In [1]:

```
import IPython.display
IPython.display.display_latex(IPython.display.Latex(filename="../macros.tex"))
```

Линейный классификатор

Гипотеза: классы разделяются гиперплоскостью.

Дано:

$\hat{X} \in \mathbb{R}^M, C = \{-1, +1\}$

$(X, Y)^N$ - тренировочная выборка

Найти:

$\theta \in \mathbb{R}^M, \theta_0 \in \mathbb{R}$

$$\alpha(x, \theta, \theta_0) = sign(< x, \theta > -\theta_0)$$

- $< x_1, x_2 >$ - скалярное произведение
- θ - нормаль к гиперплоскости
- θ_0 - сдвиг

Отступ

определим:

$$\mathbb{M}(x_i) = \mathbb{M}_i = y_i * (< x_i, \theta > -\theta_0)$$

$\mathbb{M}(x_i) < 0 \Leftrightarrow \alpha$ возвращает неверный ответ

Empirical Risk:

$$Q(\theta, \theta_0, X) = \sum_{i=1}^{N} [\mathbb{M}_i < 0]$$

Заменим:

$[\mathbb{M}_i < 0] \leq L(M_i)$, где

$L : \mathbb{R} \to \mathbb{R}_+$ - непрерывная, как правило, невозрастающая.

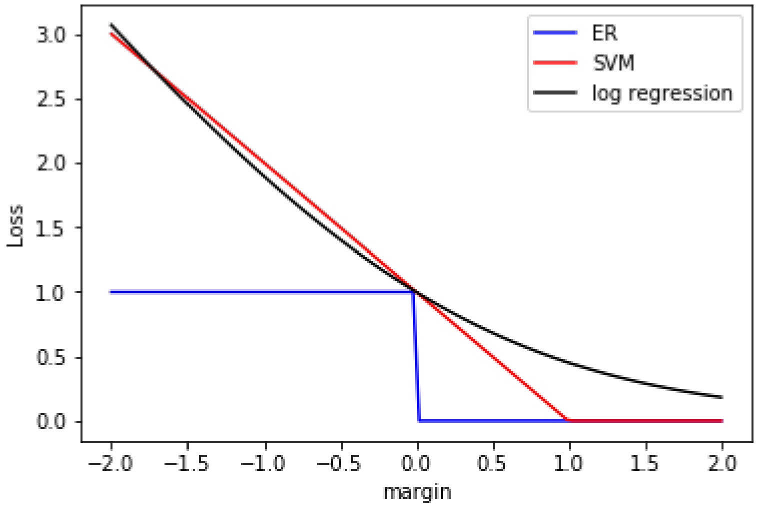
После замены функции потерь минимизируется не сам функционал эмпирического риска, а его верхняя оценка:

$$\sum_{i=1}^{N} L(\mathbb{M}_i) \to min$$

Различные апроксимации пороговых функций потерь:

- $(1 - \mathbb{M})^2$ - квадратичная
- $(1 - \mathbb{M})_+$ - кусочно-линейная
- $(2 * (1 + e^{\mathbb{M}})^{-1})$ - сигмоидальная(SVM)
- $log(1 + e^{-\mathbb{M}})$ - логистическая(лог регрессия)

In [2]:
```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-20, 20, 100) / 10.0

plt.plot(X, [1 if x < 0 else 0 for x in X], color = [0, 0, 1], label="ER")
plt.plot(X, [0 if ((1 - x) < 0) else (1 - x) for x in X], color = [1, 0, 0], label="SVM"
)
plt.plot(X, np.log2(1 + np.exp(-1 * X)), color = [0, 0, 0], label="log regression")

plt.legend()

plt.xlabel("margin")
plt.ylabel("Loss")

plt.show()
```



Логистическая регрессия Logistic Regression

Классы: 0 - negative, 1 - positive.

Логистическая функция(сигмоида) Logistic function(sigmoid function):

$$g(z) = \frac{1}{1 + e^{-z}}$$

Имеем:

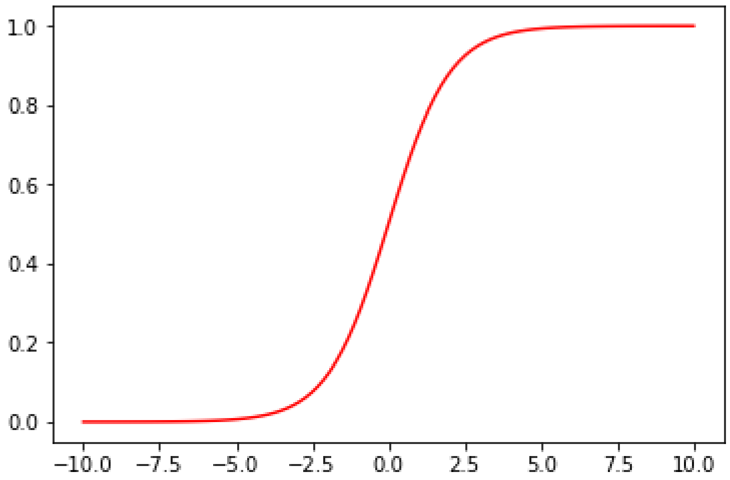
$$\alpha(x, \theta) = g(\theta^T * x)$$

In [3]:

```
X = np.linspace(-100, 100, 100) / 10.0
plt.plot(X, 1 / (1 + np.exp(-1 * X)), color = [1, 0, 0])
```

Out[3]:

```
[<matplotlib.lines.Line2D at 0x1f0957c83c8>]
```



Можно представить как:

$$\alpha_\theta(x_i) = \overline{P}(y_i = 1 | x_i, \theta)$$

функция потерь:

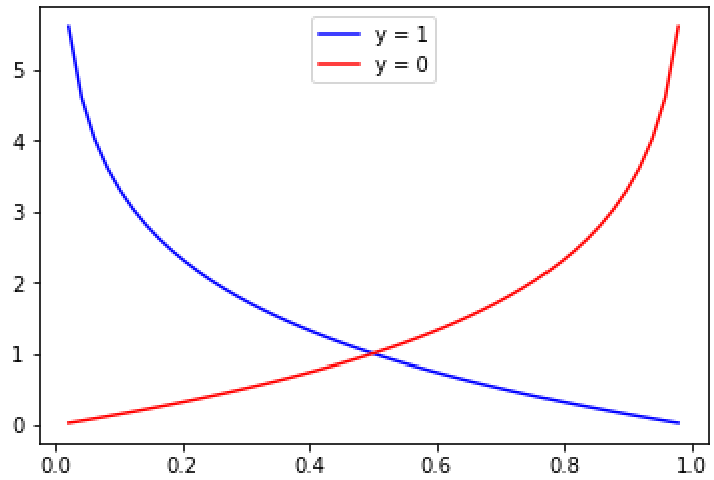
$$L(\theta, x) = \begin{cases} -log(\alpha_\theta(x)), ify = 1 \\ -log(1 - \alpha_\theta(x)), ify = 0 \end{cases}$$

In [4]:

```
X = np.linspace(0, 10, 50) / 10.0
X = X[1:-1]
plt.plot(X, np.log2(X) * -1, color = [0, 0, 1], label="y = 1")
plt.plot(X, np.log2(1 - X) * -1, color = [1, 0, 0], label="y = 0")
plt.legend()
```

Out[4]:

```
<matplotlib.legend.Legend at 0x1f095861208>
```



Empirical Risk:

$$Q(\theta) = -\frac{1}{N}\Big[\sum_{i=1}^{N}\Big(y^i * log(\alpha_\theta(x_i)) + (1-y^i) * log(1-\alpha_\theta(x^i))\Big)\Big]$$

Используя:

$$g'_x(z(x)) = g(x) * g(1-x) * z'_x$$

Получим(упр.):

$$\frac{\partial}{\partial\theta_j}Q(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(\alpha_\theta(x^i) - y^i\right) * x_j^i)$$

Градиентный спуск:

$$\theta_j := \theta_j - \gamma\frac{1}{N}\sum_{i=1}^{N}\left(\alpha_\theta(x^i) - y^i\right) * x_j^i)$$

