметр будет уменьшаться по мере увеличения числа эпох.

Описание кода

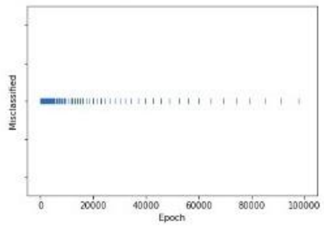
строка 2: Инициализируем вектор веса с нулями
строка 3: Установим скорость обучения на 1
строка 4: Зададим количество эпох
строка 6: Итерация n раз по всему набору данных. Итератор начинается с 1, чтобы избежать деления на ноль во время вычисления параметра регуляризации
строка 7: Итерация по каждому образцу в наборе данных.
строка 8: Условие классификации $y_i\langle x_i, w\rangle < 1$
строка 9: Правило обновления для весов $w = w + \eta(y_i x_i - 2\lambda w)$
строка 11: если классифицировать правильно, просто обновите вектор веса с помощью вычисленного члена регуляризатора $w = w + \eta(-2\lambda w)$.

Обучение SVM

Напечатаем количество неправильно классифицированных и правильно классифицированных образцов от количества итераций обучения.

```
def svm_sgd_plot(X, Y):

    w = np.zeros(len(X[0]))
    eta = 1
    epochs = 100000
    errors = []


    for epoch in range(1,epochs):
        error = 0
        for i, x in enumerate(X):
            if (Y[i]*np.dot(X[i], w)) < 1:
                w = w + eta * ( (X[i] * Y[i]) + (-2  *(1/epoch)* w) )
                error = 1
            else:
                w = w + eta * (-2  *(1/epoch)* w)
        errors.append(error)

    plt.plot(errors, '|')
    plt.ylim(0.5,1.5)
    plt.axes().set_yticklabels([])
    plt.xlabel('Epoch')
    plt.ylabel('Misclassified')
    plt.show()
```

```
svm_sgd_plot(X,y)
```



Вышеприведенный график показывает, что svm делает меньше ошибочных классификаций, тем больше он работает.

Весовой вектор SVM, включающий период смещения после 100000 эпох, равен (1,56,3,17,11,12). Теперь мы можем извлечь следующую функцию прогнозирования:

$$f(x) = \langle x, (1.56, 3.17)\rangle - 11.12$$

Весовой вектор равен (1,56,3,17), а термин смещения - третья запись 11.12.

Оценка

Давайте вручную проверим правильно ли мы обучили сеть:

Первый пример $(-2, 4)$, должен быть отрицательным:

$$-2 * 1,56 + 4 * 3,17 - 11,12 = sign(-1,56) = -1$$

Второй пример $(4, 1)$, должен быть положительным:

$$4 * 1,56 + 1 * 3,17 - 11,12 = sign(-1,71) = -1$$

Третий пример $(1, 6)$, должен быть отрицательным:

$$1 * 1,56 + 6 * 3,17 - 11,12 = sign(9,46) = +1$$

Четвертый пример $(2, 4)$, должен быть положительным:

$$2 * 1,56 + 4 * 3,17 - 11,12 = sign(4,68) = +1$$

Пятый пример $(6, 2)$, должен быть положительным:

$$6 * 1,56 + 2 * 3,17 - 11,12 = sign(4,58) = +1$$

Добавим новые точки для проверки классификатора:

Первый тестовый пример $(2, 2)$, должен быть отрицательным:

$$2 * 1,56 + 2 * 3,17 - 11,12 = sign(-1,66) = -1$$

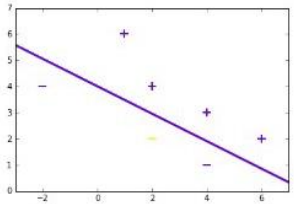
Второй тестовый пример $(4, 3)$, должен быть положительным:

$$4 * 1,56 + 3 * 3,17 - 11,12 = sign(4,63) = +1$$

Оба образца классифицируются правильно. Чтобы проверить это геометрически, давайте нарисуем образцы, включая тестовые образцы и гиперплоскость.

```
for d, sample in enumerate(X):
    # Plot the negative samples
    if d < 2:
        plt.scatter(sample[0], sample[1], s=120, marker='_', linewidths=2)
    # Plot the positive samples
    else:
        plt.scatter(sample[0], sample[1], s=120, marker='+', linewidths=2)

# Add our test samples

plt.scatter(2,2, s=120, marker='_', linewidths=2, color='yellow')
plt.scatter(4,3, s=120, marker='+', linewidths=2, color='blue')

# Print the hyperplane calculated by svm_sgd()
x2=[w[0],w[1],-w[1],w[0]]
x3=[w[0],w[1],w[1],-w[0]]

x2x3 =np.array([x2,x3])
X,Y,U,V = zip(*x2x3)
ax = plt.gca()
ax.quiver(X,Y,U,V,scale=1, color='blue')
```

```
<matplotlib.quiver.Quiver at 0x7fb5adfda400>
```



Спасибо!

Остались вопросы?

Контакты:

- https://vk.com/id104544842
- komleva.1999@inbox.ru