# HW4_report

April 4, 2016

# 1 CS189: Introduction to Machine Learning

## 1.1 Problem 1: Ridge Regression

In this problem we will return to predicting the median home value in a given Census area by extending linear regression. The data is in housing data.mat and it comes from ftp://rcom.univie.ac.at/mirrors/lib.stat.cmu.edu/datasets/.index.html . There are only 8 features for each data point; you can read about the features in housing data source.txt.

### 1.1.1 1a)

In order to find the optimizer of the loss function $J(w, \alpha)$, we need to differentiate the loss function $J(w, \alpha)$ and find the local minima. Before differentiating the loss function $J(w, \alpha)$, let us expand the mathematical expression to facilitate calcuations.

$$\begin{aligned} J(w, \alpha) &= (Xw + \alpha 1 - y)^\top (Xw + \alpha 1 - y) + \lambda w^\top w \\ &= (w^\top X^\top + \alpha 1^\top - y^\top)(Xw + \alpha 1 - y) + \lambda w^\top w \\ &= w^\top X^\top Xw + \alpha w^\top X^\top 1 - w^\top X^\top y + \alpha 1^\top Xw + \alpha^2 n - \alpha 1^\top y + -y^\top Xw - y^\top \alpha 1 + y^\top y + \lambda w^\top w \\ &= w^\top X^\top Xw - w^\top X^\top y + \alpha n - \alpha 1^\top y + -y^\top Xw - y^\top \alpha 1 + y^\top y \lambda w^\top w (\because\ X^\top 1 = 1^\top X = 0) \end{aligned}$$

If we differentiate the loss function $J(w, \alpha)$ by w, then

$$\frac{\partial J}{\partial w} = 2X^\top Xw - 2X^\top y + 2\lambda w.$$

Therefore,

$$\frac{\partial J}{\partial w} = 2X^\top Xw - 2X^\top y + 2\lambda w = 0$$

gives $\hat{w} = (X^\top X + \lambda I)^{-1} X^\top y$.

If we differentiate the loss function $J(w, \alpha)$ by $\alpha$, then we get

$$\frac{\partial J}{\partial \alpha} = 2\alpha n - 21^\top y.$$

In order for $\frac{\partial J}{\partial \alpha}$ to be 0, it should be that $\hat{\alpha} = \bar{y}$.

### 1.1.2 1b)

```
In [2]: import scipy.io as sio
        import numpy as np

        housing_data = sio.loadmat('./housing_dataset/housing_data.mat')
```

In [28]:
```
housing_xtrain = housing_data['Xtrain']
housing_ytrain = housing_data['Ytrain'][:,0]
housing_xvalid = housing_data['Xvalidate']
housing_yvalid = housing_data['Yvalidate'][:,0]
```

In [46]:
```
center_housing_xtrain.shape
```

Out[46]:
```
(19440, 8)
```

In [19]:
```
# center the feature values of the training and validation data.

center_housing_xtrain = housing_xtrain - np.expand_dims(np.mean(housing_xtrain, axis = 0), axis
center_housing_xvalid = housing_xvalid - np.expand_dims(np.mean(housing_xvalid, axis = 0), axis
```

In [23]:
```
np.sum(center_housing_xvalid, axis=0)
```

Out[23]:
```
array([  4.19220214e-13,  -2.70006240e-13,  -9.45874490e-11,
        -1.90993887e-11,  -8.36735126e-11,   2.91038305e-11,
         2.77111667e-13,  -2.55795385e-13])
```

In [5]:
```
from numpy.linalg import inv
```

In [64]:
```
# Implement ridge regression for (i)
def ridge_regression (X, y, lamda):
    w = np.dot(inv(np.dot(np.transpose(X), X) + (lamda * np.identity(8))),
               np.dot(np.transpose(X), y))
    alpha = np.mean(y)
    return [w, alpha]
```

In [6]:
```
# Implement 10-fold Cross-Validation, using the ridge_regression implemented above.


from sklearn.cross_validation import KFold
kf = KFold(19440, n_folds=10)

def tenfold_cv(lamda):
    RSS = 0
    for train_index, test_index in kf:
        X_train, X_test = center_housing_xtrain[train_index, :], center_housing_xtrain[test_inde
        y_train, y_test = housing_ytrain[train_index], housing_ytrain[test_index]
        ridge = ridge_regression(X_train, y_train, lamda)
        w = ridge[0]
        alpha = ridge[1]
        one = np.linspace(1,1,len(y_test))
        temp = (np.dot(X_test, w) + (alpha * np.transpose(one)) - y_test )
        RSS = RSS + np.dot(np.transpose(temp), temp)
    return RSS/10
```

In [88]:
```
tenfold_cv(1e-12)
```

Out[88]:
```
9425875042980.4492
```

In [89]:
```
tenfold_cv(1e-5)
```

Out[89]:
```
9425875042967.5977
```

```
In [90]: tenfold_cv(1e-4)

Out[90]: 9425875042851.9395

In [91]: tenfold_cv(1e-3)

Out[91]: 9425875041695.3809

In [92]: tenfold_cv(1e-2)

Out[92]: 9425875030137.1055

In [93]: tenfold_cv(1e-1)

Out[93]: 9425874915283.6348

In [94]: tenfold_cv(1)

Out[94]: 9425873839646.3105

In [95]: tenfold_cv(10)

Out[95]: 9425870341696.4805

In [96]: tenfold_cv(100)

Out[96]: 9426530937596.7383

In [97]: tenfold_cv(1000)

Out[97]: 9480399010571.4824

In [98]: tenfold_cv(5)

Out[98]: 9425870659168.0137

In [99]: tenfold_cv(20)

Out[99]: 9425881801097.7461

In [100]: tenfold_cv(12)

Out[100]: 9425871347405.1973

In [101]: tenfold_cv(8)

Out[101]: 9425869982405.9648

In [102]: tenfold_cv(7)

Out[102]: 9425870045690.5527

In [103]: tenfold_cv(9)

Out[103]: 9425870081144.416
```

Average RSS on different tuned lambda valuse is given as follows:

| Lambda | average RSS |
|---|---|
| 1e-12 | 9425875042980.4492 |
| 1e-5 | 9425875042967.5977 |
| 1e-4 | 9425875042851.9395 |
| 1e-3 | 9425875041695.3809 |
| 1e-2 | 9425875030137.1055 |
| 1e-1 | 9425874915283.6348 |
| 1 | 9425873839646.3105 |
| 5 | 9425870659168.0137 |
| 7 | 9425870045690.5527 |
| 8 | 9425869982405.9648 |
| 9 | 9425870081144.416 |
| 10 | 9425870341696.4805 |
| 12 | 9425871347405.1973 |
| 20 | 9425881801097.7461 |
| 100 | 9426530937596.7383 |
| 1000 | 9480399010571.4824 |

It is when $\lambda = 8$ that ridge regression with least squares achives the lowest average RSS throughout 10-fold cross validation.

```
In [109]: # RSS on validation data with the lambda 8 with the lowest cross-validation error.


          ridge = ridge_regression(center_housing_xtrain, housing_ytrain, 8)
          w = ridge[0]
          alpha = ridge[1]
          one = np.linspace(1,1,len(housing_yvalid))
          temp = (np.dot(center_housing_xvalid, w) + (alpha * np.transpose(one)) - housing_yvalid )
          RSS = np.dot(np.transpose(temp), temp)
```

```
In [110]: RSS
```

```
Out[110]: 5782552884504.541
```

When we train ridge regression with lambda of 8, RSS on validation data is given as 5782552884504.541. Compared to 5794953797654.9834, RSS of linear regression in HW3, the ridge regression exhibits lower RSS and therefore fits better to the validation data than the linear regression does.

```
In [111]: w
```

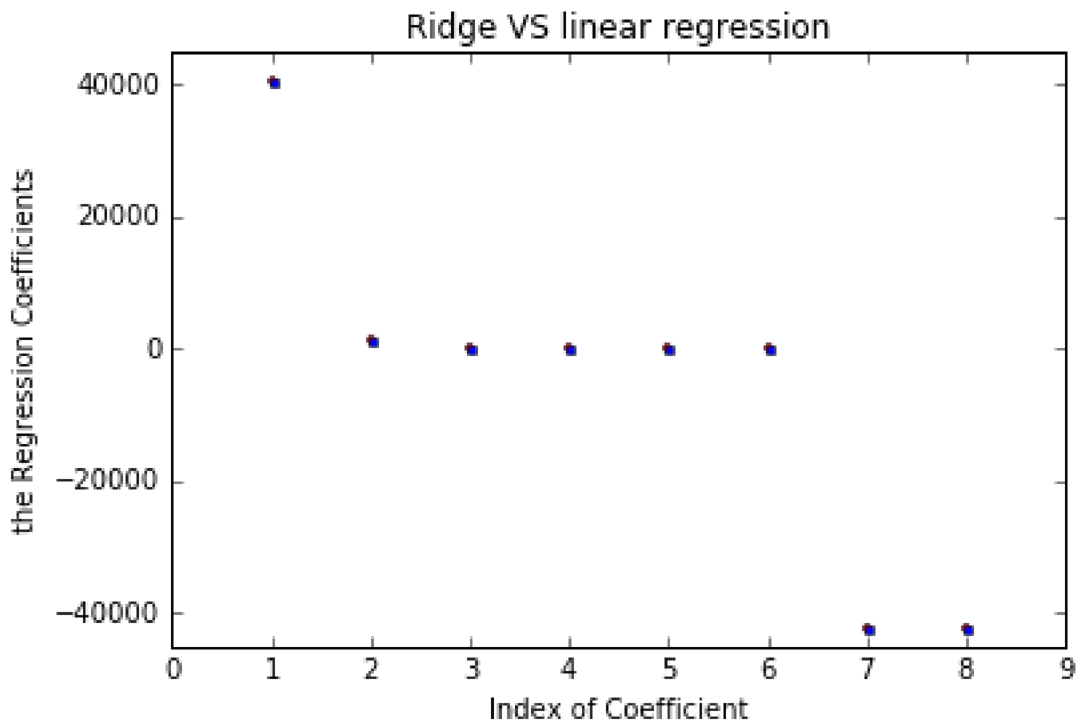
```
Out[111]: array([  4.05915089e+04,   1.19668415e+03,  -8.50719390e+00,
                   1.18261856e+02,  -3.77927882e+01,   4.32070327e+01,
                  -4.21178034e+04,  -4.23914057e+04])
```

```
In [115]: beta = np.array([ 4.05879986e+04, 1.19561189e+03, -8.50145688e+00,
          1.18352188e+02, -3.77900280e+01, 4.30562637e+01,
          -4.21794075e+04, -4.24573474e+04])
```

```
In [7]: import matplotlib.pyplot as plt
        %matplotlib inline
```

```
In [124]: xpos = np.arange(len(w))+ 1
          plt.axis([0,9, -45000, 45000])
          plt.plot(xpos, w, 'ro', ms = 3)
```

```
plt.plot(xpos, beta, 'bs', ms =3)
plt.xlabel('Index of Coefficient')
plt.ylabel('the Regression Coefficients')
plt.title('Ridge VS linear regression')
plt.show()
```




We plot the regression coefficents of 8 feature in both ridge regression and linear regression in HW3. This plot shows that the maxmimum absolute value of the regression coefficients is slightly smaller in the ridge regression than in the linear regression, as we could expected. Since we impose penalty on large norm of regression coefficient vector in ridge regression, the ridge regression tends to have smaller absolute value of regression coefficients.

### 1.2 Problem 2 : Logistic Regression

```
In [131]: X = np.array([[0,3,1],[1,3,1],[0,1,1],[1,1,1]])
          y = np.transpose(np.array([1,1,0,0]))
          1/(np.array([1,2,3]) + 1)

Out[131]: array([ 0.5       ,  0.33333333,  0.25      ])
```

```
In [8]: # Implement the logistic function
        def s(gamma):
            return 1/(1+ np.exp(- gamma))

        # Implement the logistic loss, aka the cross-entropy loss.
        def R(w,X,y):
            temp = np.dot(X,w)
            one = np.linspace(1,1, len(y))
```

```
        return np.dot(np.transpose(1- y),temp) + np.dot(one, np.log(1+ np.exp(-temp)))

        #return - np.dot(np.transpose(y) ,np.log(s(np.dot(X, w)))) - np.dot(np.transpose(1-y), np.l

    def batch_gradient(w,X,y,e):
        return w + e* np.dot(np.transpose(y- s(np.dot(X, w))), X)
```

```
In [266]: w0 = np.transpose(np.array([-2,1,0]))

In [269]: R(w0, X, y)

Out[269]: 1.9883724141284105

In [219]: mu0 = s(np.dot(X,w0))
          mu0

Out[219]: array([ 0.95257413,  0.73105858,  0.73105858,  0.26894142])

In [220]: w1 = batch_gradient(w0, X, y, 1)
          w1

Out[220]: array([-2.        ,  0.94910188, -0.68363271])

In [221]: mu1 = s(np.dot(X,w1))
          mu1

Out[221]: array([ 0.89693957,  0.54082713,  0.56598026,  0.15000896])

In [222]: w2 = batch_gradient(w1, X, y, 1)
          w2

Out[222]: array([-1.69083609,  1.91981257, -0.83738862])

In [271]: R(w2, X, y)

Out[271]: 1.8546997847922486
```

All things considered, * the value of $R(w_0)$ is 1.9883724141284105, * the value of $\mu_0$ is ( 0.95257413, 0.73105858, 0.73105858, 0.26894142), * the value of $w_1$ is $[-2., 0.94910188, -0.68363271]^\top$, * the value of $\mu_1$ is (0.89693957, 0.54082713, 0.56598026, 0.15000896), * the value of $w_2$ is $[-1.69083609, 1.91981257, -0.83738862]^\top$, * the value of $R(w_2)$ is 1.8546997847922486.

