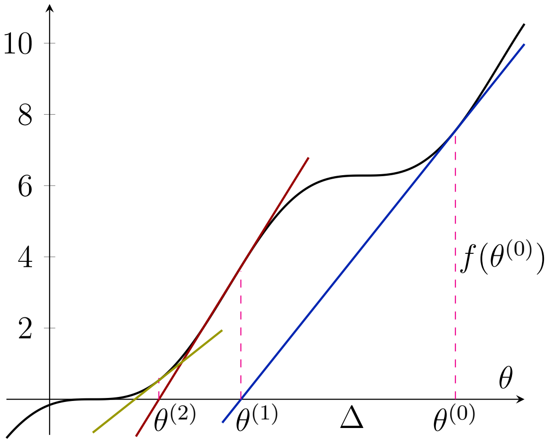
Lecture 3

1. Newton's method
2. Exponential family
3. Generalized Linear Models (GLM)
4. Generative learning algorithm

1 Newton's method

Newton's method is an algorithm that help to solve an equation $f(\theta) = 0$.



We start from some random $\theta^{(0)}$, by definition:

$$f'(\theta^{(0)}) = \frac{f(\theta^{(0)})}{\Delta},$$

which implies that $\Delta = \dfrac{f(\theta^{(0)})}{f'(\theta^{(0)})}$. Then

$$\theta^{(1)} = \theta^{(0)} - \frac{f(\theta^{(0)})}{f'(\theta^{(0)})}$$

or, in general,

$$\theta^{(t+1)} = \theta^{(t)} - \frac{f(\theta^{(t)})}{f'(\theta^{(t)})}.$$

To maximize likelihood $l(\theta)$ we find θ such that $l'(\theta) = 0$, then one iteration of the Newton's methods is written as

$$\theta^{(t+1)} = \theta^{(t)} - \frac{l'(\theta^{(t)})}{l''(\theta^{(t)})}.$$

This algorithm has a quadratic convergence. It means that error decreasing as square after each iteration. For example, in linear regression we need just 3 iterations to find the parameters with very good accuracy.

When the problem is to find several parameters, one iteration of the Newton's methods can be written as a vector equation:

$$\theta^{(t+1)} = \theta^{(t)} - H^{-1}\nabla_\theta l,$$

where θ is a vector and H is the Hessian matrix:

$$H_{ij} = \frac{\partial^2 l}{\partial\theta_i \partial\theta_j}.$$

Notice that if we want to minimize something (instead of maximization) the algorithm does not change.

2 Exponential family

In this section we generalise the ideas of linear and logistic regressions. Before we considered two cases:

- $y \in \mathbb{R} \Rightarrow$ Gaussian distribution $N(\mu, \sigma^2) \Rightarrow$ linear regression
- $y \in \{0, 1\} \Rightarrow$ Bernoulli distribution with parameter φ such that $p(y = 1|\varphi) = \varphi \Rightarrow$ logistic regression

We will show that in both cases we deal with particular cases of general class of exponential family distributions:

$$p(y; \eta) = b(y) \exp\left(\eta^T T(y) - a(\eta)\right),$$

where η is a natural parameter, $T(y)$ is a sufficient statistics (in many cases $T(y) = y$).

For different choices of a, b and T we will have different distributions:

1. Bernoulli distribution with φ: $p(y = 1; \varphi) = \varphi$. Then

$$\begin{aligned} p(y; \varphi) &= \varphi^y(1-\varphi)^{1-y} = \exp\left(\ln\left(\varphi^y(1-\varphi)^{1-y}\right)\right) = \\ &= \exp\left(y\ln\varphi + (1-y)\ln(1-\varphi)\right) = \exp\left(\ln\frac{\varphi}{1-\varphi}y + \ln(1-\varphi)\right). \end{aligned}$$

By introducing the notations $\eta = \ln\frac{\varphi}{1-\varphi}$, $T(y) = y$, $-a(\eta) = \ln(1-\varphi)$, $b(y) = 1$, we obtain the exponential family formula. Notice that we can find

$$\varphi = \frac{1}{1+e^{-\eta}},$$

then $a(\eta) = \ln(1 + e^\eta)$.