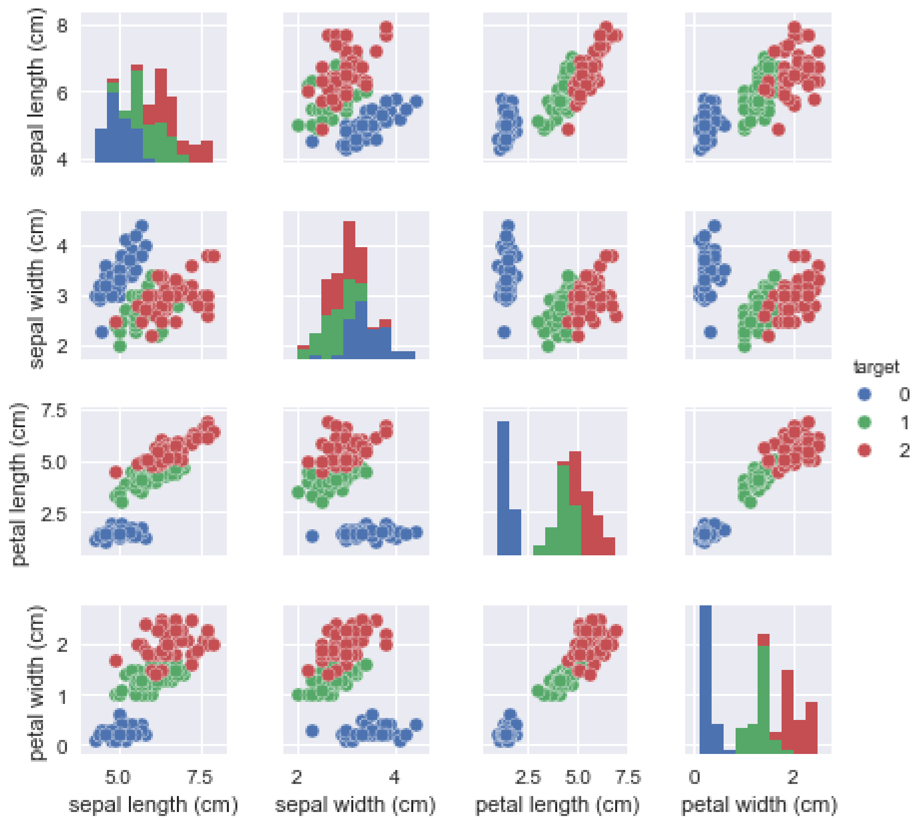
In [5]:
```
from sklearn import linear_model, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

import seaborn as sns

import pandas as pd
```

In [6]:
```
iris = datasets.load_iris()
```

In [7]:
```
iris.feature_names
```

Out[7]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [8]:
```
X_DF = pd.DataFrame(data = iris.data, columns = iris.feature_names)
X_DF["target"] = iris.target
g = sns.pairplot(X_DF, hue="target",vars=iris.feature_names, size=1.5)
```



In [9]:

```
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42
)
```

In [10]:

```
logreg = linear_model.LogisticRegression()
logreg.fit(X_train, y_train)
```

Out[10]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [11]:
```
y_pred = logreg.predict(X_test)
```

In [12]:
```
confusion_matrix(y_test, y_pred)
```

Out[12]:
```
array([[29,  0,  0],
       [ 0, 21,  2],
       [ 0,  0, 23]])
```

In [13]:
```
logreg.coef_, logreg.intercept_
```

Out[13]:
```
(array([[ 0.35663172,  1.19008689, -1.93201857, -0.87482579],
        [ 0.4362906 , -1.39149169,  0.34607673, -0.85671156],
        [-1.35422035, -1.23986252,  2.01043039,  1.82999058]]),
 array([ 0.22843635,  0.65635471, -0.8781996 ]))
```

Логистическая регрессия: вывод сигмоиды

Рассмотрим задачу бинарной классификации, то есть задачу пример-ответ $\{X_i, Y_i\}$, где величины y_i могут принимать только значения из множества {0, 1}. Хотим предсказывать, с какой вероятностью (p_i) объект x_i принадлежит тому или иному классу.

Самый простой вариант: $p_i = x_i * \theta$

Однако слева величина от 0 до 1, а справа просто число.

Вместо вераоятности рассмотрим отношение шансов:

$$odds_i = \frac{p_i}{1 - p_i}$$

то есть отношение числа благоприятных исходов к неблагоприятным.Но данная величина всегда больше 0, что не подходит нам.

Давайте прологарифмируем, получим величину от $-inf$ до $+inf$

$$logit(odds_i) = ln(\frac{p_i}{1 - p_i})$$

$$\eta_i = p_i = x_i * \theta$$

А полученную функцию будем называть логистическим преобразорванием. Тогда:

$$ln(\frac{p_i}{1 - p_i}) = x_i * \theta$$

$$\eta_i = ln(\frac{p_i}{1 - p_i})$$

$$e^{\eta_i}(1 - p_i) = p_i$$

$$\frac{e^{\eta_i}}{(1 + e^{\eta_i})} = p_i$$

$$p_i = \frac{1.}{(1 + e^{-x_i * \theta})}$$

Softmax: **как логистическая регрессия, но на много классов**

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=0}^{k} e^{x_j}}$$

- бинарная классификация

$$\overline{P}(y_i = 1|x_i, \theta) = g(\theta^T * x)$$

- многоклассовая

$$\overline{P}(y_i = c|x_i, \theta_c) = softmax(\theta_c^T * x)$$