### 1.3 Problem 3: Spam classification using Logistic Regression

The spam dataset given as part of the homework in spam.mat consists of 5172 email messages, from which 32 features have been extracted. Please use the standard features for the first four parts of this problem. In part 5, we are asked to predict the labels of the test set in test.mat and submit the predictions to Kaggle. Feel free to use your own featurizes to boost up your score! One can imagine performing several kinds of preprocessing to this data matrix. Try each of the following separately:

1. Standardize each column to have mean 0 and unit variance.
2. Transform the features using $X_{ij} \mapsto \log(X_{ij} + 0.1)$, where the $X_{ij}$ 's are the entries of the design matrix.
3. Binarize the features using $X_{ij} \mapsto I(X_{ij} > 0)$. $I$ denotes an indicator variable.

```
In [21]: # import the data

         spam_data = sio.loadmat('./spam_dataset/spam_data.mat')
         sp_test = spam_data['test_data']
         sp_xtrain = spam_data['training_data']
         sp_ytrain = spam_data['training_labels'][0,]
         sp_test = np.hstack((sp_test, np.expand_dims(np.transpose(np.linspace(1,1, sp_test.shape[0])),
```

**Preprocessing**

- Standardize each column to have mean 0 and unit variance.

```
In [22]: # Standardize each column to have mean 0 and unit variance.
         col_mean = np.mean(sp_xtrain, axis=0)
         col_std = np.std(sp_xtrain, axis = 0)

         sp_xtrain1 = (sp_xtrain - np.expand_dims(col_mean,axis = 0))/np.expand_dims(col_std, axis = 0)
         sp_xtrain1 = np.hstack((sp_xtrain1, np.expand_dims(np.transpose(np.linspace(1,1, sp_xtrain1.sha
```

- Transform the features using $X_{ij} \mapsto \log(X_{ij} + 0.1)$, where the $X_{ij}$ 's are the entries of the design matrix.

```
In [23]: sp_xtrain2 = np.log(sp_xtrain + 0.1)
         sp_xtrain2 = np.hstack((sp_xtrain2, np.expand_dims(np.transpose(np.linspace(1,1, sp_xtrain2.sha
```

- Binarize the features using $X_{ij} \mapsto I(X_{ij} > 0)$. $I$ denotes an indicator variable.

```
In [24]: sp_xtrain3 = (sp_xtrain > 0) * 1
         sp_xtrain3 = np.hstack((sp_xtrain3, np.expand_dims(np.transpose(np.linspace(1,1, sp_xtrain3.sha
```

**Batch Gradient Descent** We run the batch gradient method with learning rate $10^{-3}$ for the first pre-processing data.

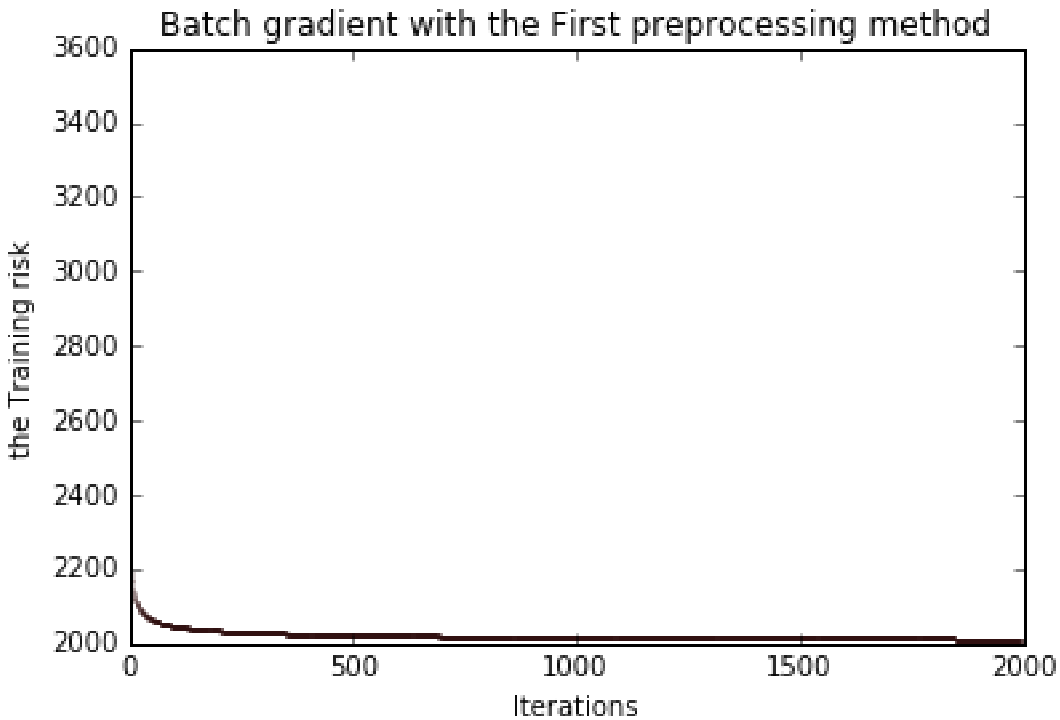
```
In [25]: w_0 = np.transpose(np.zeros(sp_xtrain1.shape[1]))

In [26]: def logistic_with_batch_gradient(w,X,y,e,n):
             w_temp = w
             risk = []
             for i in np.arange(n):
                 risk_temp = R(w_temp,X,y)
                 print ('the risk is ', risk_temp ,': iteration ', (i+1))
                 w_temp = batch_gradient(w_temp, X, y, e)
                 risk = risk + [risk_temp]
             return [risk, w_temp]


In [27]: empirical_risk1 = logistic_with_batch_gradient(w_0, sp_xtrain1, sp_ytrain, 1e-3, 2000)
```

```
the risk is  3584.95721786 : iteration  1
the risk is  2387.22054291 : iteration  2
the risk is  2228.25903423 : iteration  3
the risk is  2202.00609786 : iteration  4
the risk is  2184.67717818 : iteration  5
the risk is  2171.56129252 : iteration  6
the risk is  2161.02106254 : iteration  7
the risk is  2152.25131949 : iteration  8
the risk is  2144.779402 : iteration  9
the risk is  2138.29922082 : iteration  10
the risk is  2132.59971606 : iteration  11
the risk is  2127.52901278 : iteration  12
the risk is  2122.97444877 : iteration  13
the risk is  2118.85047442 : iteration  14
the risk is  2115.09083712 : iteration  15
the risk is  2111.64331447 : iteration  16
```

```
the risk is  2009.00718748 : iteration  1961
the risk is  2009.00448491 : iteration  1962
the risk is  2009.00178461 : iteration  1963
the risk is  2008.99908658 : iteration  1964
the risk is  2008.99639081 : iteration  1965
the risk is  2008.99369729 : iteration  1966
the risk is  2008.99100603 : iteration  1967
the risk is  2008.98831702 : iteration  1968
the risk is  2008.98563026 : iteration  1969
the risk is  2008.98294574 : iteration  1970
the risk is  2008.98026347 : iteration  1971
the risk is  2008.97758344 : iteration  1972
the risk is  2008.97490565 : iteration  1973
the risk is  2008.97223009 : iteration  1974
the risk is  2008.96955677 : iteration  1975
the risk is  2008.96688567 : iteration  1976
the risk is  2008.96421681 : iteration  1977
the risk is  2008.96155017 : iteration  1978
the risk is  2008.95888574 : iteration  1979
the risk is  2008.95622354 : iteration  1980
the risk is  2008.95356356 : iteration  1981
the risk is  2008.95090578 : iteration  1982
the risk is  2008.94825022 : iteration  1983
the risk is  2008.94559687 : iteration  1984
the risk is  2008.94294572 : iteration  1985
the risk is  2008.94029677 : iteration  1986
the risk is  2008.93765003 : iteration  1987
the risk is  2008.93500548 : iteration  1988
the risk is  2008.93236312 : iteration  1989
the risk is  2008.92972296 : iteration  1990
the risk is  2008.92708499 : iteration  1991
the risk is  2008.9244492 : iteration  1992
the risk is  2008.9218156 : iteration  1993
the risk is  2008.91918418 : iteration  1994
the risk is  2008.91655493 : iteration  1995
the risk is  2008.91392786 : iteration  1996
the risk is  2008.91130297 : iteration  1997
the risk is  2008.90868024 : iteration  1998
the risk is  2008.90605969 : iteration  1999
the risk is  2008.9034413 : iteration  2000
```

2000 iterations lower the trainig risk down to 2008.9034413. We plot the training risk (the empirical risk of the training set) vs. the number of iterations.

```
In [28]: xpos = np.arange(len(empirical_risk1[0]))+ 1
         plt.plot(xpos, empirical_risk1[0], 'ro', ms =0.4)
         plt.xlabel('Iterations')
         plt.ylabel('the Training risk')
         plt.title('Batch gradient with the First preprocessing method')
         plt.show()
```




We run the batch gradient method with learning rate $10^5$ for the second preprocessing data.

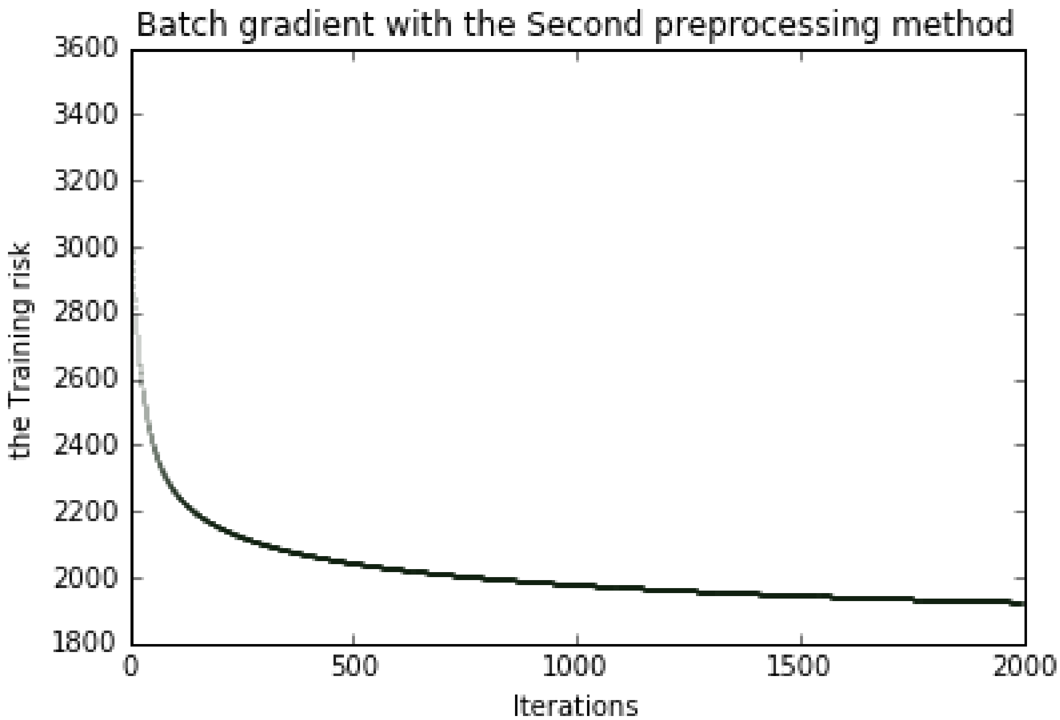
```
In [29]: empirical_risk2 = logistic_with_batch_gradient(w_0, sp_xtrain2, sp_ytrain, 1e-5, 2000)
```

```
the risk is  3584.95721786 : iteration  1
the risk is  3297.42549882 : iteration  2
the risk is  3062.14542981 : iteration  3
the risk is  3017.58606369 : iteration  4
the risk is  2983.2229002 : iteration  5
the risk is  2954.59019699 : iteration  6
the risk is  2928.11948285 : iteration  7
the risk is  2903.12739095 : iteration  8
the risk is  2879.32569929 : iteration  9
the risk is  2856.59869658 : iteration  10
the risk is  2834.87595794 : iteration  11
the risk is  2814.10140317 : iteration  12
the risk is  2794.22548882 : iteration  13
the risk is  2775.20178215 : iteration  14
the risk is  2756.98658009 : iteration  15
the risk is  2739.53842088 : iteration  16
the risk is  2722.81804465 : iteration  17
the risk is  2706.78824526 : iteration  18
the risk is  2691.41380139 : iteration  19
the risk is  2676.66137714 : iteration  20
the risk is  2662.49943896 : iteration  21
the risk is  2648.8981665 : iteration  22
the risk is  2635.8293684 : iteration  23
the risk is  2623.26639864 : iteration  24
```

```
the risk is  1923.50301295 : iteration  1969
the risk is  1923.46869388 : iteration  1970
the risk is  1923.43439948 : iteration  1971
the risk is  1923.40012972 : iteration  1972
the risk is  1923.36588459 : iteration  1973
the risk is  1923.33166404 : iteration  1974
the risk is  1923.29746805 : iteration  1975
the risk is  1923.2632966 : iteration  1976
the risk is  1923.22914966 : iteration  1977
the risk is  1923.1950272 : iteration  1978
the risk is  1923.16092919 : iteration  1979
the risk is  1923.1268556 : iteration  1980
the risk is  1923.09280641 : iteration  1981
the risk is  1923.05878159 : iteration  1982
the risk is  1923.02478111 : iteration  1983
the risk is  1922.99080495 : iteration  1984
the risk is  1922.95685308 : iteration  1985
the risk is  1922.92292547 : iteration  1986
the risk is  1922.88902209 : iteration  1987
the risk is  1922.85514292 : iteration  1988
the risk is  1922.82128793 : iteration  1989
the risk is  1922.78745709 : iteration  1990
the risk is  1922.75365038 : iteration  1991
the risk is  1922.71986777 : iteration  1992
the risk is  1922.68610923 : iteration  1993
the risk is  1922.65237474 : iteration  1994
the risk is  1922.61866426 : iteration  1995
the risk is  1922.58497778 : iteration  1996
the risk is  1922.55131526 : iteration  1997
the risk is  1922.51767668 : iteration  1998
the risk is  1922.48406202 : iteration  1999
the risk is  1922.45047124 : iteration  2000
```

After 2000 iterations, the training risk decreases down to 1922.45047124, which is lower than what the first preprocessing method achieved. We plot the training risk over the number of iterations.

```
In [30]: xpos = np.arange(len(empirical_risk2[0]))+ 1
         plt.plot(xpos, empirical_risk2[0], 'go', ms = 0.4)
         plt.xlabel('Iterations')
         plt.ylabel('the Training risk')
         plt.title('Batch gradient with the Second preprocessing method')
         plt.show()
```




We run the batch gradient method with learning rate $2 * 10^3$ for the third preprocessing data.

