CSE 2004: DATABASE MANAGEMENT SYSTEMS PROJECT

LOGISTIC REGRESSION WITH A NEURAL NETWORK MINDSET

Rohit Subramanian(17BCE1291)

B Shruti(17BCE1299)

V Priyanka(17BCE1001)

LOGISTIC REGRESSION WITH A NEURAL NETWORK MINDSET

Rohit Subramanian, V Priyanka, B Shruti

ABSTRACT - For the last ten years Neural networks have attracted a great deal of attention. They offer an alternative approach to computing and to understanding of the human brain. Neural networks take a different approach to problem solving than that of conventional computers. Conventional computers use an algorithmic approach i.e. the computer follows a set of instructions in order to solve a problem. Unless the specific steps that the computer needs to follow are known the computer cannot solve the problem. That restricts the problem solving capability of conventional computers to problems that we already understand and know how to solve. But computers would be so much more useful if they could do things that we don't exactly know how to do.

KEYWORDS - Logistic Regression, Neural Network, Machine Learning, Binary Classification

INTRODUCTION - Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables. In this project we build a logistic regression classifier to recognize cats. This is a simple image recognition algorithm that can correctly classify pictures as cat or non-cat.

OTHER RELATED WORKS (LITERATURE SURVEY)-

PAPER 1:

TITLE: Variable Selection in Logistic Regression Model

This paper proposes a new technique for variable selection in a logistic regression model considering that variable selection is one of the most important aspects of solving a pattern recognition or machine learning problem The newly proposed variable selection method is – the logistic elastic net which is particularly useful when the number of predictors (p) is much bigger than the number of observations (n). However it is not very useful when the number of predictors (p) is much larger than the number of observations (n).

In least squares estimation (LSE), the unknown values of the parameters, $\beta 0, \beta 1, \ldots$, in the regression function, $f(\vec{x};\vec{\beta})$, are estimated by finding numerical values for the parameters that minimize the sum of the squared deviations between the observed responses and the functional portion of the model. However LSE has two major drawbacks. Two standard techniques (ridge regression and subset selection) can be used to improve LSE, but they also have defects. Ridge regression is a constant process that shrinks coefficients (which makes the model stable). However it is not an easy to interpret model. Subset selection makes it easy to interpret, but it is a discrete process and this can affect the prediction accuracy.

In linear regression model, Least Absolute Shrinkage and Selection Operator (LASSO) and many improved methods of LASSO are used. It shrinks a few coefficients and sets others to 0, and consequently influences it to hold the advantages of both subset selection and ridge regression. The LASSO method is useful not only for the simple linear regression model, but also other models, such as Cox model, graph model, semi-parametric model and Generalized linear model (GLM).

In statistics and, in particular, in the fitting of linear or **logistic** regression models, the **elastic net** is a regularized regression method that linearly combines the L_1 and L_2 penalties of the lasso and ridge methods. One of the most important properties of the logistic elastic net is the grouping effect which means that the strongly correlated predictors tend to be in or out of the model together. This has been proved in the paper.

The correlation algorithm and the tuning parameters selection method are similar to those in Ref.[13] by Zou and Hastie. Logistic elastic net is quite effective in dealing with micro-array data, in which the explanatory variables have strong multi-collinearity, but tends to have its own set of drawbacks as well.

PAPER 2:

TITLE: Logistic Regression by Means of Evolutionary Radial Basis Function Neural Networks

This paper displays a focused report in multiclass learning which combines different statistical and delicate computing elements such as multilogistic regression, radial basis function neural networks. This paper proposes a hybrid multilogistic methodology, named logistic regression using initial and radial basis function (RBF) covariates. The procedure for getting the coefficients is completed in three stages. Firstly , an evolutionary programming (EP) algorithm is applied with the end goal to create an RBF neural system (RBFNN) with a decreased number of RBF transformations and the least complex structure conceivable. Then, the initial attribute space is transformed by adding the nonlinear transformations of the input variables given by the RBFs of the best individual in the final generation. Finally, a maximum likelihood optimization method determines the coefficients associated with a multilogistic regression model built in this augmented covariate space.

We expand the ideas presented in a recently proposed combination of NNs and logistic regression. This strategy permits the generation of hybrid linear/nonlinear classification surfaces and the identification of possible strong interactions that may exist between the attributes which characterize the classification problem. The methodology, called logistic regression using initial and radial basis function (LIRBF) covariates, comprises of a multilogistic regression model based on the combination of the initial covariates and the RBFs of a RBFNN (local approximator). RBFNNs usually use hyper-ellipsoids to split the pattern space, which is different from multilayer perceptrons that base their classifications on hyperplanes, characterized by a weighted sum.

This paper joined three ground-breaking methods utilized in machine learning research: multilogistic regression, EAs, and RBFNNs. The methodology is a mixture of the three components to solve various multiclass problems and follows up the proposal of Hastie, Tibshirani, and Friedman of generalizing the linear logistic model using additive models of basis functions. In particular, this new methodology comprises of a multilogistic regression model based on the combination of linear covariates and Gaussian RBFs for the predictor function.

From the experimental analysis of the outcomes, a few conclusions can be made . Above all else, the covariate determination process incorporated in the SLogistic algorithm of the SLIRBF and SLRBF techniques is vital in some datasets to abstain from overfitting.

<u>PAPER 3:</u>

TITLE: Optimizing Single-Trial EEG Classification by Stationary Matrix Logistic Regression in Brain–Computer Interface

In this paper, a single-trial EEG classification technique is proposed that simultaneously optimizes the classifier's discriminativity and robustness against the within-session nonstationarity of the EEG data through a single optimization paradigm. In addition, the trial results on two benchmark data sets illustrate that the methodology proposed in the paper absolutely outperforms the alternate approaches that help in decreasing classification error rates.

A brain–computer interface (BCI), sometimes called a neural-control interface (NCI), mind-machine interface (MMI), direct neural interface (DNI), or brain–machine interface (BMI), is a direct communication pathway between an enhanced or wired brain and an external device. BCI differs from neuromodulation in that it allows for bidirectional information flow. BCIs are often directed at researching, mapping, assisting, augmenting, or repairing human cognitive or sensory-motor functions.

Among various methods used for the noninvasive estimation of electrophysiological signals induced by voluntary changes in the elements of brain oscillations, the electroencephalography (EEG) is commercially affordable, furthermore, has magnificent temporal resolution , which empowers realtime interaction through BCI.

The common spatial patterns (CSPs) algorithm is used to to transform the signals acquired from a large and equivocal array of sensors into a small number of task discriminative components that can be fed into some relatively simple classifiers.The principle behind CSP is to find the discriminative spatial projection with the band power representation of signals, so that the projected signals have maximal variance in one class and minimal variance in the other class. Also at the same time, it diagonalizes the covariance matrices corresponding to two classes of data, and then, the projection comprises of eigenvectors with a few of the largest and smallest eigenvalues. Once these CSP projections have been designed, the logarithms of the projected signal powers are used as the low-dimensional features for a supervised classification algorithm. The linear discriminant analysis (LDA) classifier is often preferred for being extremely simple. A stationarity regularizer for the MLR is designed to enforce samples from the same class to have similar latent function values, which helps to find a stable classifier that generalizes to test trials well. Furthermore, we have provided an efficient optimization algorithm to solve the resulting convex problem, by applying the off-the-shelf APG.

PAPER 4 :

TITLE : Orthogonal Decision Trees

This paper presented the idea of symmetrical choice trees and offered a technique to build them. Symmetrical choice trees are practically symmetrical to one another and they give a proficient repetition free portrayal of expansive groups that are regularly created by procedures like Boosting [2], [3], Bagging [4], Stacking [5], and Random Forests [6]. The proposed system is additionally liable to be exceptionally helpful in ensemble-based mining of distributed [10] and stream information [7], [8]. The proposed methodology misuses the before work done by Kargupta et al. [20], [9], which demonstrated that the Fourier change of choice trees can be proficiently figured. This work demonstrates that we can process the tree once again from its Fourier range. The paper likewise offered a gathering of new outcomes with respect to the properties of the multivariate Fourier range of choice trees. In spite of the fact that the paper thinks about the Fourier portrayal, this is unmistakably by all account not the only accessible direct portrayal around. In any case, our work demonstrates that it is especially reasonable for speaking to choice trees.