2. Gaussian distribution $N(\mu, \sigma^2)$. For simplicty we set $\sigma^2 = 1$. Then

$$p(y;\mu) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(y-\mu)^2\right) = \ldots = \\ = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2\right) \exp\left(\mu y - \frac{1}{2}\mu^2\right).$$

By introducing the notations $b(y) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2\right)$, $\eta = \mu$, $T(y) = y$, $a(\eta) = \frac{1}{2}\eta^2$, we obtain the exponential family formula.

3 Generalized Linear Models (GLM)

The Generalized Linear Models use the exponential family distributions to build different algorithms in one general framework. Assuming some distribution for the output y we obtain the form for the hypothesis $h_\theta(x)$ and find the parameters by finding the maximum of log-likelihood. We assume the following:

- $y \mid x; \theta \sim \text{ExpFamily}(\eta)$
- given x our goal is to output $E[T(y) \mid x]$, in other words,

$$h(x) = E[T(y) \mid x]$$

- $\eta = \theta^T x$ (in more general case if $\eta \in \mathbb{R}^k$, then $\eta_i = \theta_i^T x$)

Examples of GLM:

1. **Bernoulli distribution**

   - $y \mid x; \theta \sim \text{ExpFamily}(\eta)$
   - for fixed x and θ, the algorithm output is

$$h_\theta(x) = E[y \mid x; \theta] = p(y = 1 \mid x; \theta) = \varphi = \frac{1}{1+e^{-\eta}} = \frac{1}{1+e^{-\theta^T x}}$$

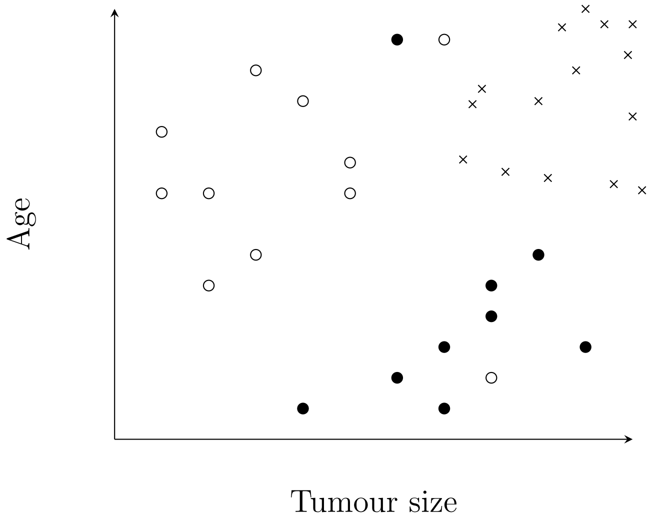
   Definition.

$$g(\eta) = E[y \mid \eta] = \frac{1}{1+e^{-\eta}} \text{ is a canonical response function}$$
$$g^{-1} \text{ is a canonical link function}$$

2. **Gaussian distribution**

   Exercise for the homework

3. **Multinomial distribution**: $y \in \{1, \ldots, k\}$



Parameters φ_1, φ_2, ..., φ_k are defined as$p(y = i) = \varphi_i$. Such formulation is redundant, because φ_k can be expressed as $\varphi_k = 1 - \varphi_1 - \ldots - \varphi_{k-1}$. That is why we do not take φ_k as a parameter. We introduce function $T(y), y \in \{1, \ldots, k\}$ as follows:

$$T(1) = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix},\ T(2) = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \ldots, T(k-1) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

and

$$T(k) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix},\ T(1), T(2), \ldots, T(k) \in \mathbb{R}^{k-1}.$$

The indicator function $\mathbb{1}\{True\} = 1$, $\mathbb{1}\{False\} = 0$, for example, $\mathbb{1}\{2 = 3\} = 0$. Then $T(y)_i = \mathbb{1}\{y = i\}$. The distribution:

$$\begin{aligned} P(y) &= \varphi_1^{\mathbb{1}\{y=1\}} \cdot \varphi_2^{\mathbb{1}\{y=2\}} \cdot \ldots \cdot \varphi_k^{\mathbb{1}\{y=k\}} = \\ &= \varphi_1^{T(y)_1} \cdot \varphi_2^{T(y)_2} \cdot \ldots \cdot \varphi_{k-1}^{T(y)_{k-1}} \cdot \varphi_k^{1-\sum\limits_{j=1}^{k-1} T(y)_j} = \ldots = \\ &= b(y) \exp\left(\eta^T T(y) - a(\eta)\right), \end{aligned}$$

where

$$\eta = \begin{bmatrix} \ln(\varphi_1/\varphi_k) \\ \vdots \\ \ln(\varphi_{k-1}/\varphi_k) \end{bmatrix} \in \mathbb{R}^{k-1},\ a(\eta) = -\ln \varphi_k,\ b(y) = 1.$$