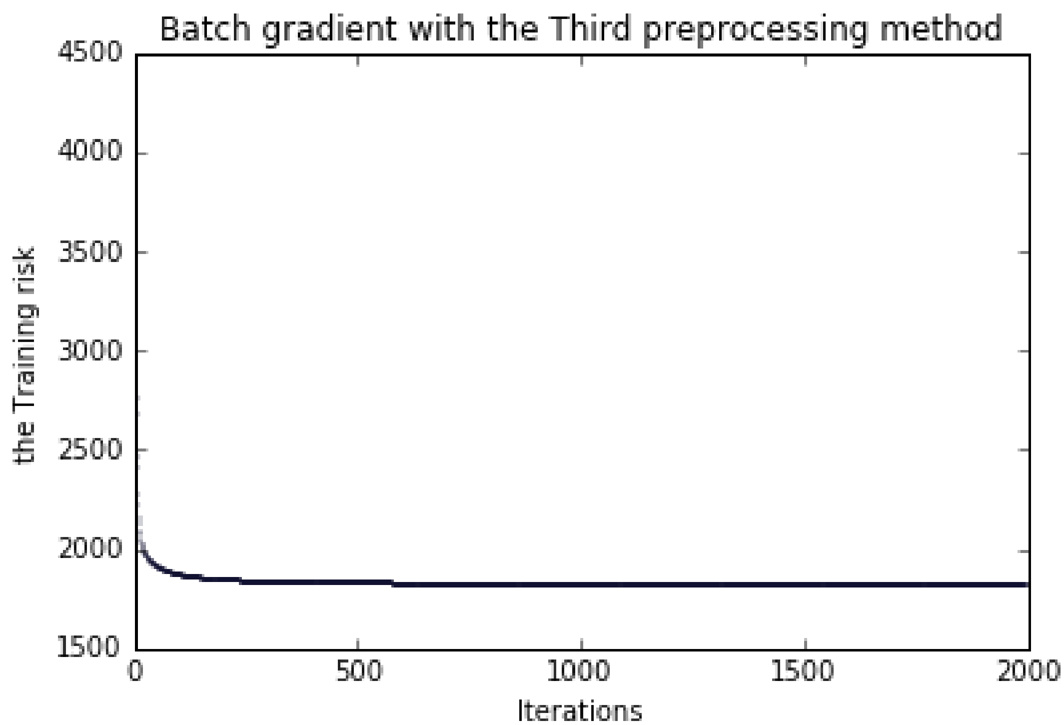
```
In [31]: empirical_risk3 = logistic_with_batch_gradient(w_0, sp_xtrain3, sp_ytrain, 2e-3, 2000)
```

```
the risk is  3584.95721786 : iteration  1
the risk is  4106.32058967 : iteration  2
the risk is  3197.9220317 : iteration  3
the risk is  3967.10472524 : iteration  4
the risk is  2682.77707914 : iteration  5
the risk is  2754.18057516 : iteration  6
the risk is  2463.95894291 : iteration  7
the risk is  2417.07294157 : iteration  8
the risk is  2271.68299745 : iteration  9
the risk is  2223.3775302 : iteration  10
the risk is  2155.00067837 : iteration  11
the risk is  2123.86911517 : iteration  12
the risk is  2089.30249473 : iteration  13
the risk is  2069.74877244 : iteration  14
the risk is  2050.14159893 : iteration  15
the risk is  2037.07824844 : iteration  16
the risk is  2024.64782528 : iteration  17
the risk is  2015.31136188 : iteration  18
the risk is  2006.66649052 : iteration  19
the risk is  1999.60404597 : iteration  20
the risk is  1993.12895398 : iteration  21
the risk is  1987.52734896 : iteration  22
the risk is  1982.38003115 : iteration  23
the risk is  1977.75244675 : iteration  24
```

```
the risk is  1820.99212377 : iteration  1969
the risk is  1820.99105226 : iteration  1970
the risk is  1820.98998186 : iteration  1971
the risk is  1820.98891256 : iteration  1972
the risk is  1820.98784437 : iteration  1973
the risk is  1820.98677729 : iteration  1974
the risk is  1820.9857113 : iteration  1975
the risk is  1820.98464642 : iteration  1976
the risk is  1820.98358263 : iteration  1977
the risk is  1820.98251994 : iteration  1978
the risk is  1820.98145834 : iteration  1979
the risk is  1820.98039784 : iteration  1980
the risk is  1820.97933843 : iteration  1981
the risk is  1820.97828011 : iteration  1982
the risk is  1820.97722287 : iteration  1983
the risk is  1820.97616673 : iteration  1984
the risk is  1820.97511166 : iteration  1985
the risk is  1820.97405768 : iteration  1986
the risk is  1820.97300478 : iteration  1987
the risk is  1820.97195296 : iteration  1988
the risk is  1820.97090222 : iteration  1989
the risk is  1820.96985255 : iteration  1990
the risk is  1820.96880396 : iteration  1991
the risk is  1820.96775644 : iteration  1992
the risk is  1820.96670999 : iteration  1993
the risk is  1820.96566461 : iteration  1994
the risk is  1820.9646203 : iteration  1995
the risk is  1820.96357705 : iteration  1996
the risk is  1820.96253487 : iteration  1997
the risk is  1820.96149375 : iteration  1998
the risk is  1820.96045369 : iteration  1999
the risk is  1820.95941468 : iteration  2000
```

After 2000 iterations, the training risk decreases down to 1820.95941468, which is the lowest among the three preprocessing method. We plot the training risk over the number of iterations.

```
In [32]: xpos = np.arange(len(empirical_risk3[0]))+ 1
         plt.plot(xpos, empirical_risk3[0], 'bo', ms =0.4)
         plt.xlabel('Iterations')
         plt.ylabel('the Training risk')
         plt.title('Batch gradient with the Third preprocessing method')
         plt.show()
```




**Stochastic Gradient Descent**

```
In [3]: import numpy.random as nr
```

```
In [342]: def logistic_with_stochastic_gradient(w,X,y,e,n):
              w_temp = w
              risk = []
              nr.seed(0)
              indice = nr.choice(X.shape[0], n, replace =True)
              for i in np.arange(n):
                  risk_temp = R(w_temp,X,y)
                  j = indice[i]
                  print ('the risk is ', risk_temp ,': iteration ', (i+1))
                  w_temp = w_temp + e* (y[j]- s(np.dot(X[j,], w_temp)))* X[j,]
                  risk = risk + [risk_temp]
              return risk
```

We run the stochastic gradient descent with learning rate $10^{-2}$ on the first preprocessed data.

```
In [349]: training_risk1 = logistic_with_stochastic_gradient(w_0,sp_xtrain1,sp_ytrain,1e-2,3000)
```

```
the risk is  3584.95721786 : iteration  1
the risk is  3577.18569409 : iteration  2
the risk is  3569.54942435 : iteration  3
the risk is  3557.08974585 : iteration  4
the risk is  3549.43081846 : iteration  5
the risk is  3545.22638536 : iteration  6
the risk is  3544.27684629 : iteration  7
```

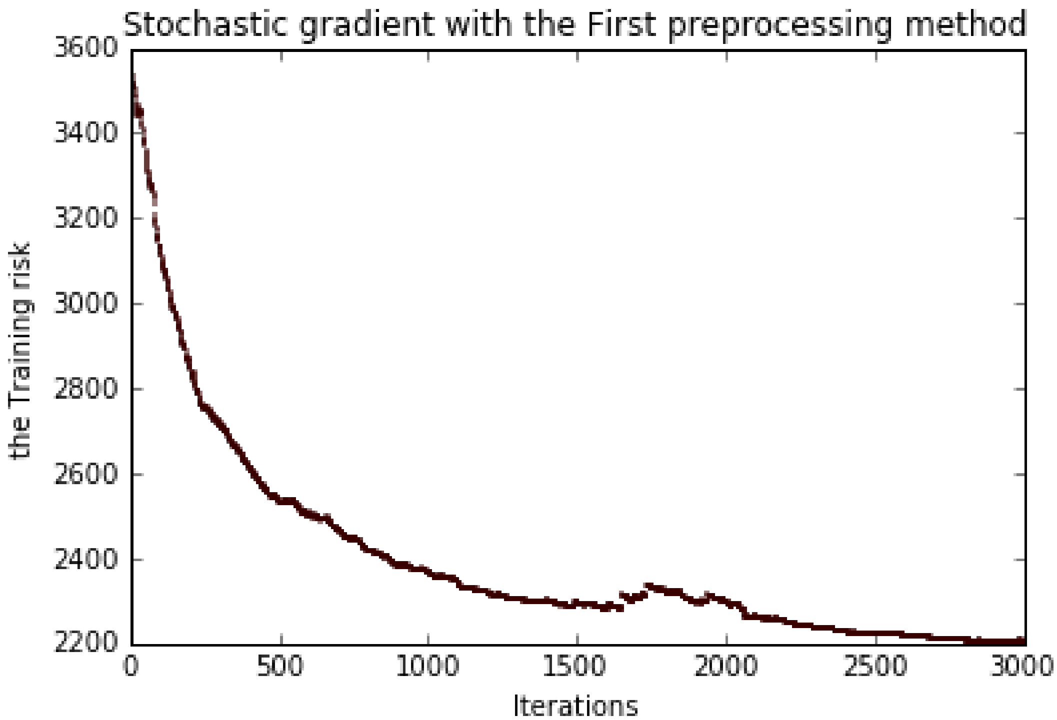
```
the risk is  2207.56654746 : iteration  2978
the risk is  2207.32699551 : iteration  2979
the risk is  2207.06153589 : iteration  2980
the risk is  2207.69725855 : iteration  2981
the risk is  2207.68775331 : iteration  2982
the risk is  2207.72057017 : iteration  2983
the risk is  2207.74726163 : iteration  2984
the risk is  2207.72799554 : iteration  2985
the risk is  2207.66483498 : iteration  2986
the risk is  2207.39742025 : iteration  2987
the risk is  2207.60431926 : iteration  2988
the risk is  2207.34799286 : iteration  2989
the risk is  2208.38634599 : iteration  2990
the risk is  2208.41171998 : iteration  2991
the risk is  2208.17888505 : iteration  2992
the risk is  2207.96174752 : iteration  2993
the risk is  2207.75378217 : iteration  2994
the risk is  2207.50733708 : iteration  2995
the risk is  2207.31335389 : iteration  2996
the risk is  2207.14605318 : iteration  2997
the risk is  2206.99784651 : iteration  2998
the risk is  2206.99700984 : iteration  2999
the risk is  2206.8646236 : iteration  3000
```

```
In [380]: np.amin(training_risk1)
```

```
Out[380]: 2201.0702252575302
```

The training risk decreases down to 2206.8646236 after 3000 iterations and the smallest risk we obtained is 2201.0702252575302. We plot the training risk over the number of iterations.

```
In [379]: xpos = np.arange(len(training_risk1))+ 1
          plt.plot(xpos, training_risk1, 'ro', ms =1)
          plt.xlabel('Iterations')
          plt.ylabel('the Training risk')
          plt.title('Stochastic gradient with the First preprocessing method')
          plt.show()
```