This work likewise opens up a few new conceivable outcomes. Direct frameworks hypothesis offers numerous apparatuses for investigating properties like solidness and assembly. For instance, eigenvalues of a straight framework are straightforwardly connected with the soundness of the framework. Comparable ideas might be helpful in under-standing the conduct of substantial gatherings.

The proposed methodology abuses the before work done by Kargupta et al. [20], [9], which demonstrated that the Fourier change of choice trees can be effectively registered. This work demonstrates that we can figure the tree once again from its Fourier range. The paper likewise offered a gathering of new outcomes with respect to the properties of the multivariate Fourier range of choice trees. In spite of the fact that the paper thinks about the Fourier portrayal, this is plainly by all account not the only accessible straight portrayal around. Notwithstanding, our work demonstrates that it is especially reasonable for speaking to choice trees.

This work likewise opens up a few new conceivable outcomes. Direct frameworks hypothesis offers numerous devices for investigating properties like steadiness and assembly. For instance, eigenvalues of a straight framework are specifically connected with the soundness of the framework. Comparable ideas might be helpful in under-standing the conduct of expansive gatherings.

PAPER 5 :

TITLE : Compression and Aggregation for Logistic Regression Analysis in Data Cubes

Logistic Regression is a critical procedure for breaking down and foreseeing information with clear cut properties. In this paper, they think about supporting on the web investigative handling (OLAP) of calculated relapse examination for multidimensional information in an information 3D shape where it is costly in time and space to fabricate strategic relapse models for every cell from the crude information. They propose a novel plan to pack the information so that we can remake strategic relapse models to answer any OLAP question without getting to the crude information. In light of a first arrange guess to the greatest probability evaluating conditions, they build a compression scheme that packs each base cell into a little compacted information hinder with fundamental data to help the total of strategic relapse models. Accumulation recipes for inferring abnormal state strategic relapse models from lower level part cells are given. They demonstrate that the pressure is asymptotically lossless as in the totaled estimator veers off from the genuine model by a blunder that is limited and ways to deal with zero when the information measure increments. The outcomes demonstrate that the proposed pressure and accumulation plan can make possible OLAP of strategic relapse in an information 3D shape. Further, it underpins ongoing calculated relapse examination of stream information, which must be checked once and can't be for all time held. Trial results approve their hypothetical investigation and show that their strategy can drastically spare time and space costs with no debasement of the demonstrating exactness.

In this paper, they have proposed an asymptotically lossless pressure strategy to help proficient calculated regression examination in a data cube environment. They have built a compression scheme that packs a data cell into a compressed representation whose size is free of the extent of the cell. Under regularity conditions, they have demonstrated that the accumulated estimator is emphatically reliable and asymptotically mistake free.

The proposed procedure enables us to rapidly perform OLAP activities and produce calculated relapse models at any level in an information solid shape without recovering or storing raw data. Their exploratory investigations demonstrate that their compression and aggregation strategies can altogether spare processing time with little loss of precision. In addition, the accumulation blunder reduces as the extent of the information block increments. They are presently stretching out the strategy to more broad circumstances, for example, semi probability estimation for generalized measurable models.

PAPER 6 :

TITLE : Minimax Sparse Logistic Regression for Very High-Dimensional Feature Selection

As a result of the solid convexity and probabilistic underpinnings, calculated relapse (LR) is generally utilized in some real-world applications. In any case, in numerous issues, for example, bioinformatics, picking a small subset of highlights with the most discriminative power are alluring for deciphering the expectation show, strong forecasts or more profound investigation. To accomplish a meager arrangement as for information highlights, numerous scanty LR models are proposed. Notwithstanding, it is still challenging for them to productively acquire fair inadequate answers for high-dimensional issues (e.g., recognizing the most discriminative subset from a great many highlights). In this paper, we propose another minimax inadequate LR display for high-dimensional component choices, which can be proficiently understood by a cutting plane calculation. To settle the resultant nonsmooth minimax subproblems, a smoothing coordinate plunge strategy is introduced. Numerical issues and intermingling rate of this strategy are deliberately contemplated. Trial results on a few engineered and true datasets demonstrate that the proposed strategy can get better expectation precision with a similar number of chosen includes and has better or aggressive adaptability on high-dimensional issues contrasted and the benchmark techniques, including the l1-regularized LR.

In this paper, they proposed another weight scaling plan to accomplish the scanty LR by presenting a scaling vector $d \in [0,1]m$ into the LR show. To incite the sparsity, they forced a l1-requirement $||d||1 \leq r$ on d, where r was a moderate estimation to the coveted number of highlights. They additionally changed the model as a Taste issue and after that understood it through a productive cutting plane strategy. To understand the resultant nonsmooth minimax subproblem of the cutting plane technique, we exhibited a smoothing coordinate drop calculation that could accomplish a feebly straight intermingling rate. In every emphasis of the cutting plane calculation, at most r highlights were chosen. In this way, the quantity of chose highlights can be controlled by changing the most extreme number of cutting plane emphasess Tm or r. Broad examinations on a few engineered and true datasets checked the productivity and viability of the proposed technique. In this paper, they considered just the issue of high-dimensional straight element choices. Notwithstanding, in some genuine issues, the highlights may have complex nonlinear structures like the Corral (or XOR) issue [34]. The identification of these nonlinear highlights of complex structures was alluring for some applications.

IMPLEMENTATION-

Logistic Regression with a Neural Network mindset

Welcome to your first (required) programming assignment! You will build a logistic regression classifier to recognize cats. This assignment will step you through how to do this with a Neural Network mindset, and so will also hone your intuitions about deep learning.

Instructions:

- Do not use loops (for/while) in your code, unless the instructions explicitly ask you to do so.

You will learn to:

- Build the general architecture of a learning algorithm, including:
    - Initializing parameters
    - Calculating the cost function and its gradient
    - Using an optimization algorithm (gradient descent)
- Gather all three functions above into a main model function, in the right order.

1 - Packages

First, let's run the cell below to import all the packages that you will need during this assignment.

- numpy is the fundamental package for scientific computing with Python.
- h5py is a common package to interact with a dataset that is stored on an H5 file.
- matplotlib is a famous library to plot graphs in Python.
- PIL and scipy are used here to test your model with your own picture at the end.

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import h5py
import scipy
from PIL import Image
from scipy import ndimage
from lr_utils import load_dataset

%matplotlib inline
```

2 - Overview of the Problem set

Problem Statement: You are given a dataset ("data.h5") containing:

```
- a training set of m_train images labeled as cat (y=1) or non-cat (y=0)
- a test set of m_test images labeled as cat or non-cat
- each image is of shape (num_px, num_px, 3) where 3 is for the 3 channels (RGB). Thus,
each image is square (height = num_px) and (width = num_px).
```

You will build a simple image-recognition algorithm that can correctly classify pictures as cat or non-cat.

Let's get more familiar with the dataset. Load the data by running the following code.

In [2]:

```
# Loading the data (cat/non-cat)
train_set_x_orig, train_set_y, test_set_x_orig, test_set_y, classes = load_dataset()
```

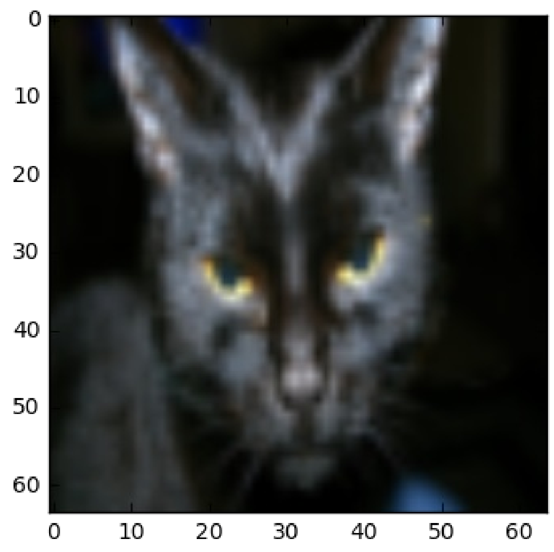
We added "_orig" at the end of image datasets (train and test) because we are going to preprocess them. After preprocessing, we will end up with train_set_x and test_set_x (the labels train_set_y and test_set_y don't need any preprocessing).

Each line of your train_set_x_orig and test_set_x_orig is an array representing an image. You can visualize an example by running the following code. Feel free also to change the `index` value and re-run to see other images.

In [3]:

```
# Example of a picture
index = 25
plt.imshow(train_set_x_orig[index])
print ("y = " + str(train_set_y[:,index]) + ", it's a '" + classes[np.squeeze(train_set_y[:,index])
].decode("utf-8") +  "' picture.")
```

```
y = [1], it's a 'cat' picture.
```



Many software bugs in deep learning come from having matrix/vector dimensions that don't fit. If you can keep your matrix/vector dimensions straight you will go a long way toward eliminating many bugs.

Exercise: Find the values for:

```
- m_train (number of training examples)
- m_test (number of test examples)
- num_px (= height = width of a training image)
```

Remember that `train_set_x_orig` is a numpy-array of shape (m_train, num_px, num_px, 3). For instance, you can access `m_train` by writing `train_set_x_orig.shape[0]`.

In [5]:

```
### START CODE HERE ### (≈ 3 lines of code)
m_train = train_set_y.shape[1]
m_test = test_set_y.shape[1]
num_px = train_set_x_orig.shape[1]
### END CODE HERE ###

print ("Number of training examples: m_train = " + str(m_train))
print ("Number of testing examples: m_test = " + str(m_test))
print ("Height/Width of each image: num_px = " + str(num_px))
print ("Each image is of size: (" + str(num_px) + ", " + str(num_px) + ", 3)")
print ("train_set_x shape: " + str(train_set_x_orig.shape))
print ("train_set_y shape: " + str(train_set_y.shape))
print ("test_set_x shape: " + str(test_set_x_orig.shape))
print ("test_set_y shape: " + str(test_set_y.shape))
```

```
Number of training examples: m_train = 209
Number of testing examples: m_test = 50
Height/Width of each image: num_px = 64
Each image is of size: (64, 64, 3)
train_set_x shape: (209, 64, 64, 3)
train_set_y shape: (1, 209)
test_set_x shape: (50, 64, 64, 3)
test_set_y shape: (1, 50)
```

Expected Output for m_train, m_test and num_px:

m_train	209
m_test	50
num_px	64

For convenience, you should now reshape images of shape (num_px, num_px, 3) in a numpy-array of shape (num_px $*$ num_px $*$ 3, 1). After this, our training (and test) dataset is a numpy-array where each column represents a flattened image. There should be m_train (respectively m_test) columns.

Exercise: Reshape the training and test data sets so that images of size (num_px, num_px, 3) are flattened into single vectors of shape (num_px $*$ num_px $*$ 3, 1).

A trick when you want to flatten a matrix X of shape (a,b,c,d) to a matrix X_flatten of shape (b$*$c$*$d, a) is to use:

```
X_flatten = X.reshape(X.shape[0], -1).T      # X.T is the transpose of X
```

In [6]:

```
# Reshape the training and test examples

### START CODE HERE ### (≈ 2 lines of code)
train_set_x_flatten = train_set_x_orig.reshape(train_set_x_orig.shape[0], -1).T
test_set_x_flatten = test_set_x_orig.reshape(test_set_x_orig.shape[0], -1).T
### END CODE HERE ###

print ("train_set_x_flatten shape: " + str(train_set_x_flatten.shape))
print ("train_set_y shape: " + str(train_set_y.shape))
print ("test_set_x_flatten shape: " + str(test_set_x_flatten.shape))
print ("test_set_y shape: " + str(test_set_y.shape))
print ("sanity check after reshaping: " + str(train_set_x_flatten[0:5,0]))
```

```
train_set_x_flatten shape: (12288, 209)
train_set_y shape: (1, 209)
test_set_x_flatten shape: (12288, 50)
test_set_y shape: (1, 50)
sanity check after reshaping: [17 31 56 22 33]
```

Expected Output:

train_set_x_flatten shape	(12288, 209)
train_set_y shape	(1, 209)
test_set_x_flatten shape	(12288, 50)
test_set_y shape	(1, 50)
sanity check after reshaping	[17 31 56 22 33]

To represent color images, the red, green and blue channels (RGB) must be specified for each pixel, and so the pixel value is actually a vector of three numbers ranging from 0 to 255.

One common preprocessing step in machine learning is to center and standardize your dataset, meaning that you substract the mean of the whole numpy array from each example, and then divide each example by the standard deviation of the whole numpy array. But for picture datasets, it is simpler and more convenient and works almost as well to just divide every row of the dataset by 255 (the maximum value of a pixel channel).

Let's standardize our dataset.

In [9]:

```
train_set_x = train_set_x_flatten / 255.
test_set_x = test_set_x_flatten / 255.
```

What you need to remember:

Common steps for pre-processing a new dataset are:

Common steps for pre-processing a new dataset are:

- Figure out the dimensions and shapes of the problem (m_train, m_test, num_px, ...)
- Reshape the datasets such that each example is now a vector of size (num_px * num_px * 3, 1)
- "Standardize" the data

3 - General Architecture of the learning algorithm

It's time to design a simple algorithm to distinguish cat images from non-cat images.

You will build a Logistic Regression, using a Neural Network mindset. The following Figure explains why **Logistic Regression is actually a very simple Neural Network!**

Mathematical expression of the algorithm:

For one example $x^{(i)}$: $$z^{(i)} = w^T x^{(i)} + b \tag{1}$$ $$\hat{y}^{(i)} = a^{(i)} = sigmoid(z^{(i)})\tag{2}$$ $$ \mathcal{L}(a^{(i)}, y^{(i)}) = - y^{(i)} \log(a^{(i)}) - (1-y^{(i)}) \log(1-a^{(i)})\tag{3}$$

The cost is then computed by summing over all training examples: $$ J = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(a^{(i)}, y^{(i)})\tag{6}$$

Key steps: In this exercise, you will carry out the following steps:

```
- Initialize the parameters of the model
- Learn the parameters for the model by minimizing the cost
- Use the learned parameters to make predictions (on the test set)
- Analyse the results and conclude
```

4 - Building the parts of our algorithm

The main steps for building a Neural Network are:

1. Define the model structure (such as number of input features)
2. Initialize the model's parameters
3. Loop:
    - Calculate current loss (forward propagation)
    - Calculate current gradient (backward propagation)
    - Update parameters (gradient descent)

You often build 1-3 separately and integrate them into one function we call `model()`.

4.1 - Helper functions

Exercise: Using your code from "Python Basics", implement `sigmoid()`. As you've seen in the figure above, you need to compute $sigmoid(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}}$ to make predictions. Use np.exp().

In [10]:

```
# GRADED FUNCTION: sigmoid

def sigmoid(z):
    """
    Compute the sigmoid of z

    Arguments:
    x -- A scalar or numpy array of any size.

    Return:
    s -- sigmoid(z)
    """

    ### START CODE HERE ### (≈ 1 line of code)
    s = 1 / (1 + np.exp(-z))
    ### END CODE HERE ###

    return s
```

In [11]:

```
print ("sigmoid(0) = " + str(sigmoid(0)))
print ("sigmoid(9.2) = " + str(sigmoid(9.2)))
```

```
sigmoid(0) = 0.5
sigmoid(9.2) = 0.999898970806
```

Expected Output:

sigmoid([0, 2])	[0.5 0.88079708]

4.2 - Initializing parameters

Exercise: Implement parameter initialization in the cell below. You have to initialize w as a vector of zeros. If you don't know what numpy function to use, look up np.zeros() in the Numpy library's documentation.

In [12]:

```
# GRADED FUNCTION: initialize_with_zeros

def initialize_with_zeros(dim):
    """
    This function creates a vector of zeros of shape (dim, 1) for w and initializes b to 0.