We could solve these equations with respect to η and obtain

$$\varphi_i = \frac{e^{\eta_i}}{1 + \sum\limits_{j=1}^{k-1} e^{\eta_j}} = \frac{e^{\theta_i^T x}}{1 + \sum\limits_{j=1}^{k-1} e^{\theta_j^T x}}, i = 1 \ldots k - 1.$$

The purpose of these transformation is to obtain the learning algorithm:

$$h_\theta(x) = E[T(y) \mid x;\theta] = E\left[\begin{array}{c|c} \mathbb{1}\{y=1\} & \\ \vdots & x;\theta \\ \mathbb{1}\{y=k-1\} & \end{array}\right] =$$

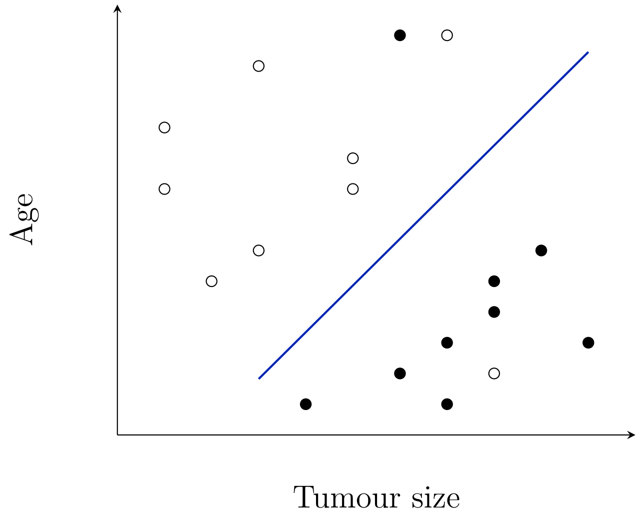
$$= \begin{bmatrix} \varphi_1 \\ \vdots \\ \varphi_{k-1} \end{bmatrix} = \begin{bmatrix} \dfrac{e^{\theta_1^T x}}{1+\sum\limits_{j=1}^{k-1} e^{\theta_j^T x}} \\ \vdots \\ \dfrac{e^{\theta_{k-1}^T x}}{1+\sum\limits_{j=1}^{k-1} e^{\theta_j^T x}} \end{bmatrix}$$

This algorithm is called **softmax regression** - generalisation of logistic regression for k classes. To fit the parameters for the given training set $(x^{(1)}, y^{(1)})$, ..., $(x^{(m)}, y^{(m)})$, we find the likelihood in the form

$$L(\theta) = \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)};\theta) = \prod_{i=1}^{m} \varphi_1^{\mathbb{1}\{y^{(i)}=1\}} \cdot \varphi_2^{\mathbb{1}\{y^{(i)}=2\}} \cdot \ldots \cdot \varphi_k^{\mathbb{1}\{y^{(i)}=k\}},$$

where $\varphi_1 = \dfrac{e^{\theta_1^T x}}{1+\sum\limits_{j=1}^{k-1} e^{\theta_j^T x}}$. The last step will be to find the log-likelihood, find the derivatives and use stochastic gradient descent to maximize the log-likelihood function with respect to parameters θ.

4 Generative learning algorithm



The logistic regression tries to find the straight line between two classes. This algorithm belongs to the class of **discriminative algorithms**. The idea of properties for discriminative algorithms is the following:

- learns $p(y \mid x)$
- or learns $h_\theta(x) \in \{0, 1\}$

Now we are going to talk about different approach. Assuming that we have two classes, we build two models: one - on positive examples only, the second - on negative examples only. When we get new sample we try to understand which model this sample match better and based on this matching we make a conclusion. This algorithm belongs to the class of **generative algorithms**. The idea of the generative algorithms is