We run stochastic gradient descent on the second preprocessed data with learning rate $6 * 10^{-3}$.

```
In [374]: training_risk2 = logistic_with_stochastic_gradient(w_0,sp_xtrain2,sp_ytrain,6e-3,3000)
```

```
the risk is  3584.95721786 : iteration  1
the risk is  3212.15851585 : iteration  2
the risk is  3103.95591907 : iteration  3
the risk is  3100.14270579 : iteration  4
the risk is  3162.06581537 : iteration  5
the risk is  3254.15187477 : iteration  6
the risk is  3352.29449397 : iteration  7
the risk is  3062.88300343 : iteration  8
the risk is  3103.98918932 : iteration  9
the risk is  3166.53726792 : iteration  10
the risk is  3075.5836251 : iteration  11
the risk is  3040.81037176 : iteration  12
the risk is  3075.65130709 : iteration  13
the risk is  3152.06946332 : iteration  14
the risk is  3244.78102454 : iteration  15
the risk is  3335.65849333 : iteration  16
the risk is  3034.76868602 : iteration  17
the risk is  3063.65335317 : iteration  18
the risk is  3125.60909426 : iteration  19
the risk is  3203.31187791 : iteration  20
the risk is  3315.23172421 : iteration  21
the risk is  3420.23836054 : iteration  22
the risk is  3515.86146489 : iteration  23
the risk is  3607.81114933 : iteration  24
```

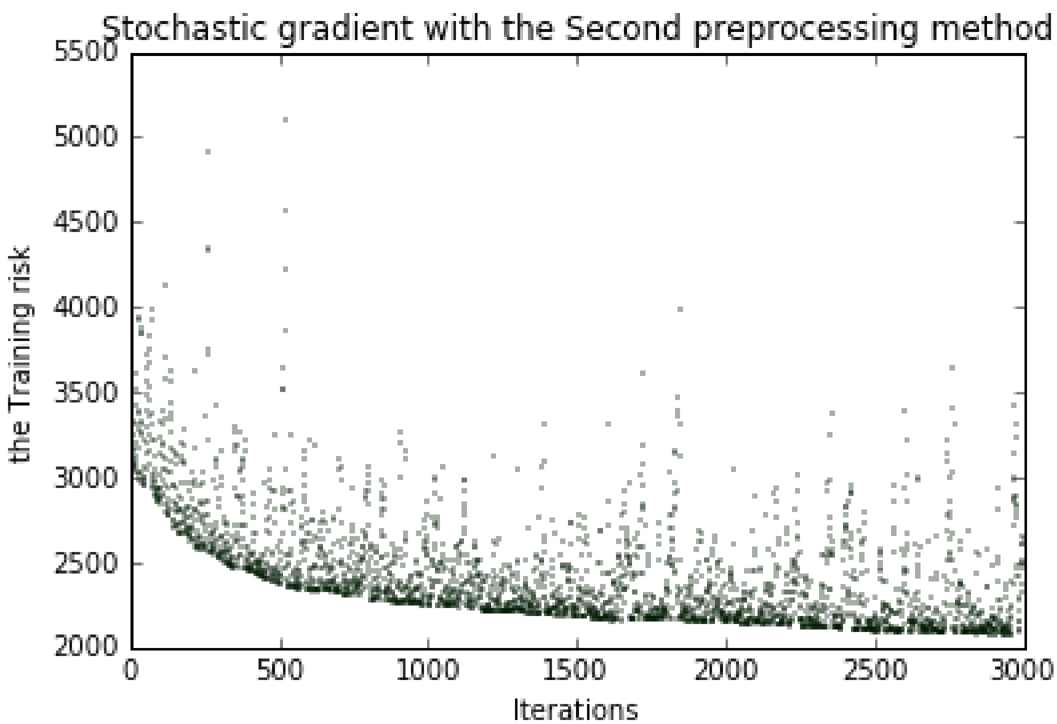
```
the risk is  2498.91498185 : iteration  2995
the risk is  2520.87270224 : iteration  2996
the risk is  2604.12963914 : iteration  2997
the risk is  2651.88789088 : iteration  2998
the risk is  2604.4552991 : iteration  2999
the risk is  2642.34550665 : iteration  3000
```

In [376]:
```
np.amin(training_risk2)
```

Out[376]: 2080.0943692828496

After 3000 iteration, the minimum of the training risk we obtained is 2080.0943692828496. We plot the training risk over the number of iterations.

In [378]:
```
xpos = np.arange(len(training_risk2))+ 1
plt.plot(xpos, training_risk2, 'go', ms =0.7)
plt.xlabel('Iterations')
plt.ylabel('the Training risk')
plt.title('Stochastic gradient with the Second preprocessing method')
plt.show()
```




We run the stochastic gradient descent with learning rate 0.25 on the third processed data.

In [390]:
```
training_risk3 = logistic_with_stochastic_gradient(w_0,sp_xtrain3,sp_ytrain,0.25,3000)
```

```
the risk is  3584.95721786 : iteration  1
the risk is  3394.71136248 : iteration  2
the risk is  3263.56439499 : iteration  3
```

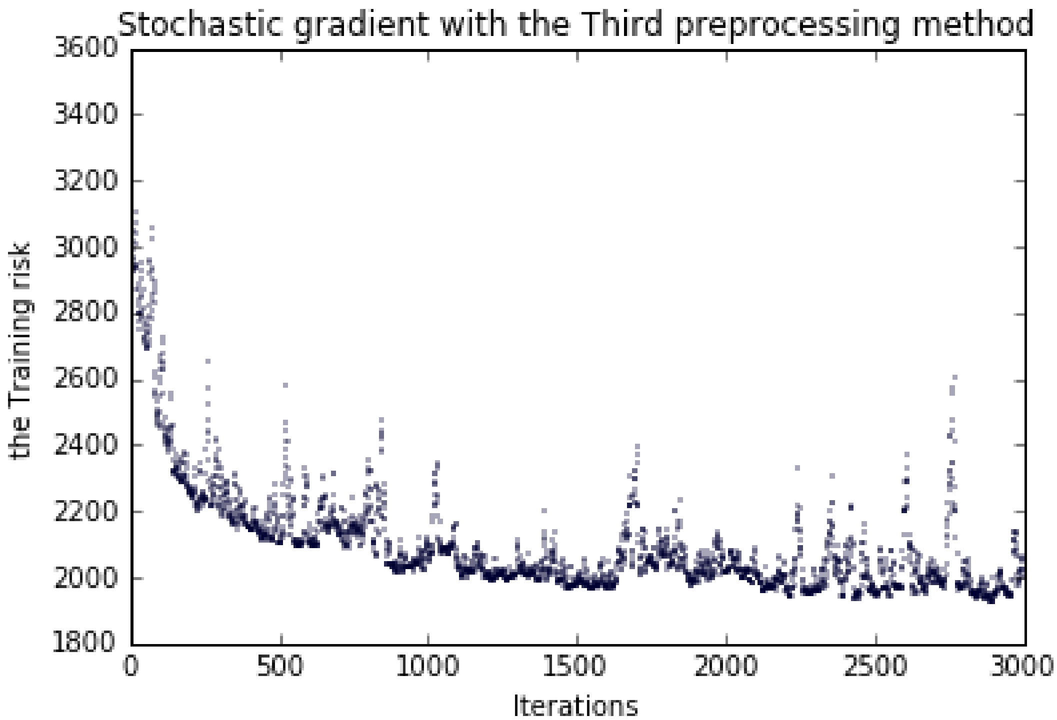
```
the risk is  2105.73797256 : iteration  2974
the risk is  2134.69381042 : iteration  2975
the risk is  2008.54715556 : iteration  2976
the risk is  1979.28252688 : iteration  2977
the risk is  2086.98264892 : iteration  2978
the risk is  2076.96616593 : iteration  2979
the risk is  2066.02879129 : iteration  2980
the risk is  2127.56851354 : iteration  2981
the risk is  2135.11036447 : iteration  2982
the risk is  2064.00367304 : iteration  2983
the risk is  2018.88532541 : iteration  2984
the risk is  2018.86484439 : iteration  2985
the risk is  2022.29227704 : iteration  2986
the risk is  1993.09866324 : iteration  2987
the risk is  1990.51389123 : iteration  2988
the risk is  1974.20998364 : iteration  2989
the risk is  1985.98964033 : iteration  2990
the risk is  1995.15919228 : iteration  2991
the risk is  2000.89313779 : iteration  2992
the risk is  2007.40622337 : iteration  2993
the risk is  2022.65532272 : iteration  2994
the risk is  2025.03936747 : iteration  2995
the risk is  2027.44919338 : iteration  2996
the risk is  2046.51990959 : iteration  2997
the risk is  2055.39792415 : iteration  2998
the risk is  2055.31713203 : iteration  2999
the risk is  2065.03700687 : iteration  3000
```

```
In [391]: np.amin(training_risk3)
```

```
Out[391]: 1928.7904915537711
```

After 3000 iterations, the training risk decreases down to 2065.03700687 and the smallest risk we obtained is 1928.7904915537711. We plot the training risk over the number of iterations.

```
In [392]: xpos = np.arange(len(training_risk3))+ 1
          plt.plot(xpos, training_risk3, 'bo', ms =0.7)
          plt.xlabel('Iterations')
          plt.ylabel('the Training risk')
          plt.title('Stochastic gradient with the Third preprocessing method')
          plt.show()
```




**Stochastic gradient descent VS Batch gradient descent** It appears that the convergence rate, the speed in which a method converge to the minima, is higher in the batch gradient descent and the batch gradient method guarantees continuous decrease in the training risk. However, the stochastic gradient descent takes shorter time to compute for each step.

**Stochastic gradient descent with learning rate decreasing over iterations**

```
In [407]: def logistic_with_stochastic_gradient_decreasing_stepsize(w,X,y,e,n):
              w_temp = w
              risk = []
              nr.seed(0)
              indice = nr.choice(X.shape[0], n, replace =True)
              for i in np.arange(n):
                  risk_temp = R(w_temp,X,y)
                  j = indice[i]
                  print ('the risk is ', risk_temp ,': iteration ', (i+1))
                  w_temp = w_temp + (e/(i+1)) * (y[j]- s(np.dot(X[j,], w_temp)))* X[j,]
                  risk = risk + [risk_temp]
              return risk
```

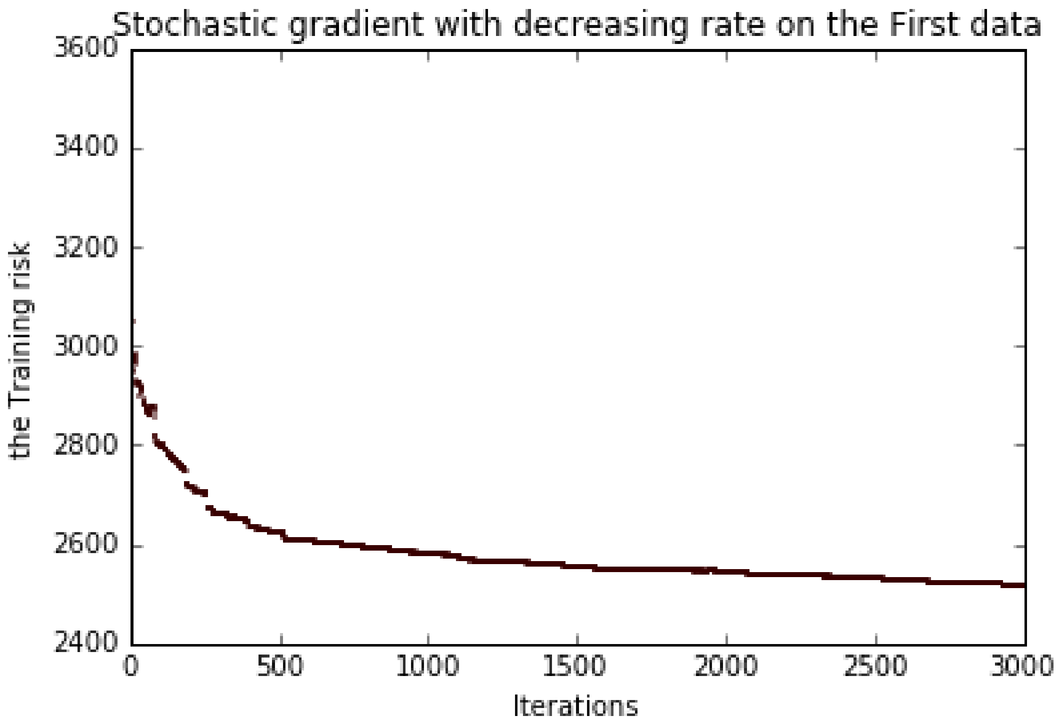
We run the stochastic gradient descent with learning rate $\frac{1}{t}$ at step $t$ on the first preprocessed data.

```
In [434]: training_risk_1 =logistic_with_stochastic_gradient_decreasing_stepsize(w_0,sp_xtrain1,sp_ytra

the risk is  3584.95721786 : iteration  1
the risk is  3252.40655651 : iteration  2
the risk is  3292.63860538 : iteration  3
the risk is  3085.89503634 : iteration  4
```

```
the risk is  2518.31359094 : iteration  2975
the risk is  2518.32229277 : iteration  2976
the risk is  2518.3229979 : iteration  2977
the risk is  2518.38436352 : iteration  2978
the risk is  2518.36244733 : iteration  2979
the risk is  2518.32153486 : iteration  2980
the risk is  2518.36607779 : iteration  2981
the risk is  2518.34297763 : iteration  2982
the risk is  2518.34759356 : iteration  2983
the risk is  2518.32727516 : iteration  2984
the risk is  2518.32661548 : iteration  2985
the risk is  2518.12750563 : iteration  2986
the risk is  2518.10939783 : iteration  2987
the risk is  2518.14073466 : iteration  2988
the risk is  2518.12265615 : iteration  2989
the risk is  2518.24819988 : iteration  2990
the risk is  2518.24551491 : iteration  2991
the risk is  2518.23155956 : iteration  2992
the risk is  2518.21762553 : iteration  2993
the risk is  2518.19963968 : iteration  2994
the risk is  2518.15902748 : iteration  2995
the risk is  2518.13735112 : iteration  2996
the risk is  2518.11944996 : iteration  2997
the risk is  2518.10149039 : iteration  2998
the risk is  2518.09542587 : iteration  2999
the risk is  2518.08163549 : iteration  3000
```