    Argument:
    dim -- size of the w vector we want (or number of parameters in this case)

    Returns:
    w -- initialized vector of shape (dim, 1)
    b -- initialized scalar (corresponds to the bias)
    """

    ### START CODE HERE ### (≈ 1 line of code)
    w = np.zeros(shape=(dim, 1))
    b = 0
    ### END CODE HERE ###

    assert(w.shape == (dim, 1))
    assert(isinstance(b, float) or isinstance(b, int))

    return w, b
```

In [13]:

```
dim = 2
w, b = initialize_with_zeros(dim)
print ("w = " + str(w))
print ("b = " + str(b))
```

```
w = [[ 0.]
 [ 0.]]
b = 0
```

Expected Output:

** w **	[[0.] [0.]]
** b **	0

For image inputs, w will be of shape (num_px $\times$ num_px $\times$ 3, 1).

4.3 - Forward and Backward propagation

Now that your parameters are initialized, you can do the "forward" and "backward" propagation steps for learning the parameters.

Exercise: Implement a function `propagate()` that computes the cost function and its gradient.

Hints:

Forward Propagation:

- You get X
- You compute $A = \sigma(w^T X + b) = (a^{(1)}, a^{(2)}, ..., a^{(m-1)}, a^{(m)})$
- You calculate the cost function: $J = -\frac{1}{m}\sum_{i=1}^{m}y^{(i)}\log(a^{(i)})+(1-y^{(i)})\log(1-a^{(i)})$

Here are the two formulas you will be using:
$$ \frac{\partial J}{\partial w} = \frac{1}{m}X(A-Y)^T\tag{7}$$$$ \frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^m (a^{(i)}-y^{(i)})\tag{8}$$

In [14]:

```
# GRADED FUNCTION: propagate

def propagate(w, b, X, Y):
    """
    Implement the cost function and its gradient for the propagation explained above

    Arguments:
    w -- weights, a numpy array of size (num_px * num_px * 3, 1)
    b -- bias, a scalar
    X -- data of size (num_px * num_px * 3, number of examples)
    Y -- true "label" vector (containing 0 if non-cat, 1 if cat) of size (1, number of examples)

    Return:
    cost -- negative log-likelihood cost for logistic regression
    dw -- gradient of the loss with respect to w, thus same shape as w
    db -- gradient of the loss with respect to b, thus same shape as b

    Tips:
    - Write your code step by step for the propagation
    """

    m = X.shape[1]

    # FORWARD PROPAGATION (FROM X TO COST)
    ### START CODE HERE ### (≈ 2 lines of code)
    A = sigmoid(np.dot(w.T, X) + b)  # compute activation
    cost = (- 1 / m) * np.sum(Y * np.log(A) + (1 - Y) * (np.log(1 - A)))  # compute cost
    ### END CODE HERE ###

    # BACKWARD PROPAGATION (TO FIND GRAD)
    ### START CODE HERE ### (≈ 2 lines of code)
    dw = (1 / m) * np.dot(X, (A - Y).T)
    db = (1 / m) * np.sum(A - Y)
    ### END CODE HERE ###

    assert(dw.shape == w.shape)
    assert(db.dtype == float)
    cost = np.squeeze(cost)
    assert(cost.shape == ())
    grads = {"dw": dw,
             "db": db}
    
    return grads, cost
```

In [15]:

```
w, b, X, Y = np.array([[1], [2]]), 2, np.array([[1,2], [3,4]]), np.array([[1, 0]])
grads, cost = propagate(w, b, X, Y)
print ("dw = " + str(grads["dw"]))
print ("db = " + str(grads["db"]))
print ("cost = " + str(cost))
```

```
dw = [[ 0.99993216]
 [ 1.99980262]]
db = 0.499935230625
cost = 6.000064773 19
```

Expected Output:

** dw **	[[0.99845601] [2.39507239]]
** db **	0.00145557813678
** cost **	5.801545319394553

4.4 - Optimization

- You have initialized your parameters.
- You are also able to compute a cost function and its gradient.
- Now, you want to update the parameters using gradient descent.

Exercise: Write down the optimization function. The goal is to learn $w$ and $b$ by minimizing the cost function $J$. For a parameter $\theta$, the update rule is $ \theta = \theta - \alpha \text{ } d\theta$, where $\alpha$ is the learning rate.

In [16]:

```
# GRADED FUNCTION: optimize

def optimize(w, b, X, Y, num_iterations, learning_rate, print_cost = False):
    """
    This function optimizes w and b by running a gradient descent algorithm
    
    Arguments:
    w -- weights, a numpy array of size (num_px * num_px * 3, 1)
    b -- bias, a scalar
    X -- data of shape (num_px * num_px * 3, number of examples)
    Y -- true "label" vector (containing 0 if non-cat, 1 if cat), of shape (1, number of examples)
    num_iterations -- number of iterations of the optimization loop
    learning_rate -- learning rate of the gradient descent update rule
    print_cost -- True to print the loss every 100 steps
    
    Returns:
    params -- dictionary containing the weights w and bias b
    grads -- dictionary containing the gradients of the weights and bias with respect to the cost
function
    costs -- list of all the costs computed during the optimization, this will be used to plot the
learning curve.
    
    Tips:
    You basically need to write down two steps and iterate through them:
        1) Calculate the cost and the gradient for the current parameters. Use propagate().
        2) Update the parameters using gradient descent rule for w and b.
    """
    
    costs = []
    
    for i in range(num_iterations):
        
        
        # Cost and gradient calculation (≈ 1-4 lines of code)
        ### START CODE HERE ###
        grads, cost = propagate(w, b, X, Y)
        ### END CODE HERE ###

        # Retrieve derivatives from grads
        dw = grads["dw"]
        db = grads["db"]

        # update rule (≈ 2 lines of code)
        ### START CODE HERE ###
        w = w - learning_rate * dw   # need to broadcast
        b = b - learning_rate * db
        ### END CODE HERE ###

        # Record the costs
        if i % 100 == 0:
            costs.append(cost)

        # Print the cost every 100 training examples
        if print_cost and i % 100 == 0:
            print ("Cost after iteration %i: %f" % (i, cost))

    params = {"w": w,
              "b": b}

    grads = {"dw": dw,
             "db": db}

    return params, grads, costs
```

In [17]:

```
params, grads, costs = optimize(w, b, X, Y, num_iterations= 100, learning_rate = 0.009, print_cost = 
False)

print ("w = " + str(params["w"]))
print ("b = " + str(params["b"]))
print ("dw = " + str(grads["dw"]))
print ("db = " + str(grads["db"]))
```

```
w = [[ 0.1124579 ]
 [ 0.23106775]]
b = 1.55930492484
dw = [[ 0.90158428]
 [ 1.76250842]]
db = 0.430462071679
```

Expected Output:

w	[[0.19033591] [0.12259159]]
b	1.92535983008
dw	[[0.67752042] [1.41625495]]
db	0.219194504541

Exercise: The previous function will output the learned w and b. We are able to use w and b to predict the labels for a dataset X. Implement the `predict()` function. There are two steps to computing predictions:

1. Calculate $\hat{Y} = A = \sigma(w^T X + b)$
2. Convert the entries of a into 0 (if activation <= 0.5) or 1 (if activation > 0.5), stores the predictions in a vector `Y_prediction`. If you wish, you can use an `if`/`else` statement in a `for` loop (though there is also a way to vectorize this).

In [18]:

```
# GRADED FUNCTION: predict

def predict(w, b, X):
    '''
    Predict whether the label is 0 or 1 using learned logistic regression parameters (w, b)

    Arguments:
```

```
    w -- weights, a numpy array of size (num_px * num_px * 3, 1)
    b -- bias, a scalar
    X -- data of size (num_px * num_px * 3, number of examples)

    Returns:
    Y_prediction -- a numpy array (vector) containing all predictions (0/1) for the examples in X
    '''

    m = X.shape[1]
    Y_prediction = np.zeros((1, m))
    w = w.reshape(X.shape[0], 1)

    # Compute vector "A" predicting the probabilities of a cat being present in the picture
    ### START CODE HERE ### (≈ 1 line of code)
    A = sigmoid(np.dot(w.T, X) + b)
    ### END CODE HERE ###

    for i in range(A.shape[1]):
        # Convert probabilities a[0,i] to actual predictions p[0,i]
        ### START CODE HERE ### (≈ 4 lines of code)
        Y_prediction[0, i] = 1 if A[0, i] > 0.5 else 0
        ### END CODE HERE ###

    assert(Y_prediction.shape == (1, m))

    return Y_prediction
```

In [20]:

```
w = np.array([[0.1124579],[0.23106775]])
b = -0.3
X = np.array([[1.,-1.1,-3.2],[1.2,2.,0.1]])
print ("predictions = " + str(predict(w, b, X)))
```