- learn $p(x \mid y)$ and $p(y)$
- use Bayes rule to get $p(y = 1 \mid x) = \dfrac{p(x \mid y = 1)p(y)}{p(x)}$, where

$$p(x) = p(x \mid y = 0)p(y = 0) + p(x \mid y = 1)p(y = 1)$$

4.1 Python implementation

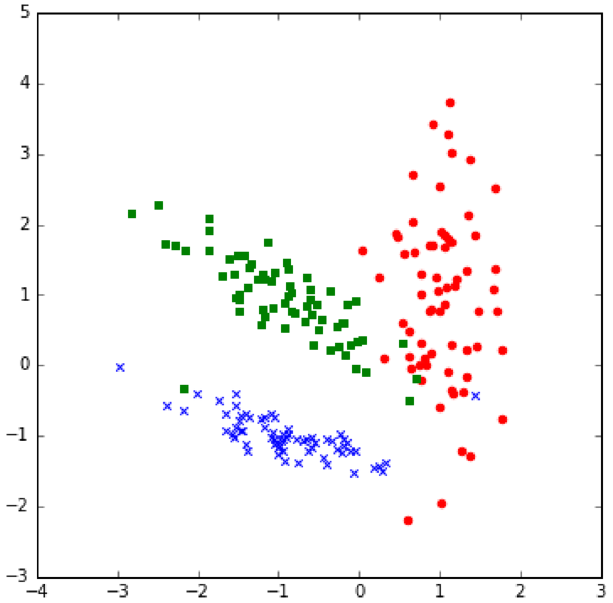
Loading necessary libraries:

```
In [1]: from matplotlib import cm
        import matplotlib.pyplot as plt
        import numpy as np
        import sklearn.datasets as ds
```

The softmax regression in `scikit-learn` can be trained using LogisticRegression class. We generate data with 3 classes and 2 features first:

```
In [2]: from sklearn.linear_model import LogisticRegression
```

```
In [3]: X,y = ds.make_classification(n_features=2,
                                     n_redundant=0,
                                     n_informative=2,
                                     n_clusters_per_class=1,
                                     n_classes=3,
                                     n_samples=200,
                                     random_state=3216)
        ix0 = [i for i,x in enumerate(y) if x == 0]
        ix1 = [i for i,x in enumerate(y) if x == 1]
        ix2 = [i for i,x in enumerate(y) if x == 2]
        fig = plt.figure(figsize=(6,6))
        #ax = plt.axes(xlim=(-3.1, 3.1), ylim=(-3.1, 3.1))
        plt.scatter(X[ix0,0],X[ix0, 1],marker=’o’,color=’red’)
        plt.scatter(X[ix1,0],X[ix1, 1],marker=’x’,color=’blue’)
        plt.scatter(X[ix2,0],X[ix2, 1],marker=’s’,color=’green’)
        plt.show()
```



Training the softmax regression is easy as before:

```
In [4]: model = LogisticRegression(multi_class='multinomial', solver='newton-cg')
        model.fit(X,y)
        print 'Intercepts:'
        print model.intercept_
        print 'Coefficients:'
        print model.coef_
```

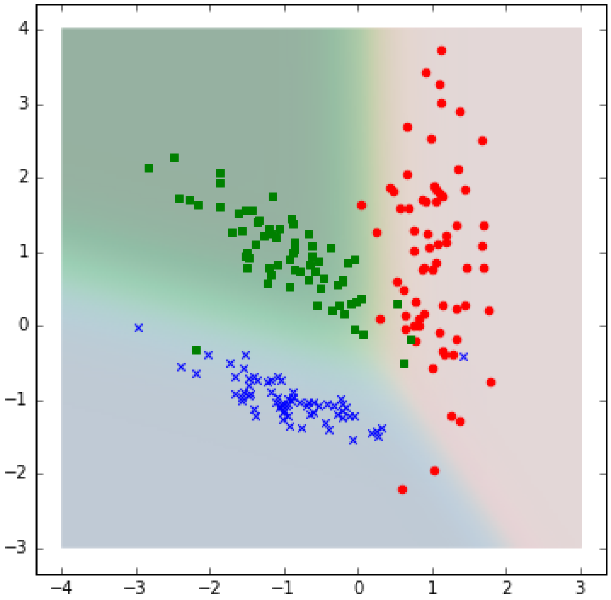
```
Intercepts:
[-0.10153312 -0.56865552  0.67018864]
Coefficients:
[[ 2.80913084  1.10436996]
 [-1.8280927  -2.14320504]
 [-0.98103814  1.03883507]]
```