After 3000 iterations, the training risk decreases down to 2518.08163549. We plot the training risk over the number of iterations.

```
In [435]: xpos = np.arange(len(training_risk_1))+ 1
          plt.plot(xpos, training_risk_1, 'ro', ms =1)
          plt.xlabel('Iterations')
          plt.ylabel('the Training risk')
          plt.title('Stochastic gradient with decreasing rate on the First data')
          plt.show()
```




We run the stochastic gradient descent with the decreasing step size $\frac{1.1}{t}$ at step $t$.

```
In [455]: training_risk_2 =logistic_with_stochastic_gradient_decreasing_stepsize(w_0,sp_xtrain2,sp_ytra
```

```
the risk is  3584.95721786 : iteration  1
the risk is  124707.212082 : iteration  2
the risk is  124707.212082 : iteration  3
the risk is  124707.212082 : iteration  4
the risk is  124707.212082 : iteration  5
the risk is  124707.212082 : iteration  6
the risk is  124707.212082 : iteration  7
the risk is  91134.397048 : iteration  8
the risk is  91134.397048 : iteration  9
the risk is  91134.397048 : iteration  10
the risk is  65899.9551046 : iteration  11
the risk is  65899.9551046 : iteration  12
the risk is  65899.9551046 : iteration  13
the risk is  65899.9551046 : iteration  14
the risk is  65899.9551046 : iteration  15
the risk is  65899.9551046 : iteration  16
the risk is  49957.5734774 : iteration  17
the risk is  49957.5734774 : iteration  18
the risk is  49957.5734774 : iteration  19
the risk is  49957.5734774 : iteration  20
the risk is  49957.5734774 : iteration  21
the risk is  49957.5734774 : iteration  22
the risk is  49957.5734774 : iteration  23
the risk is  49957.5734774 : iteration  24
```

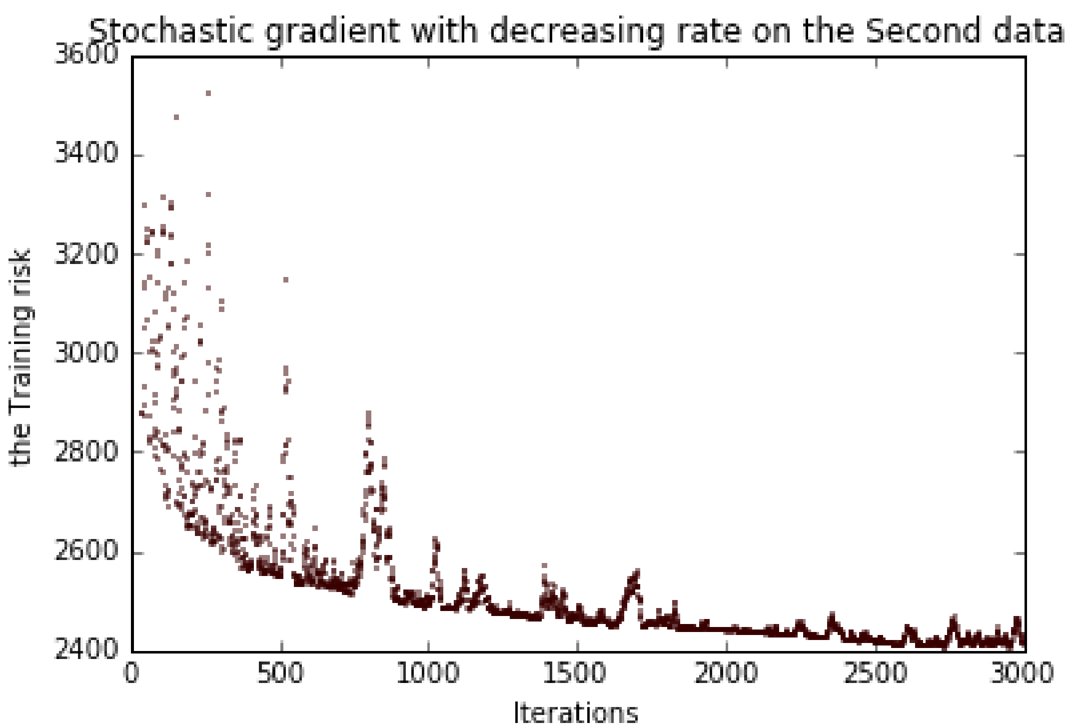
```
the risk is  2421.23057353 : iteration  2995
the risk is  2418.74254936 : iteration  2996
the risk is  2415.20851453 : iteration  2997
the risk is  2412.41764359 : iteration  2998
the risk is  2412.59857607 : iteration  2999
the risk is  2411.97082038 : iteration  3000
```

In [457]: 
```
np.amin(training_risk_2)
```

Out[457]: 2404.8189851332536

After 3000 iterations, the training risk decreases down to 2411.97082038 and the smallest risk we obtained is 2404.8189851332536. We plot the training risk over the number of iterations.

In [460]: 
```
xpos = np.arange(len(training_risk_2))+ 1

plt.plot(xpos, training_risk_2, 'ro', ms =1)
plt.xlabel('Iterations')
plt.ylabel('the Training risk')
plt.title('Stochastic gradient with decreasing rate on the Second data')
x1,x2,y1,y2 = plt.axis()
plt.axis((x1,x2,2400,3600))
plt.show()
```




We run the stochastic gradient descent with decreasing learning rate $\frac{8}{t}$ at step $t$ on the third preprocessed data.

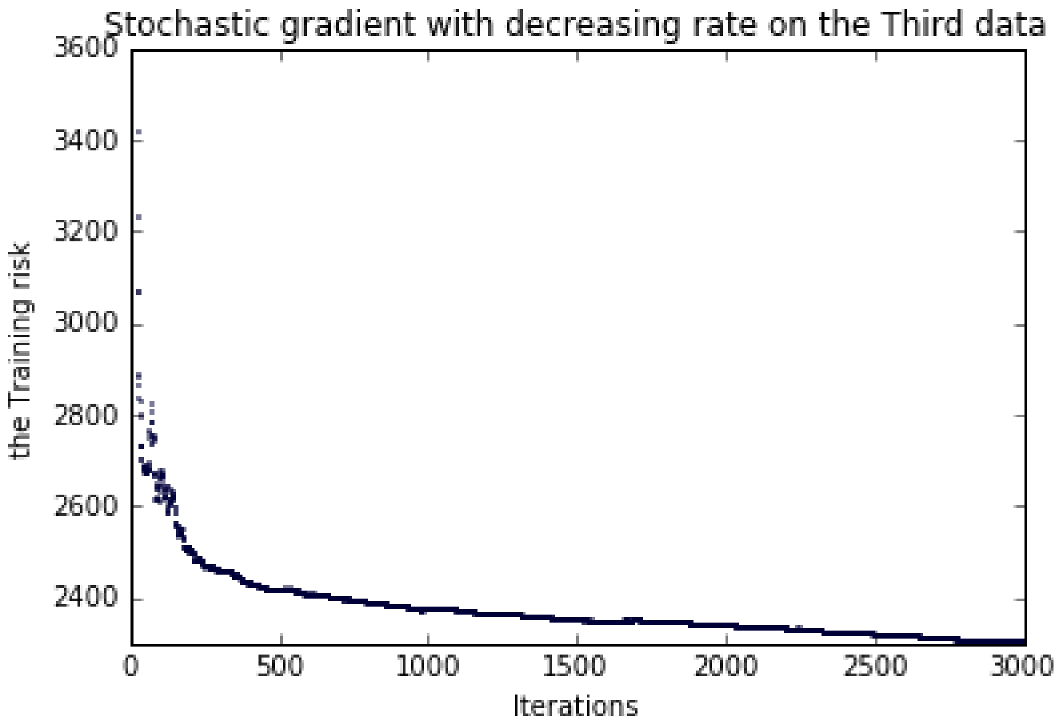
In [473]: 
```
training_risk_3 =logistic_with_stochastic_gradient_decreasing_stepsize(w_0,sp_xtrain3,sp_ytra
```

```
the risk is  3584.95721786 : iteration  1
the risk is  7630.11768245 : iteration  2
the risk is  7632.57883964 : iteration  3
the risk is  7634.56358585 : iteration  4
the risk is  7686.37778253 : iteration  5
the risk is  7754.55048236 : iteration  6
the risk is  7755.60365454 : iteration  7
the risk is  4867.83373023 : iteration  8
the risk is  5108.06361153 : iteration  9
the risk is  5109.29759209 : iteration  10
the risk is  3928.76723755 : iteration  11
the risk is  3993.73069164 : iteration  12
the risk is  3998.25388454 : iteration  13
the risk is  4046.87853921 : iteration  14
the risk is  4056.59256502 : iteration  15
the risk is  4057.69071289 : iteration  16
the risk is  3638.55055255 : iteration  17
the risk is  3639.86729894 : iteration  18
the risk is  3675.36510593 : iteration  19
the risk is  3676.41449269 : iteration  20
the risk is  4122.11430793 : iteration  21
the risk is  4200.39539916 : iteration  22
the risk is  4232.40003714 : iteration  23
the risk is  4262.45159182 : iteration  24
the risk is  3597.43288372 : iteration  25
the risk is  3630.37275309 : iteration  26
the risk is  3416.90555764 : iteration  27
the risk is  3231.67752292 : iteration  28
the risk is  3067.06135222 : iteration  29
the risk is  3068.01751595 : iteration  30
the risk is  2837.21316484 : iteration  31
the risk is  2861.54175094 : iteration  32
the risk is  2882.57628875 : iteration  33
the risk is  2884.26828219 : iteration  34
the risk is  2828.53404188 : iteration  35
the risk is  2796.81024109 : iteration  36
the risk is  2797.53900777 : iteration  37
the risk is  2732.19345743 : iteration  38
the risk is  2732.61579409 : iteration  39
the risk is  2731.71791174 : iteration  40
the risk is  2731.69969785 : iteration  41
the risk is  2697.96490653 : iteration  42
the risk is  2701.75150852 : iteration  43
the risk is  2691.7827796 : iteration  44
the risk is  2685.52503714 : iteration  45
the risk is  2682.24063911 : iteration  46
the risk is  2682.9200891 : iteration  47
the risk is  2677.89536187 : iteration  48
the risk is  2682.55850291 : iteration  49
the risk is  2679.22931728 : iteration  50
the risk is  2670.30610646 : iteration  51
the risk is  2671.0322023 : iteration  52
the risk is  2671.6853807 : iteration  53
the risk is  2671.95900081 : iteration  54
```

```
the risk is  2302.62544852 : iteration  2971
the risk is  2302.58945269 : iteration  2972
the risk is  2302.58648492 : iteration  2973
the risk is  2302.57391052 : iteration  2974
the risk is  2302.25005454 : iteration  2975
the risk is  2302.31946634 : iteration  2976
the risk is  2302.32903285 : iteration  2977
the risk is  2302.35492851 : iteration  2978
the risk is  2302.31174504 : iteration  2979
the risk is  2302.27500694 : iteration  2980
the risk is  2302.31435114 : iteration  2981
the risk is  2302.19095506 : iteration  2982
the risk is  2302.26699555 : iteration  2983
the risk is  2302.15816727 : iteration  2984
the risk is  2302.15528497 : iteration  2985
the risk is  2301.85693878 : iteration  2986
the risk is  2301.84030531 : iteration  2987
the risk is  2301.88186747 : iteration  2988
the risk is  2301.86694427 : iteration  2989
the risk is  2301.93227755 : iteration  2990
the risk is  2301.91759511 : iteration  2991
the risk is  2301.93009355 : iteration  2992
the risk is  2301.94260635 : iteration  2993
the risk is  2301.93036143 : iteration  2994
the risk is  2301.896843 : iteration  2995
the risk is  2301.85756132 : iteration  2996
the risk is  2301.84642162 : iteration  2997
the risk is  2301.82495288 : iteration  2998
the risk is  2301.82008546 : iteration  2999
the risk is  2301.83293867 : iteration  3000
```

After 3000 iterations, the training risk decreases down to 2301. We plot the training risk over the number of iterations.

```
In [474]: xpos = np.arange(len(training_risk_3))+ 1

          plt.plot(xpos, training_risk_3, 'bo', ms =1)
          plt.xlabel('Iterations')
          plt.ylabel('the Training risk')
          plt.title('Stochastic gradient with decreasing rate on the Third data')
          x1,x2,y1,y2 = plt.axis()
          plt.axis((x1,x2,2300,3600))
          plt.show()
```




**Is this strategy better than having a constant $\epsilon$?** With the constant step size(learning rate), the stochastic gradient produces lower training risk after the same number of iterations. This could be predictable in that the step size gets too small too fast when we use decreasing learning rates and therefore could not approach to the minimum as much as the stochastic gradient descent with constant step size does. It is worth to mention that the graph of training risk is much smoother curve with decreasing learning rate than with constant learning rate.