```
predictions = [[ 1.  1.  0.]]
```

Expected Output:

predictions	[[1. 1. 0.]]

What to remember: You've implemented several functions that:

- Initialize (w,b)
- Optimize the loss iteratively to learn parameters (w,b):
    - computing the cost and its gradient
    - updating the parameters using gradient descent
- Use the learned (w,b) to predict the labels for a given set of examples

5 - Merge all functions into a model

You will now see how the overall model is structured by putting together all the building blocks (functions implemented in the previous parts) together, in the right order.

Exercise: Implement the model function. Use the following notation:

```
    - Y_prediction_test for your predictions on the test set
    - Y_prediction_train for your predictions on the train set
    - w, costs, grads for the outputs of optimize()
```

In [21]:

```
# GRADED FUNCTION: model

def model(X_train, Y_train, X_test, Y_test, num_iterations=2000, learning_rate=0.5, print_cost=Fals
e):
    """
    Builds the logistic regression model by calling the function you've implemented previously

    Arguments:
```

```
    X_train -- training set represented by a numpy array of shape (num_px * num_px * 3, m_train)
    Y_train -- training labels represented by a numpy array (vector) of shape (1, m_train)
    X_test -- test set represented by a numpy array of shape (num_px * num_px * 3, m_test)
    Y_test -- test labels represented by a numpy array (vector) of shape (1, m_test)
    num_iterations -- hyperparameter representing the number of iterations to optimize the paramet
ers
    learning_rate -- hyperparameter representing the learning rate used in the update rule of opti
mize()
    print_cost -- Set to true to print the cost every 100 iterations

    Returns:
    d -- dictionary containing information about the model.
    """

    ### START CODE HERE ###
    # initialize parameters with zeros (≈ 1 line of code)
    w, b = initialize_with_zeros(X_train.shape[0])

    # Gradient descent (≈ 1 line of code)
    parameters, grads, costs = optimize(w, b, X_train, Y_train, num_iterations, learning_rate, prin
t_cost)

    # Retrieve parameters w and b from dictionary "parameters"
    w = parameters["w"]
    b = parameters["b"]

    # Predict test/train set examples (≈ 2 lines of code)
    Y_prediction_test = predict(w, b, X_test)
    Y_prediction_train = predict(w, b, X_train)

    ### END CODE HERE ###

    # Print train/test Errors
    print("train accuracy: {} %".format(100 - np.mean(np.abs(Y_prediction_train - Y_train)) * 100))
    print("test accuracy: {} %".format(100 - np.mean(np.abs(Y_prediction_test - Y_test)) * 100))


    d = {"costs": costs,
         "Y_prediction_test": Y_prediction_test,
         "Y_prediction_train" : Y_prediction_train,
         "w" : w,
         "b" : b,
         "learning_rate" : learning_rate,
         "num_iterations": num_iterations}

    return d
```

Run the following cell to train your model.

In [22]:

```
d = model(train_set_x, train_set_y, test_set_x, test_set_y, num_iterations = 2000, learning_rate =
0.005, print_cost = True)
```

```
Cost after iteration 0: 0.693147
Cost after iteration 100: 0.584508
Cost after iteration 200: 0.466949
Cost after iteration 300: 0.376007
Cost after iteration 400: 0.331463
Cost after iteration 500: 0.303273
Cost after iteration 600: 0.279880
Cost after iteration 700: 0.260042
Cost after iteration 800: 0.242941
Cost after iteration 900: 0.228004
Cost after iteration 1000: 0.214820
Cost after iteration 1100: 0.203078
Cost after iteration 1200: 0.192544
Cost after iteration 1300: 0.183033
Cost after iteration 1400: 0.174399
Cost after iteration 1500: 0.166521
Cost after iteration 1600: 0.159305
Cost after iteration 1700: 0.152667
Cost after iteration 1800: 0.146542
Cost after iteration 1900: 0.140872
train accuracy: 99.04306220095694 %
```

```
test accuracy: 70.0 %
```

Expected Output:

**Cost after iteration 0 **	0.693147
$\vdots$	$\vdots$
Train Accuracy	99.04306220095694 %
Test Accuracy	70.0 %

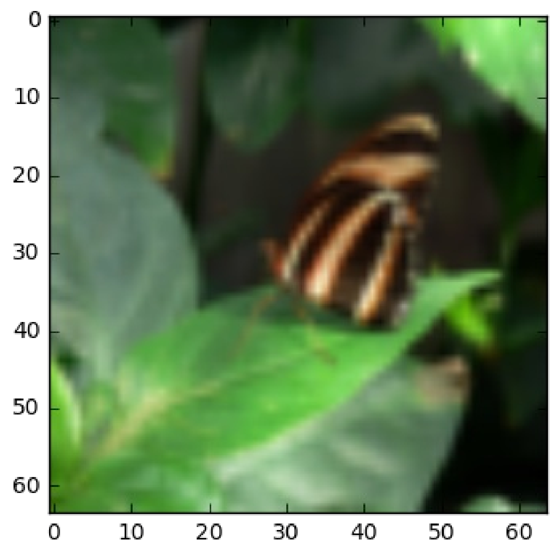
Comment: Training accuracy is close to 100%. This is a good sanity check: your model is working and has high enough capacity to fit the training data. Test error is 68%. It is actually not bad for this simple model, given the small dataset we used and that logistic regression is a linear classifier. But no worries, you'll build an even better classifier next week!

Also, you see that the model is clearly overfitting the training data. Later in this specialization you will learn how to reduce overfitting, for example by using regularization. Using the code below (and changing the `index` variable) you can look at predictions on pictures of the test set.

In [23]:

```
# Example of a picture that was wrongly classified.
index = 5
plt.imshow(test_set_x[:,index].reshape((num_px, num_px, 3)))
print ("y = " + str(test_set_y[0, index]) + ", you predicted that it is a \"" + classes[d["Y_predic
tion_test"][0, index]].decode("utf-8") +  "\" picture.")
```

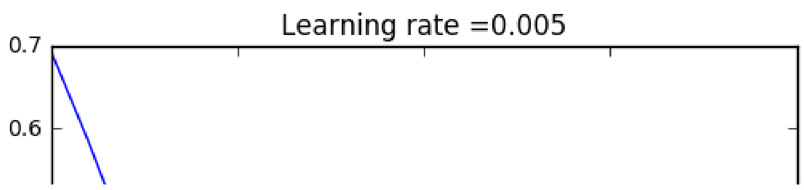
```
y = 0, you predicted that it is a "cat" picture.
```

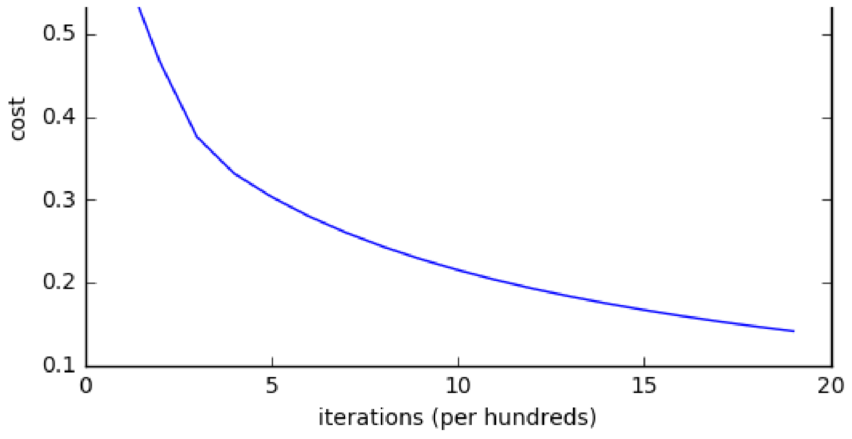


Let's also plot the cost function and the gradients.

In [24]:

```
# Plot learning curve (with costs)
costs = np.squeeze(d['costs'])
plt.plot(costs)
plt.ylabel('cost')
plt.xlabel('iterations (per hundreds)')
plt.title("Learning rate =" + str(d["learning_rate"]))
plt.show()
```





Interpretation: You can see the cost decreasing. It shows that the parameters are being learned. However, you see that you could train the model even more on the training set. Try to increase the number of iterations in the cell above and rerun the cells. You might see that the training set accuracy goes up, but the test set accuracy goes down. This is called overfitting.

6 - Further analysis (optional/ungraded exercise)

Congratulations on building your first image classification model. Let's analyze it further, and examine possible choices for the learning rate $\alpha$.

Choice of learning rate

Reminder: In order for Gradient Descent to work you must choose the learning rate wisely. The learning rate $\alpha$ determines how rapidly we update the parameters. If the learning rate is too large we may "overshoot" the optimal value. Similarly, if it is too small we will need too many iterations to converge to the best values. That's why it is crucial to use a well-tuned learning rate.

Let's compare the learning curve of our model with several choices of learning rates. Run the cell below. This should take about 1 minute. Feel free also to try different values than the three we have initialized the `learning_rates` variable to contain, and see what happens.

In [ ]:

```
learning_rates = [0.01, 0.001, 0.0001]
models = {}
for i in learning_rates:
    print ("learning rate is: " + str(i))
    models[str(i)] = model(train_set_x, train_set_y, test_set_x, test_set_y, num_iterations = 1500,
learning_rate = i, print_cost = False)
    print ('\n' + "-------------------------------------------------------" + '\n')

for i in learning_rates:
    plt.plot(np.squeeze(models[str(i)]["costs"]), label= str(models[str(i)]["learning_rate"]))

plt.ylabel('cost')
plt.xlabel('iterations (hundreds)')

legend = plt.legend(loc='upper center', shadow=True)
frame = legend.get_frame()
frame.set_facecolor('0.90')
plt.show()
```

Interpretation:

- Different learning rates give different costs and thus different predictions results.
- If the learning rate is too large (0.01), the cost may oscillate up and down. It may even diverge (though in this example, using 0.01 still eventually ends up at a good value for the cost).
- A lower cost doesn't mean a better model. You have to check if there is possibly overfitting. It happens when the training accuracy is a lot higher than the test accuracy.
- In deep learning, we usually recommend that you:
    - Choose the learning rate that better minimizes the cost function.
    - If your model overfits, use other techniques to reduce overfitting. (We'll talk about this in later videos.)

7 - Test with your own image (optional/ungraded exercise)

Congratulations on finishing this assignment. You can use your own image and see the output of your model. To do that:

```
1. Click on "File" in the upper bar of this notebook, then click "Open" to go on your
Coursera Hub.
2. Add your image to this Jupyter Notebook's directory, in the "images" folder
3. Change your image's name in the following code
4. Run the code and check if the algorithm is right (1 = cat, 0 = non-cat)!
```

In [ ]:

```
## START CODE HERE ## (PUT YOUR IMAGE NAME)
my_image = "my_image.jpg"   # change this to the name of your image file
## END CODE HERE ##

# We preprocess the image to fit your algorithm.
fname = "images/" + my_image
image = np.array(ndimage.imread(fname, flatten=False))
my_image = scipy.misc.imresize(image, size=(num_px,num_px)).reshape((1, num_px*num_px*3)).T
my_predicted_image = predict(d["w"], d["b"], my_image)

plt.imshow(image)
print("y = " + str(np.squeeze(my_predicted_image)) + ", your algorithm predicts a \"" +
classes[int(np.squeeze(my_predicted_image)),].decode("utf-8") +  "\" picture.")
```

What to remember from this assignment:

1. Preprocessing the dataset is important.
2. You implemented each function separately: initialize(), propagate(), optimize(). Then you built a model().
3. Tuning the learning rate (which is an example of a "hyperparameter") can make a big difference to the algorithm. You will see more examples of this later in this course!

Finally, if you'd like, we invite you to try different things on this Notebook. Make sure you submit before trying anything. Once you submit, things you can play with include:

```
- Play with the learning rate and the number of iterations
- Try different initialization methods and compare the results
- Test other preprocessings (center the data, or divide each row by its standard deviation)
```

Bibliography:

- http://www.wildml.com/2015/09/implementing-a-neural-network-from-scratch/
- https://stats.stackexchange.com/questions/211436/why-do-we-normalize-images-by-subtracting-the-datasets-image-mean-and-not-the-c

DISCUSSIONS-

Different learning rates give different costs and thus different predictions results.

- If the learning rate is too large (0.01), the cost may oscillate up and down. It may even diverge (though in this example, using 0.01 still eventually ends up at a good value for the cost).

- A lower cost doesn't mean a better model. You have to check if there is possibly overfitting. It happens when the training accuracy is a lot higher than the test accuracy.

RESULTS AND CONCLUSIONS- The Logistic Regression Classification algorithm works perfectly fine. Training accuracy is close to 100%. This is a good sanity check: your model is working and has high enough capacity to fit the training data. Test error is 68%. It is actually not bad for this simple model, given the small dataset we used and that logistic regression is a linear classifier. However, the model can be made even better by increasing the size of training data.

REFERENCES-

https://www.coursera.org

https://github.com

https://www.wikipedia.org/

https://www.doc.ic.ac.uk

https://www.medcalc.org

CITATIONS AND REFERENCES

TITLE: Variable Selection in Logistic Regression Model

Zhang, S., Zhang, L., Qiu, K., Lu, Y., & Cai, B. (2015). Variable selection in logistic regression model. *Chinese Journal of Electronics*, *24*(4), 813-817.

REFERENCES:

[1] A.E. Hoerl and R.W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problem", Technometrics, Vol.12, No.1, pp.55–67, 1970.
[2] R. Tibshirani, "Regression shrinkage and selection via the LASSO", Journal of the Royal Statistical Society, Series B, Vol.58, No.1, pp.267–288, 1996
[3] B. Efron, T. Hastie, I. Johnstone and R. Tibshirani, "Least angle regression", The Annals of Statistics, Vol.32, No.2, pp.407–499, 2004.
[4] J. Fan and R.Z. Li, "Variable selection via penalized likelihood", Journal of American Statistical Association, Vol.96, No.456, pp.1348–1360, 2001.
[5] R. Tibshirani and M. Saunders, "Sparsity and smoothness via the fused LASSO", Journal of the Royal Statistical Society, Series B, Vol.67, No.1, pp.91–108, 2005.
[6] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables", Journal of the Royal Statistical Society, Series B, Vol.68, No.1, pp.49–67, 2006.
[7] N. Meinshausen, "Relaxed LASSO", Computational Statistics and Data Analysis, Vol.52, No.1, pp.374–393, 2007.
[8] R. Tibshirani, "The LASSO method for variable selection in Cox models", Statistics in Medicine, Vol.16, No.4, pp.385–395, 1997.
[9] R. Li and H. Liang, "Variable selection in semiparamtric regression modeling", The Annals of Statistics, Vol.36, No.1, pp.261– 286, 2008.
[10] M.Y. Park and T. Hastie, "L1-regularization-path algorithm for generalized linear models", Journal of the Royal Statistical Society, Series B, Vol.69, No.4, pp.659–677, 2007.
[11] P. Cai and Q. Gao, "Variable selection in generalized linear model", Journal of University of Science and Technology of China, Vol.36, No.9, pp.927–931, 2006. (In Chinese)
[12] D.R. Wang and Z.Z. Zhang, "Variable selection in joint generalized linear models", Chinese Journal of Applied Probability and Statistics, Vol.25, No.3, pp.245–256, 2009.
[13] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net", Journal of the Royal Statistical Society, Series B, Vol.67, No.2, pp.301–320, 2005.
[14] I. Guyon, J. Weston, S. Barnhill and V. Vapnil, "Gene selection for cancer classification using support vector machines", Machine Learning, Vol.46, No.1–3, pp.389–422, 2002.
[15] J. Zhu and T. Hastie, "Classification of gene microarrays by penalized logistic regression", Biostatistics, Vol.5, No.3, pp.427– 443, 2004.
[16] T. Golub, D. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek,J. Mesirov, H. Coller, M. Loh, J. Downing and M. Calingiuri, "Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring", Science, Vol.286, No.5439, pp.531–537, 1999.
[17] M. Segal, K. Dahlquist and B. Conklin, "Regression approach for microarray data analysis", Computational Biology, Vol.10,No.6, pp.961–980, 2003.
[18] S. Van De Geer and H. Van Houwelingen, "High dimensional data: $p > n$ in mathematical statistics and bio-medical applications", Bernoulli, Vol.10, No.6, pp.939–943, 2004.
[19] E. Candes and T. Tao, "The Dantzig selector: Statistical estimation when p is much larger than n", The Annals of Statistics, Vol.35, No.6, pp.2313–2351, 2007.
[20] S.L. Zhang, Z.L. Ke, G.D. Wei and L.L. Zhang, "The random simulation algorithm for variable selection", Journal of Information and Computational Science, Vol.9, No.17, pp.5119– 5125, 2012.
[21] S.L. Zhang, L.C. Wang and H. Lian, "Estimation by polynomial splines with variable selection in additive Cox models", Statistics, Vol.48, No.1, pp.67–80, 2014.
[22] H. Zou, T. Hastie and R. Tibshirani, "Sparse principal component analysis", Journal of Computational and Graphical Statistics, Vol.15, No.2, pp.265–286, 2006.

TITLE: Logistic Regression by Means of Evolutionary Radial Basis Function Neural Networks

Gutiérrez, P. A., Hervás-Martínez, C., & Martínez-Estudillo, F. J. (2011). Logistic regression by means of evolutionary radial basis function neural networks. *IEEE Transactions on Neural Networks*, *22*(2), 246-263.

REFERENCES:

[1] A. K. Jain, R. P. Duin, and J. Mao, “Statistical pattern recognition: A review,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 22, no. 1, pp. 4–37, Jan. 2000.

[2] T. J. Hastie and R. J. Tibshirani, “Nonparametric regression and classification. Part II: Nonparametric classification,” in From Statistics to Neural Networks: Theory and Pattern Recognition Applications (Computer and System Sciences), vol. 136, V. Cherkassky, J. H. Friedman, and H. Wechsler, Eds. New York: Springer-Verlag, 1996, pp. 70–82.

[3] J. Friedman, “Multivariate adaptive regression splines (with discussion),” Ann. Stat., vol. 19, no. 1, pp. 1–141, 1991.

[4] C. Hervás-Martínez and F. Martínez-Estudillo, “Logistic regression using covariates obtained by product-unit neural network models,” Pattern Recognit., vol. 40, no. 1, pp. 52–64, Jan. 2007.

[5] C. Hervás-Martínez, F. J. Martínez-Estudillo, and M. Carbonero-Ruz, “Multilogistic regression by means of evolutionary product-unit neural networks,” Neural Netw., vol. 21, no. 7, pp. 951–961, Sep. 2008.

[6] I. T. Nabney, “Efficient training of RBF networks for classification,” Int. J. Neural Syst., vol. 14, no. 3, pp. 201–208, Jun. 2004.

[7] B. Krishnapuram, L. Carin, M. A. Figueiredo, and A. J. Hartemink, “Sparse multinomial logistic regression: Fast algorithms and generalization bounds,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 27, no. 6, pp. 957–968, Jun. 2005.

[8] C. M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), 1st ed. New York: Springer-Verlag, 2006.

[9] A. Asuncion and D. Newman. (2007). UCI Machine Learning Repository [Online]. Available: http://www.ics.uci.edu/

∼mlearn/MLRepository.html

[10] M. J. Orr, “Optimising the widths of radial basis functions,” in Proc. 5th Brazilian Symp. Neural Netw., Belo Horizonte, Brazil, Dec. 1998, pp. 26–29.

[11] S. Chen, C. F. N. Cowan, and P. M. Grant, “Orthogonal least squares learning algorithm for radial basis functions,” IEEE Trans. Neural Netw., vol. 2, no. 2, pp. 302–309, Mar. 1991.

[12] K. J. Hunt, D. Sbarbaro, R. Zbikowski, and P. J. Gawthrop, “Neural networks for control systems—A survey,” Automatica, vol. 28, no. 6, pp. 1083–1112, Nov. 1992.

[13] J. B. Gomm and D. L. Yu, “Selecting radial basis function network centers with recursive orthogonal least squares training,” IEEE Trans. Neural Netw., vol. 11, no. 2, pp. 306–314, Mar. 2000.

[14] Z. Uykan and C. Guzelis, “Input-output clustering for determining centers of radial basis function network,” in Proc. Eur. Conf. Circuit Theory Design, vol. 2. 1997, pp. 435–439.

[15] O. Buchtala, M. Klimek, and B. Sick, “Evolutionary optimization of radial basis function classifiers for data mining applications,” IEEE Trans. Syst., Man, Cybern. B: Cybern., vol. 35, no. 5, pp. 928–947, Oct. 2005.

[16] L. N. de Castro, E. R. Hruschka, and R. J. G. B. Campello, “An evolutionary clustering technique with local search to design RBF neural network classifiers,” in Proc. IEEE Int. Joint Conf. Neural Netw., vol. 3. Jul. 2004,pp. 2083–2088.

[17] Z. Q. Zhao and D. S. Huang, “A mended hybrid learning algorithm for radial basis function neural networks to improve generalization capability,” Appl. Math. Model., vol. 31, no. 7, pp. 1271–1281, Jul. 2007.

[18] S. Chen, Y. Wu, and B. L. Luk, “Combined genetic algorithm optimization and regularized orthogonal least squares learning for radial basis function networks,” IEEE Trans. Neural Netw., vol. 10, no. 5, pp. 1239–1243, Sep. 1999.