It is possible to get probabilities from the model using the method `predict_proba()`, but I have implemented the function `softmax_probs()` to evaluate probabilities with `model.intercept_` and `model.coeff_` attributes.

```
In [5]: def softmax_probs(X, model):
            Xb = np.hstack((np.ones((X.shape[0],1)), X))
            thetas = np.hstack((model.intercept_.reshape((-1,1)), model.coef_))
            probs = np.exp(np.dot(Xb, np.transpose(thetas)))
            probs_sums = probs.sum(axis=1)
            probs = probs / probs_sums[:, np.newaxis]
            return probs
```

```
x1 = np.arange(-4.0, 3.0, 0.05)
x2 = np.arange(-3.0, 4.0, 0.05)
x1,x2 = np.meshgrid(x1, x2)
y_pred = softmax_probs(np.c_[x1.ravel(), x2.ravel()], model)
extent = -4.0, 3.0, -3.0, 4.0

fig = plt.figure(figsize=(10,6))
plt.imshow(y_pred[:,0].reshape(x1.shape), cmap=cm.Reds, alpha=.4,
           interpolation='bilinear',extent = extent, origin='lower')
plt.imshow(y_pred[:,1].reshape(x1.shape), cmap=cm.Blues, alpha=.4,
           interpolation='bilinear',extent = extent, origin='lower')
plt.imshow(y_pred[:,2].reshape(x1.shape), cmap=cm.Greens, alpha=.4,
           interpolation='bilinear',extent = extent, origin='lower')
plt.scatter(X[ix0,0],X[ix0, 1],marker='o',color='red')
plt.scatter(X[ix1,0],X[ix1, 1],marker='x',color='blue')
plt.scatter(X[ix2,0],X[ix2, 1],marker='s',color='green')
plt.show()
```



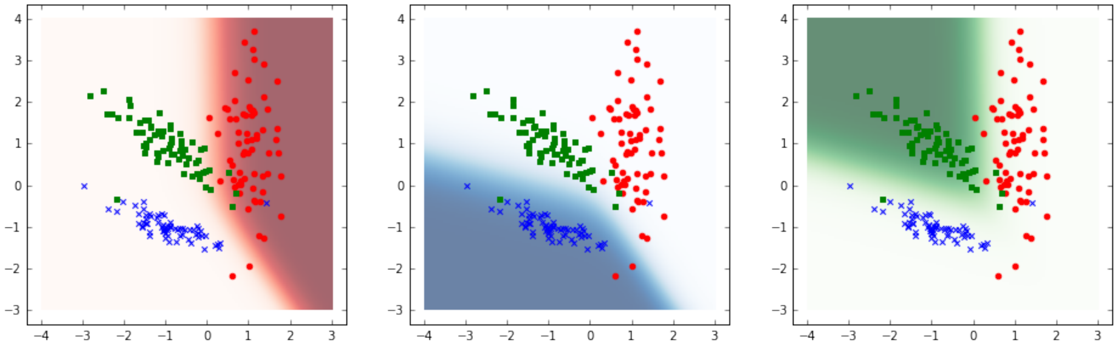
And another way to visualize it:

```
In [6]: fig = plt.figure(figsize=(16,8))
        fig.add_subplot(131)
        plt.imshow(y_pred[:,0].reshape(x1.shape), cmap=cm.Reds, alpha=.6,
                   interpolation='bilinear',extent = extent, origin='lower')
        plt.scatter(X[ix0,0],X[ix0, 1],marker='o',color='red')
        plt.scatter(X[ix1,0],X[ix1, 1],marker='x',color='blue')
        plt.scatter(X[ix2,0],X[ix2, 1],marker='s',color='green')

        fig.add_subplot(132)
        plt.imshow(y_pred[:,1].reshape(x1.shape), cmap=cm.Blues, alpha=.6,
                   interpolation='bilinear', extent = extent, origin='lower')
        plt.scatter(X[ix0,0],X[ix0, 1],marker='o',color='red')
        plt.scatter(X[ix1,0],X[ix1, 1],marker='x',color='blue')
        plt.scatter(X[ix2,0],X[ix2, 1],marker='s',color='green')

        fig.add_subplot(133)
        plt.imshow(y_pred[:,2].reshape(x1.shape), cmap=cm.Greens, alpha=.6,
                   interpolation='bilinear', extent = extent, origin='lower')
        plt.scatter(X[ix0,0],X[ix0, 1],marker='o',color='red')
        plt.scatter(X[ix1,0],X[ix1, 1],marker='x',color='blue')
        plt.scatter(X[ix2,0],X[ix2, 1],marker='s',color='green')

        plt.show()
```


4
3
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3