**kernel logistic regression with a polynomial kernel of degree 2**

```
In [19]: nr.seed(0)
         valid = nr.choice(sp_xtrain3.shape[0], sp_xtrain3.shape[0]//3, replace =False)
         sp_xvalid = sp_xtrain3[valid, ]
         sp_xtrain0 = sp_xtrain3[np.array([i not in valid for i in np.arange(sp_xtrain3.shape[0])]) ,:]
         sp_yvalid = sp_ytrain[valid]
         sp_ytrain0 = sp_ytrain[np.array([i not in valid for i in np.arange(sp_xtrain3.shape[0])])]
```

```
In [45]: sp_xtrain0.shape
         sp_xvalid.shape
```

```
Out[45]: (1724, 33)
```

```
In [12]: def quadratic_K(rho, X_train, X_test):
             return np.square(np.dot(X_train, np.transpose(X_test)) + rho)
```

```
In [13]: kf_ = KFold(3448, n_folds=10)
```

```
In [10]: # Risk function
         def R_(a, y_test, K):
```

```
        temp = np.dot(K,a)
        one = np.linspace(1,1, len(y_test))
        return np.dot(np.transpose(1- y_test),temp) + np.dot(one, np.log(1+ np.exp(-temp)))

    def stochastic_gradient_kernel(a,X,y, rho,e, lamda,n, K_):
        K__ = K_(rho,X, X)
        a_temp = a
        risk = []
        nr.seed(0)
        indice = nr.choice(X.shape[0], n, replace = True)
        for i in np.arange(n):
            # risk_temp = R_(a_temp, X, y, K)
            j = indice[i]
            # print ('the risk is ', risk_temp ,': iteration ', (i+1))
            b = lamda * a_temp
            b[j] = b[j] - y[j] + s(np.dot(K__[j,], a_temp))
            a_temp = a_temp - e * b
            # risk = risk + [risk_temp]
        return a_temp

    def tenfold_cv_(rho, e, lamda, n, K_):
        R = 0
        for train_index, test_index in kf_:
            X_train, X_test = sp_xtrain0[train_index, :], sp_xtrain0[test_index, :]
            y_train, y_test = sp_ytrain0[train_index], sp_ytrain0[test_index]
            a0 = np.zeros(X_train.shape[0])
            a = stochastic_gradient_kernel(a0,X_train,y_train, rho,e, lamda,n, K_)
            temp = np.dot(a, np.square( np.dot(X_train, np.transpose(X_test))+ rho))
            one = np.linspace(1,1, len(y_test))
            R = R + np.dot(np.transpose(1- y_test),temp) + np.dot(one, np.log(1+ np.exp(-temp)))
        return R/10
```

In [94]: `tenfold_cv_(0, 1e-5, 1e-3, 500, quadratic_K)`

Out[94]: 238.81979676288202

In [95]: `tenfold_cv_(1e-3, 1e-5, 1e-3, 500, quadratic_K)`

Out[95]: 238.81964581654387

In [96]: `tenfold_cv_(1e-1, 1e-5, 1e-3, 500, quadratic_K)`

Out[96]: 238.80433972893556

In [97]: `tenfold_cv_(1e-2, 1e-5, 1e-3, 500, quadratic_K)`

Out[97]: 238.81828400128205

In [98]: `tenfold_cv_(1, 1e-5, 1e-3, 500, quadratic_K)`

Out[98]: 238.6327606396122

In [99]: `tenfold_cv_(10, 1e-5, 1e-3, 500, quadratic_K)`

Out[99]: 234.76740849818816

In [100]: `tenfold_cv_(20, 1e-5, 1e-3, 500, quadratic_K)`

Out[100]: 232.1178878996912

In [101]: tenfold_cv_(30, 1e-5, 1e-3, 500, quadratic_K)

Out[101]: 235.93093307801436

In [102]: tenfold_cv_(18, 1e-5, 1e-3, 500, quadratic_K)

Out[102]: 232.13978447133462

In [103]: tenfold_cv_(19, 1e-5, 1e-3, 500, quadratic_K)

Out[103]: 232.09213791761448

10-fold cross validation shows that average training risk is the lowest when $\rho = 19$.

In [11]:
```
def stochastic_gradient_kernel_print(X_train, X_test,  y_train, y_test, rho,e, lamda,n, K_):
    K1 = K_(rho,X_train,X_train)
    K2 = K_(rho, X_train, X_test)
    risk = []
    nr.seed(0)
    indice = nr.choice(X_train.shape[0], n, replace = True)
    a0 = np.zeros(X_train.shape[0])
    temp = np.dot(a0, K2)
    one = np.linspace(1,1, len(y_test))
    a_temp = a0
    for i in np.arange(n):
        risk_temp = np.dot(np.transpose(1- y_test),temp) + np.dot(one, np.log(1+ np.exp(-temp)
        j = indice[i]
        print ('the risk is ', risk_temp ,': iteration ', (i+1))
        b = lamda * a_temp
        diff = y_train[j] - s(np.dot(a_temp,K1[j, ]))
        b[j] = b[j] - diff
        a_temp = a_temp - e * b
        temp = (1-e * lamda) * temp + e * diff * K2[j, ]
        risk = risk + [risk_temp]
    return [risk, a_temp]
```

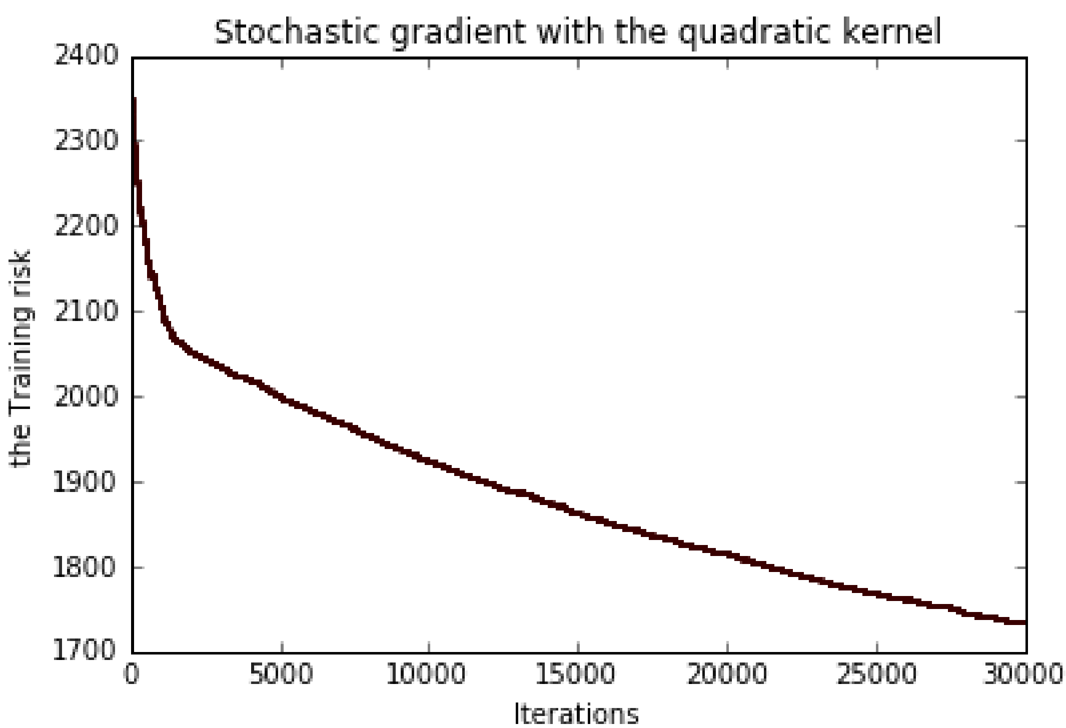
In [87]:
```
training_risk_kernel = stochastic_gradient_kernel_print(sp_xtrain0, sp_xtrain0,
                                                        sp_ytrain0,sp_ytrain0, 19, 1e-5, 1e-3,
```

```
the risk is  2389.97147857 : iteration  1
the risk is  2388.50723823 : iteration  2
the risk is  2386.91925433 : iteration  3
the risk is  2385.37438057 : iteration  4
the risk is  2383.98065696 : iteration  5
the risk is  2385.3855362 : iteration  6
the risk is  2386.7790168 : iteration  7
the risk is  2385.25283763 : iteration  8
the risk is  2383.78251136 : iteration  9
the risk is  2382.38667382 : iteration  10
the risk is  2383.79471226 : iteration  11
the risk is  2382.27707668 : iteration  12
the risk is  2380.88604362 : iteration  13
the risk is  2379.4186275 : iteration  14
the risk is  2378.00834721 : iteration  15
```

```
the risk is  1733.23091165 : iteration  29986
the risk is  1733.34676539 : iteration  29987
the risk is  1733.25907708 : iteration  29988
the risk is  1733.17172024 : iteration  29989
the risk is  1733.08556903 : iteration  29990
the risk is  1733.27284225 : iteration  29991
the risk is  1733.19904327 : iteration  29992
the risk is  1733.130197 : iteration  29993
the risk is  1733.08688898 : iteration  29994
the risk is  1733.31562529 : iteration  29995
the risk is  1733.26332413 : iteration  29996
the risk is  1733.35728836 : iteration  29997
the risk is  1733.57522346 : iteration  29998
the risk is  1733.48289971 : iteration  29999
the risk is  1733.59507488 : iteration  30000
```

After the 30000 iteration, we achieve the lowest training risk 1733.59507488. We plot the training risk over the number of iterations.

```
In [88]: xpos = np.arange(len(training_risk_kernel[0]))+ 1

         plt.plot(xpos, training_risk_kernel[0], 'ro', ms =1)
         plt.xlabel('Iterations')
         plt.ylabel('the Training risk')
         plt.title('Stochastic gradient with the quadratic kernel')

         plt.show()
```



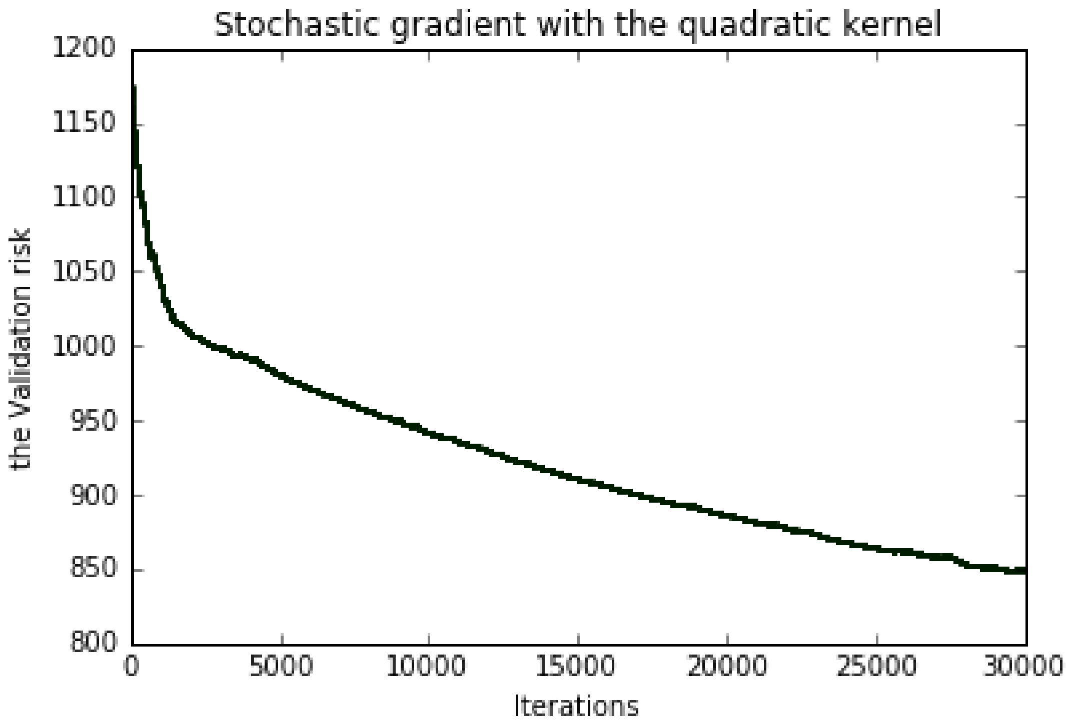
```
In [90]: validation_risk_kernel = stochastic_gradient_kernel_print(sp_xtrain0, sp_xvalid, sp_ytrain0,
                                                                    sp_yvalid, 19, 1e-5, 1e-3, 30000, quad
```

```
the risk is  1194.98573929 : iteration  1
the risk is  1194.20773155 : iteration  2
the risk is  1193.36570734 : iteration  3
the risk is  1192.55076004 : iteration  4
the risk is  1191.80927024 : iteration  5
the risk is  1192.55777194 : iteration  6
the risk is  1193.30053708 : iteration  7
the risk is  1192.48966295 : iteration  8
the risk is  1191.70459066 : iteration  9
the risk is  1190.96184941 : iteration  10
the risk is  1191.7099116 : iteration  11
the risk is  1190.90356364 : iteration  12
the risk is  1190.16327242 : iteration  13
the risk is  1189.3846922 : iteration  14
the risk is  1188.63467339 : iteration  15
the risk is  1187.88011407 : iteration  16
the risk is  1187.10411683 : iteration  17
the risk is  1187.89314283 : iteration  18
the risk is  1187.09505982 : iteration  19
the risk is  1186.31956597 : iteration  20
the risk is  1185.5250791 : iteration  21
the risk is  1184.80206725 : iteration  22
the risk is  1184.07914126 : iteration  23
the risk is  1184.81928945 : iteration  24
the risk is  1184.05880372 : iteration  25
the risk is  1183.34059455 : iteration  26
the risk is  1182.58525699 : iteration  27
the risk is  1181.87188959 : iteration  28
the risk is  1181.06952257 : iteration  29
the risk is  1181.80351573 : iteration  30
the risk is  1182.53824473 : iteration  31
the risk is  1181.82484125 : iteration  32
the risk is  1181.11381816 : iteration  33
the risk is  1180.37291976 : iteration  34
the risk is  1179.66487554 : iteration  35
the risk is  1180.42071671 : iteration  36
the risk is  1179.70344952 : iteration  37
the risk is  1178.91792199 : iteration  38
the risk is  1179.64837315 : iteration  39
the risk is  1180.45956979 : iteration  40
the risk is  1181.19454139 : iteration  41
the risk is  1180.4698393 : iteration  42
the risk is  1179.67675291 : iteration  43
the risk is  1180.40076674 : iteration  44
the risk is  1179.62847651 : iteration  45
the risk is  1178.9219093 : iteration  46
the risk is  1178.09916193 : iteration  47
the risk is  1178.85748216 : iteration  48
the risk is  1179.61947512 : iteration  49
the risk is  1178.90099568 : iteration  50
the risk is  1179.69532893 : iteration  51
```

```
the risk is  849.327567256 : iteration  29968
the risk is  849.287251117 : iteration  29969
the risk is  849.243460775 : iteration  29970
the risk is  849.180511968 : iteration  29971
the risk is  849.101660624 : iteration  29972
the risk is  849.029430679 : iteration  29973
the risk is  848.937254226 : iteration  29974
the risk is  848.878913036 : iteration  29975
the risk is  848.820981821 : iteration  29976
the risk is  848.753475218 : iteration  29977
the risk is  848.702492902 : iteration  29978
the risk is  848.792638151 : iteration  29979
the risk is  848.708065881 : iteration  29980
the risk is  848.641003732 : iteration  29981
the risk is  848.608099105 : iteration  29982
the risk is  848.729436766 : iteration  29983
the risk is  848.64519449 : iteration  29984
the risk is  848.599187664 : iteration  29985
the risk is  848.540981381 : iteration  29986
the risk is  848.636837815 : iteration  29987
the risk is  848.570570446 : iteration  29988
the risk is  848.504476826 : iteration  29989
the risk is  848.439001153 : iteration  29990
the risk is  848.580956226 : iteration  29991
the risk is  848.528980923 : iteration  29992
the risk is  848.479532176 : iteration  29993
the risk is  848.450313802 : iteration  29994
the risk is  848.618014203 : iteration  29995
the risk is  848.58231981 : iteration  29996
the risk is  848.662712355 : iteration  29997
the risk is  848.825113273 : iteration  29998
the risk is  848.756474267 : iteration  29999
the risk is  848.848799934 : iteration  30000
```

The validataion risk, which intially was 1194.9857, decreases down to 848.848799934 after 30000 iterations.

```
In [91]: xpos = np.arange(len(validation_risk_kernel[0]))+ 1

         plt.plot(xpos, validation_risk_kernel[0], 'go', ms =1)
         plt.xlabel('Iterations')
         plt.ylabel('the Validation risk')
         plt.title('Stochastic gradient with the quadratic kernel')

         plt.show()
```




**Repeat the same experiment with the linear kernel**

```
In [92]: # define the linear kernel function

         def linear_K(rho, X_train, X_test):
             return np.dot(X_train, np.transpose(X_test)) + rho
```

**10-fold Cross Validation**

```
In [104]: tenfold_cv_(0, 1e-5, 1e-3, 500, linear_K)

Out[104]: 238.81973628796601

In [105]: tenfold_cv_(1, 1e-5, 1e-3, 500, linear_K)

Out[105]: 238.63221702134652

In [106]: tenfold_cv_(20, 1e-5, 1e-3, 500, linear_K)

Out[106]: 232.59098370465694

In [107]: tenfold_cv_(10, 1e-5, 1e-3, 500, linear_K)

Out[107]: 234.53067277148665

In [108]: tenfold_cv_(30, 1e-5, 1e-3, 500, linear_K)

Out[108]: 252.04620848751625

In [109]: tenfold_cv_(19, 1e-5, 1e-3, 500, linear_K)
```

Out[109]: 232.18632045192498

```
In [110]: tenfold_cv_(18, 1e-5, 1e-3, 500, linear_K)
```

Out[110]: 231.97186548732543

```
In [111]: tenfold_cv_(15, 1e-5, 1e-3, 500, linear_K)
```

Out[111]: 232.2471268876871

```
In [112]: tenfold_cv_(16, 1e-5, 1e-3, 500, linear_K)
```

Out[112]: 232.02402140148493

```
In [113]: tenfold_cv_(17, 1e-5, 1e-3, 500, linear_K)
```

Out[113]: 231.92511545177331

It is when $\rho = 17$ that the 10-fold cross validation records the lowest average risk.

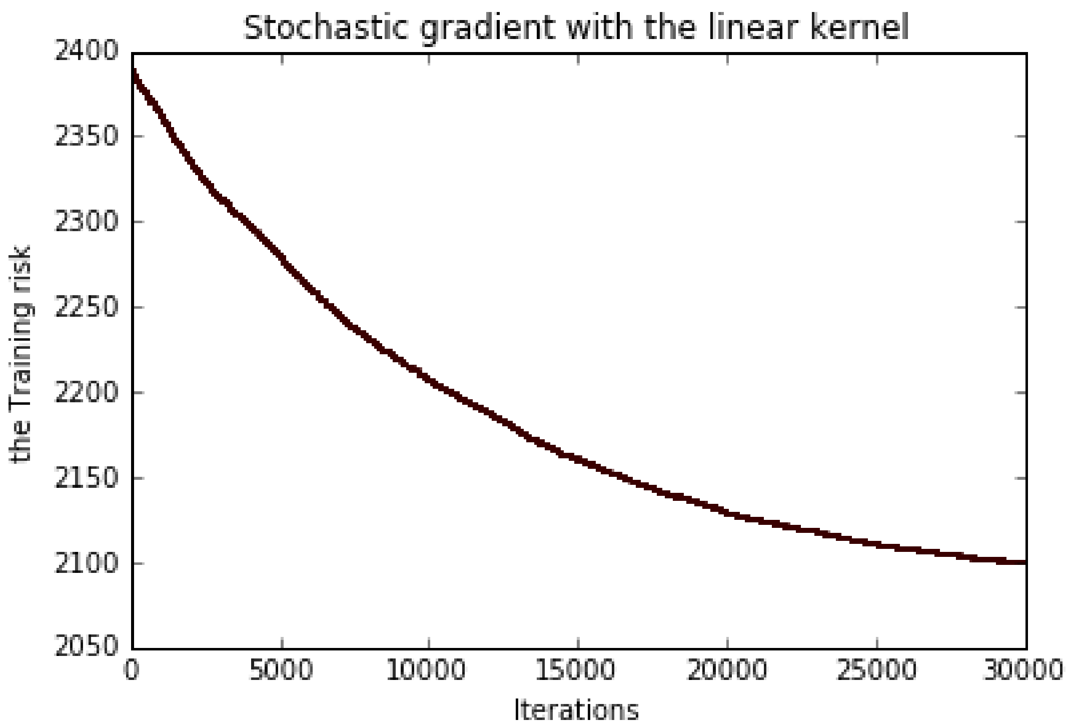
```
In [116]: training_risk_kernel_linear = stochastic_gradient_kernel_print(sp_xtrain0, sp_xtrain0,
                                                                          sp_ytrain0,sp_ytrain0, 17, 1e-5, 1e-3
```

```
the risk is  2389.97147857 : iteration  1
the risk is  2389.90650462 : iteration  2
the risk is  2389.8385066 : iteration  3
the risk is  2389.77134939 : iteration  4
the risk is  2389.70773888 : iteration  5
the risk is  2389.77139251 : iteration  6
the risk is  2389.83472035 : iteration  7
the risk is  2389.76799327 : iteration  8
the risk is  2389.70251425 : iteration  9
the risk is  2389.63874838 : iteration  10
the risk is  2389.70253505 : iteration  11
the risk is  2389.6357788 : iteration  12
the risk is  2389.57202282 : iteration  13
the risk is  2389.50624651 : iteration  14
the risk is  2389.4417862 : iteration  15
the risk is  2389.37710109 : iteration  16
the risk is  2389.31116399 : iteration  17
the risk is  2389.37715804 : iteration  18
the risk is  2389.3100696 : iteration  19
the risk is  2389.24373855 : iteration  20
the risk is  2389.17696497 : iteration  21
the risk is  2389.11327866 : iteration  22
the risk is  2389.04945504 : iteration  23
the risk is  2389.1132089 : iteration  24
the risk is  2389.04750431 : iteration  25
the risk is  2388.98383783 : iteration  26
the risk is  2388.91815372 : iteration  27
the risk is  2388.85450706 : iteration  28
the risk is  2388.78653727 : iteration  29
the risk is  2388.85018377 : iteration  30
the risk is  2388.91378182 : iteration  31
the risk is  2388.85013542 : iteration  32
the risk is  2388.78649886 : iteration  33
```

```
the risk is  2099.46398546 : iteration  29950
the risk is  2099.45182715 : iteration  29951
the risk is  2099.43939032 : iteration  29952
the risk is  2099.46167567 : iteration  29953
the risk is  2099.48244462 : iteration  29954
the risk is  2099.469774 : iteration  29955
the risk is  2099.45665782 : iteration  29956
the risk is  2099.44473125 : iteration  29957
the risk is  2099.43191747 : iteration  29958
the risk is  2099.45189557 : iteration  29959
the risk is  2099.47233093 : iteration  29960
the risk is  2099.46016775 : iteration  29961
the risk is  2099.4475425 : iteration  29962
the risk is  2099.43506049 : iteration  29963
the risk is  2099.45804901 : iteration  29964
the risk is  2099.48040111 : iteration  29965
the risk is  2099.46771892 : iteration  29966
the risk is  2099.48816267 : iteration  29967
the risk is  2099.47569469 : iteration  29968
the risk is  2099.46231486 : iteration  29969
the risk is  2099.44914244 : iteration  29970
the risk is  2099.4364409 : iteration  29971
the risk is  2099.42442745 : iteration  29972
the risk is  2099.41223704 : iteration  29973
the risk is  2099.40031315 : iteration  29974
the risk is  2099.38766452 : iteration  29975
the risk is  2099.37501964 : iteration  29976
the risk is  2099.36285995 : iteration  29977
the risk is  2099.34991765 : iteration  29978
the risk is  2099.37047535 : iteration  29979
the risk is  2099.35866724 : iteration  29980
the risk is  2099.34653594 : iteration  29981
the risk is  2099.33298303 : iteration  29982
the risk is  2099.35505313 : iteration  29983
the risk is  2099.34294781 : iteration  29984
the risk is  2099.32999438 : iteration  29985
the risk is  2099.31740012 : iteration  29986
the risk is  2099.33811277 : iteration  29987
the risk is  2099.32590296 : iteration  29988
the risk is  2099.31378006 : iteration  29989
the risk is  2099.30166082 : iteration  29990
the risk is  2099.32389443 : iteration  29991
the risk is  2099.31120034 : iteration  29992
the risk is  2099.29829192 : iteration  29993
the risk is  2099.28467202 : iteration  29994
the risk is  2099.30780137 : iteration  29995
the risk is  2099.29442467 : iteration  29996
the risk is  2099.31464708 : iteration  29997
the risk is  2099.33758979 : iteration  29998
the risk is  2099.32546177 : iteration  29999
the risk is  2099.345999 : iteration  30000
```

After 30000 iterations, the training risk decreases down to 2099.345999 in logistic ridge regression with the linear kernel. We plot the training risk over the number of iterations.

```
In [119]: xpos = np.arange(len(training_risk_kernel_linear[0]))+ 1

          plt.plot(xpos, training_risk_kernel_linear[0], 'ro', ms =1)
          plt.xlabel('Iterations')
          plt.ylabel('the Training risk')
          plt.title('Stochastic gradient with the linear kernel')

          plt.show()
```




**Validation risk**

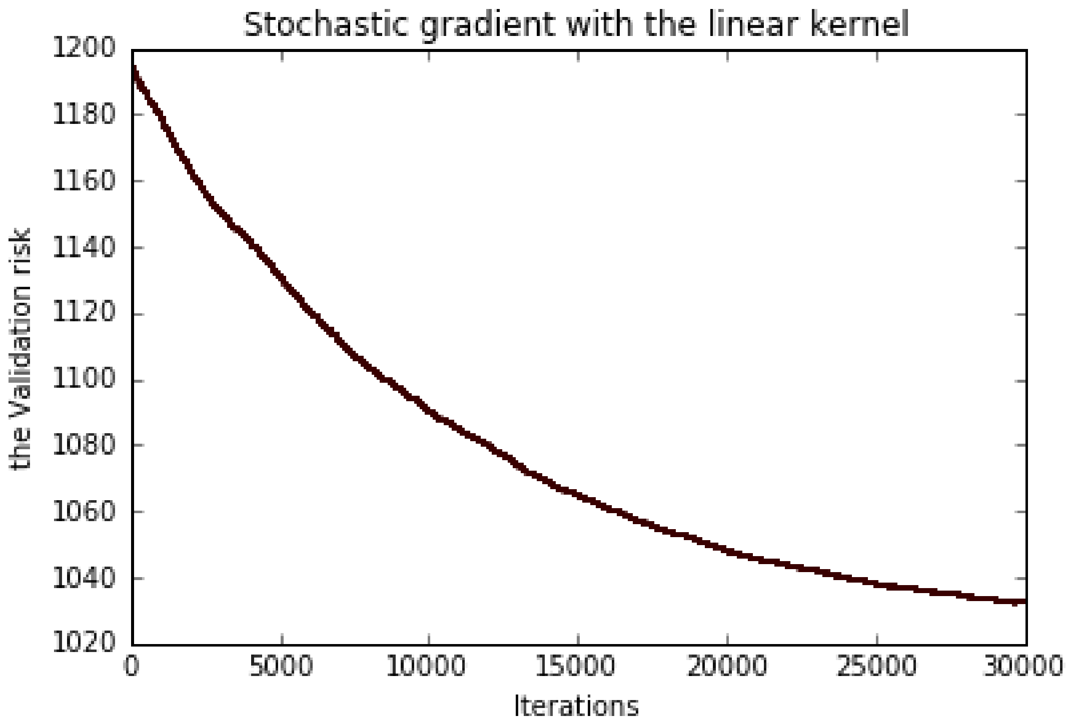
```
In [118]: validation_risk_kernel_linear = stochastic_gradient_kernel_print(sp_xtrain0, sp_xvalid,
                                                                         sp_ytrain0,sp_yvalid, 19, 1e-5, 1e-3,
```

```
the risk is  1194.98573929 : iteration  1
the risk is  1194.94743648 : iteration  2
the risk is  1194.90757046 : iteration  3
the risk is  1194.86825616 : iteration  4
the risk is  1194.83066519 : iteration  5
the risk is  1194.86830714 : iteration  6
the risk is  1194.90578659 : iteration  7
the risk is  1194.86656239 : iteration  8
the risk is  1194.82790979 : iteration  9
the risk is  1194.7902374 : iteration  10
the risk is  1194.82789426 : iteration  11
```

```
the risk is  1032.3747848 : iteration  29982
the risk is  1032.38861913 : iteration  29983
the risk is  1032.38133567 : iteration  29984
the risk is  1032.37357353 : iteration  29985
the risk is  1032.36602824 : iteration  29986
the risk is  1032.37914496 : iteration  29987
the risk is  1032.37173782 : iteration  29988
the risk is  1032.36439965 : iteration  29989
the risk is  1032.35706368 : iteration  29990
the risk is  1032.37094066 : iteration  29991
the risk is  1032.36338388 : iteration  29992
the risk is  1032.35567309 : iteration  29993
the risk is  1032.34763211 : iteration  29994
the risk is  1032.36201619 : iteration  29995
the risk is  1032.35408391 : iteration  29996
the risk is  1032.36694375 : iteration  29997
the risk is  1032.38125155 : iteration  29998
the risk is  1032.3739098 : iteration  29999
the risk is  1032.38697496 : iteration  30000
```

The validation risk, which initially was 1194.9857, decreases down to 1032.38697. We plot the validation risk over the number of iterations.

```
In [121]: xpos = np.arange(len(validation_risk_kernel_linear[0]))+ 1

          plt.plot(xpos, validation_risk_kernel_linear[0], 'ro', ms =1)
          plt.xlabel('Iterations')
          plt.ylabel('the Validation risk')
          plt.title('Stochastic gradient with the linear kernel')

          plt.show()
```




**Does the quadratic kernel overfit the data?** According to the validation risk, the quadratic kernel does not appear to overfit the data. It gives lower validation risk than the linear kernel and the propotions of numbers of data and the total risk are almost the same in the validation risk and the training risk.

**For each kernel, should you decrease or increase $\lambda$ to try to improve performance?**

```
In [123]: tenfold_cv_(17, 1e-5, 1e-3, 500, linear_K)

Out[123]: 231.92511545177331

In [126]: tenfold_cv_(17, 1e-5, 1e-2, 500, linear_K)

Out[126]: 231.92515041100859

In [128]: tenfold_cv_(17, 1e-5, 1, 500, linear_K)

Out[128]: 231.92902977056451

In [130]: tenfold_cv_(17, 1e-5, 10, 500, linear_K)

Out[130]: 231.96726038354353

In [132]: tenfold_cv_(17, 1e-5, 1e-5, 500, linear_K)

Out[132]: 231.92511160659714

In [133]: tenfold_cv_(19, 1e-5, 1e-5, 500, quadratic_K)

Out[133]: 232.09213334165784
```

In [134]: tenfold_cv_(19, 1e-5, 1e-3, 500, quadratic_K)

Out[134]: 232.09213791761448

In [135]: tenfold_cv_(19, 1e-5, 1e-1, 500, quadratic_K)

Out[135]: 232.09259575993278

In [137]: tenfold_cv_(19, 1e-5, 10, 500, quadratic_K)

Out[137]: 232.14075190405794

In [138]: tenfold_cv_(19, 1e-5, 100, 500, quadratic_K)

Out[138]: 232.72369887001486

We tried several different $\lambda$ values in cross validation to see if they can improve performance. There was no noticeable decrease in the training risk compared to the training risk when $\lambda$ is $10^{-3}$. This implies that there is no need to decrease or increase $\lambda$ to improve performance.

**Kaggle** Based on training risks, or empirical risks, of each classifier, the best classifier is the logistic regression with batch gradient descent along with the third preprocessed data. We submitted the predicted labels to Kaggle that is derived by the best classifier, and the kaggle score was 0.78231.

In [35]: empirical_risk = logistic_with_batch_gradient(w_0, sp_xtrain3, sp_ytrain, 2e-3, 20000)

```
the risk is  3584.95721786 : iteration  1
the risk is  4106.32058967 : iteration  2
the risk is  3197.9220317 : iteration  3
the risk is  3967.10472524 : iteration  4
the risk is  2682.77707914 : iteration  5
the risk is  2754.18057516 : iteration  6
the risk is  2463.95894291 : iteration  7
the risk is  2417.07294157 : iteration  8
the risk is  2271.68299745 : iteration  9
the risk is  2223.3775302 : iteration  10
the risk is  2155.00067837 : iteration  11
the risk is  2123.86911517 : iteration  12
the risk is  2089.30249473 : iteration  13
the risk is  2069.74877244 : iteration  14
the risk is  2050.14159893 : iteration  15
the risk is  2037.07824844 : iteration  16
the risk is  2024.64782528 : iteration  17
the risk is  2015.31136188 : iteration  18
the risk is  2006.66649052 : iteration  19
the risk is  1999.60404597 : iteration  20
the risk is  1993.12895398 : iteration  21
the risk is  1987.52734896 : iteration  22
the risk is  1982.38003115 : iteration  23
the risk is  1977.75244675 : iteration  24
the risk is  1973.46265985 : iteration  25
the risk is  1969.50619858 : iteration  26
the risk is  1965.79984485 : iteration  27
the risk is  1962.32388258 : iteration  28
the risk is  1959.03723767 : iteration  29
the risk is  1955.92161228 : iteration  30
```

```
the risk is  1819.05681309 : iteration  19957
the risk is  1819.05680154 : iteration  19958
the risk is  1819.05678999 : iteration  19959
the risk is  1819.05677845 : iteration  19960
the risk is  1819.0567669 : iteration  19961
the risk is  1819.05675536 : iteration  19962
the risk is  1819.05674381 : iteration  19963
the risk is  1819.05673227 : iteration  19964
the risk is  1819.05672073 : iteration  19965
the risk is  1819.05670919 : iteration  19966
the risk is  1819.05669765 : iteration  19967
the risk is  1819.05668611 : iteration  19968
the risk is  1819.05667458 : iteration  19969
the risk is  1819.05666304 : iteration  19970
the risk is  1819.05665151 : iteration  19971
the risk is  1819.05663997 : iteration  19972
the risk is  1819.05662844 : iteration  19973
the risk is  1819.05661691 : iteration  19974
the risk is  1819.05660538 : iteration  19975
the risk is  1819.05659385 : iteration  19976
the risk is  1819.05658233 : iteration  19977
the risk is  1819.0565708 : iteration  19978
the risk is  1819.05655927 : iteration  19979
the risk is  1819.05654775 : iteration  19980
the risk is  1819.05653623 : iteration  19981
the risk is  1819.0565247 : iteration  19982
the risk is  1819.05651318 : iteration  19983
the risk is  1819.05650166 : iteration  19984
the risk is  1819.05649015 : iteration  19985
the risk is  1819.05647863 : iteration  19986
the risk is  1819.05646711 : iteration  19987
the risk is  1819.0564556 : iteration  19988
the risk is  1819.05644408 : iteration  19989
the risk is  1819.05643257 : iteration  19990
the risk is  1819.05642106 : iteration  19991
the risk is  1819.05640955 : iteration  19992
the risk is  1819.05639804 : iteration  19993
the risk is  1819.05638653 : iteration  19994
the risk is  1819.05637502 : iteration  19995
the risk is  1819.05636351 : iteration  19996
the risk is  1819.05635201 : iteration  19997
the risk is  1819.05634051 : iteration  19998
the risk is  1819.056329 : iteration  19999
the risk is  1819.0563175 : iteration  20000
```

In [41]:
```
temp = np.dot(sp_test, empirical_risk[1])
prob = s(temp)
sp_ytest = prob >= 1/2
sp_ypred = np.asarray([[i+1, sp_ytest[i]] for i in np.arange(5857)])
np.savetxt('sp_ytest2.csv', sp_ypred, fmt = '%1.u' , delimiter = ',', header = 'Id,Category',c
```

### 1.3.1 Problem 4: Revisiting Logistic Regression

Recall that in logistic regression, the logistic function, $s(z) = \frac{1}{1+e^{-z}}$ , is used to map output values between $[0, 1]$. Consider the function $g(z) = \frac{\tanh(z)+1}{2}$, where $\tanh(z) = 2s(2z) - 1$, that also maps outputs between $[0, 1]$. In this problem, we will explore using $g(z)$ instead of the logistic function in logistic regression.

$$g(z) = \frac{\tanh(z) + 1}{2} = \frac{2s(2z)}{2} = \frac{2}{2 + 2e^{-2z}} = \frac{1}{2} + \frac{1 - e^{-2z}}{2 + 2e^{-2z}} = \frac{1}{2} + \frac{e^z - e^{-z}}{2(e^z + e^{-z})}.$$

and

$$g'(z) = \frac{dg(z)}{dz} = \frac{2(e^z + e^{-z})(e^z + e^{-z}) - 2(e^z - e^{-z})^2}{4(e^z + e^{-z})^2} = \frac{1}{2} - \frac{\tanh(z)^2}{2}$$

The appropriate batch gradient ascent update function is

$$\begin{aligned}
w &\leftarrow w - \lambda \nabla J(w) \\
&= w - \lambda \sum_{i=1}^{n} \left(\frac{y_i g'(X_i \cdot w)}{g(X_i \cdot w)} - \frac{(1 - y_i) g'(X_i \cdot w)}{1 - g(X_i \cdot w)}\right) X_i \\
&= w - \lambda \sum_{i=1}^{n} \left(\frac{y_i(\frac{1}{2} - \frac{\tanh^2(X_i \cdot w)}{2})}{\frac{\tanh(X_i \cdot w)+1}{2}} - \frac{(1 - y_i)(\frac{1}{2} - \frac{\tanh^2(X_i \cdot w)}{2})}{1 - \frac{\tanh(X_i \cdot w)+1}{2}}\right) X_i \\
&= w - \lambda \sum_{i=1}^{n} (2y_i - 1 - \tanh(X_i \cdot w)) X_i
\end{aligned}$$

### 1.3.2 Problem 5: Real World Spam Classification

Linear SVM can not utilize the new feature as he desired, because the spams usually are sent around midnight and number of milliseconds since the previous midnight are concentrated in either (0, t] or [86400000 -t ,86400000) for small $t$ and linear SVM can not classify wheter this new feature is either in one of the two intervals (0, t] and [86400000 -t ,86400000) or not. Therefore in order to classify the feature correctly with linear SVM, Daniel may want to try one of following methods: 1. introduce the new feature as the number of milliseconds since the previous noon, rather than since the previous midnight so that the time around midnight could have continous possible values, 2. use quadratic kernel instead of linear kernel so that the linear SVM can classify wheter the new feature is either in one of the two intervals (0, t] and [86400000 -t ,86400000) or not.