[19] J. González, I. Rojas, J. Ortega, H. Pomares, F. J. Fernández, and A. F. Días, “Multiobjective evolutionary optimization of the size, shape, and position parameters of radial basis function networks for function approximation,” IEEE Trans. Neural Netw., vol. 14, no. 6, pp. 1478– 1495, Nov. 2003.

[20] B. A. Whitehead and T. D. Choate, “Cooperative-competitive genetic evolution of radial basis function centers and widths for time series prediction,” IEEE Trans. Neural Netw., vol. 7, no. 4, pp. 869–880, Jul. 1996.

[21] B. A. Whitehead, “Genetic evolution of radial basis function coverage using orthogonal niches,” IEEE Trans. Neural Netw., vol. 7, no. 6, pp. 1525–1528, Nov. 1996.

[22] B. A. Whitehead and T. D. Choate, “Evolving space-filling curves to distribute radial basis functions over an input space,” IEEE Trans. Neural Netw., vol. 5, no. 1, pp. 15–23, Jan. 1994.

[23] M. J. Embrechts, B. Szymanski, and M. Sternickel, “Introduction to scientific data mining: Direct kernel methods and applications,” in Computationally Intelligent Hybrid Systems, S. J. Ovaska, Ed. New York: Wiley, 2004, ch. 10, pp. 317–362.

[24] N. Cristianini and J. Shawe-Taylor, An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods, 1st ed. Cambridge, U.K.: Cambridge Univ. Press, 2000.

[25] B. Schölkopf and A. J. Smola, Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond, 1st ed. Cambridge, MA: MIT Press, 2001.

[26] B. Boser, I. Guyon, and V. Vapnik, “A training algorithm for optimal margin classifiers,” in Proc. 5th Annu. ACM Workshop Comput. Learn. Theory, Pittsburgh, PA, 1992, pp. 144–152.

[27] C. Cortes and V. Vapnik, “Support-vector networks,” Mach. Learn., vol. 20, no. 3, pp. 273–297, Sep. 1995.

[28] T.-F. Wu, C.-J. Lin, and R. C. Weng, “Probability estimates for multiclass classification by pairwise coupling,” J. Mach. Learn. Res., vol. 5, pp. 975–1005, 2004.

[29] S. S. Keerthi, . B. Duan, S. K. Shevade, and A. N. Poo, “A fast dual algorithm for kernel logistic regression,” Mach. Learn., vol. 61, nos. 1–3, pp. 151–165, Nov. 2005.

[30] N. Lawrence, M. Seeger, and R. Herbrich, “Fast sparse Gaussian process methods: The informative vector machine,” in Advances in Neural Information Processing Systems, vol. 15. Cambridge, MA: MIT Press,2001, pp. 625–632.

[31] M. E. Tipping, “Sparse Bayesian learning and the relevance vector machine,” J. Mach. Learn. Res., vol. 1, pp. 211–244, Jun.

2001.

[32] H. Chen, P. Tino, and X. Yao, “Probabilistic classification vector machines,” IEEE Trans. Neural Netw., vol. 20, no. 6, pp. 901–914, Jun. 2009.

[33] C.-W. Hsu and C.-J. Lin, “A comparison of methods for multi-class support vector machines,” IEEE Trans. Neural Netw., vol. 13, no. 2, pp. 415–425, Mar. 2002.

[34] L. Narlikar ad A. J. Hartemink, “Sequence features of DNA binding sites reveal structural class of associated transcription factor,” Bioinformatics, vol. 22, no. 2, pp. 157–163, Jan. 2006.

[35] J. Liu, J. Chen, and J. Ye, “Large-scale sparse logistic regression,” in Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining,Paris, France, 2009, pp. 547–556.

[36] N. Subrahmanya and Y. Shin, “Sparse multiple kernel learning for signal processing applications,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 32, no. 5, pp. 788–798, May 2010.

[37] Y.-J. Oyang, S.-C. Hwang, Y.-Y. Ou, C.-Y. Chen, and Z.-W. Chen, “Data classification with radial basis function network based on a novel kerneldensity estimation algorithm,” IEEE Trans. Neural Netw., vol. 16, no.1, pp. 225–236, Jan. 20 05.

[38] I. H. Witten and E. Frank, Data Mining: Practical Machine Learning Tools and Techniques (Data Management Systems), 2nd ed. San Mateo, CA: Morgan Kaufmann, 2005.

[39] S. le Cessie and J. van Houwelingen, “Ridge estimators in logistic regression,” Appl. Stat., vol. 41, no. 1, pp. 191–201, 1992.

[40] P. E. Gill, W. Murray, and M. H. Wright, Practical Optimization. San Francisco, CA: Academic, 1982.

[41] J. Friedman, T. Hastie, and R. Tibshirani, “Additive logistic regressio:A statistical view of boosting,” Ann. Stat., vol. 38, no. 2, pp. 337–374,2000.

[42] N. Landwehr, M. Hall, and E. Frank, “Logistic model trees,” Mach.Learn., vol. 59, nos. 1–2, pp. 161–205, May 2005.

[43] P. J. Angeline, G. M. Sauders, and J. B. Pollack, “An evolutionary algorithm that constructs recurrent neural networks,” IEEE Trans. NeuralNetw., vol. 5, no. 1, pp. 54–65, Jan. 1994.

[44] X. Yao, “Global optimization by evolutionary algorithms,” in Proc. 2nd Aizu Int. Symp. Parallel Algorithms/Architecutre Synthesis, Aizu- Wakamatsu, Japan, 1997, pp. 282–291.

[45] A. C. Martínez-Estudillo, C. Hervás-Martínez, F. J. Martínez-Estudillo, and N. García-Pedrajas, “Hybridization of evolutionary algorithms and local search by means of a clustering method,” IEEE Trans. Syst., Man Cybern. B: Cybern., vol. 36, no. 3, pp. 534–545, Jun. 2006.

[46] F. J. Martínez-Estudillo, C. Hervás-Martínez, P. A. Gutiérrez, and A. C. Martínez-Estudillo, “Evolutionary product-unit neural networks classifiers,” Neurocomputing, vol. 72, nos. 1–2, pp. 548–561, Dec. 2008.

[47] K. Fukunaga, Introduction to Statistical Pattern Recognition, 2nd ed. San Francisco, CA: Academic, 1999.

[48] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing,” Science, vol. 220, no. 4598, pp. 671–680, May1983.

[49] I. Rechenberg, Evolutionsstrategie: Optimierung Technischer Systeme Nach Prinzipien der Biologischen Evolution. Stuttgart, Germany: Frommann-Holzboog, 1973.

[50] Y. Freund and R. E. Schapire, “Experiments with a new boosting algorithm,” in Proc. 13th Int. Conf. Mach. Learn., 1996, pp. 148–156.

[51] T. Hastie, R. Tibshirani, and J. H. Friedman, The Elements of Statistical Learning. New York: Springer-Verlag, 2001.

[52] S. Ventura, C. Romero, A. Zafra, J. A. Delgado, and C. Hervas, “JCLEC A Java framework for evolutionary computation,” Soft Comput., vol. 12, no. 4, pp. 381–392, Oct. 2008.

[53] C.-C. Chang and C.-J. Lin. (2001). LIBSVM: A Library for Support Vector Machines [Online]. Available: http://www.csie.ntu.edu.tw/ ~cjlin/libsvm

[54] S. South, J. Qi, and D. P. Lusch, “Optimal classification methods for mapping agricultural tillage practices,” Remote Sens. Environ., vol. 91, no. 1, pp. 90–97, May 2004.

[55] J. Peña-Barragán, F. Léopez-Granados, M. Jurado-Expéosito, and L. García-Torres, “Spectral discrimination of ridolfia segetum and sunflower as affected by phenological stage,” Weed Res., vol. 46, no. 1, pp. 10–21, Feb. 2006. [56] J. Demšar, “Statistical comparisons of classifiers over multiple data sets,” J. Mach. Learn. Res., vol. 7, pp. 1–30, 2006.[57] M. Friedman, “A comparison of alternative tests of significance for the problem of m rankings,” Ann. Math. Stat., vol. 11, no. 1, pp. 86–92, Mar. 1940.

TITLE: Optimizing Single-Trial EEG Classification by Stationary Matrix Logistic Regression in Brain–Computer Interface

Zeng, H., & Song, A. (2016). Optimizing single-trial EEG classification by stationary matrix logistic regression in brain–computer interface. *IEEE transactions on neural networks and learning systems, 27*(11), 2301-2313.

REFERENCES:

[1] G. Pfurtscheller, G. R. Muller-Putz, R. Scherer, and C. Neuper, “Rehabilitation with brain-computer interface systems,” Computer, vol. 41, no. 10, pp. 58–65, Oct. 2008.
[2] J. R. Wolpaw, D. J. McFarland, and T. M. Vaughan, “Brain-computer interface research at the Wadsworth center,” IEEE Trans. Rehabil. Eng., vol. 8, no. 2, pp. 222–226, Jun. 2000.
[3] L. C. Parra et al., “Spatiotemporal linear decoding of brain state,” IEEE Signal Process. Mag., vol. 25, no. 1, pp. 107–115, Jan. 2008.
[4] R. Yang, A. Song, and B. Xu, “Feature extraction of motor imagery eeg based on wavelet transform and higher-order statistics,” Int. J. Wavelets, Multiresolution Inf. Process., vol. 8, no. 3, pp. 373–384, 2010.
[5] S. Sanei, S. Ferdowsi, K. Nazarpour, and A. Cichocki, “Advances in electroencephalography signal processing [life sciences],” IEEE Signal Process. Mag., vol. 30, no. 1, pp. 170–176, Jan. 2013.
[6] Z. J. Koles, “The quantitative extraction and topographic mapping of the abnormal components in the clinical EEG,” Electroencephalogr. Clin. Neurophysiol., vol. 79, no. 6, pp. 440–447, Dec. 1991.
[7] R. Tomioka, K. Aihara, and K.-R. Müller, “Logistic regression for single trial EEG classification,” in Proc. 20th Annu. Conf. Neural Inf. Process. Syst., Vancouver, BC, Canada, Dec. 2006, pp. 1377–1384.
[8] B. Blankertz, R. Tomioka, S. Lemm, M. Kawanabe, and K.-R. Müller, “Optimizing spatial filters for robust EEG single-trial analysis,” IEEE Signal Process. Mag., vol. 25, no. 1, pp. 41–56, Jan. 2008.
[9] W. Samek, M. Kawanabe, and K.-R. Müller, “Divergence-based framework for common spatial patterns algorithms,” IEEE Rev. Biomed. Eng., vol. 7, pp. 50–73, Apr. 2014.
[10] C. Christoforou, R. Haralick, P. Sajda, and L. C. Parra, “Second-order bilinear discriminant analysis,” J. Mach. Learn. Res., vol. 11, no. 1, pp. 665–685, 2010.
[11] J. D. R. Farquhar, “A linear feature space for simultaneous learning of spatio-spectral filters in BCI,” Neural Netw., vol. 22, no. 9, pp. 1278–1285, 2009.
[12] R. Tomioka and K.-R. Müller, “A regularized discriminative framework for EEG analysis with application to brain–computer interface,” NeuroImage, vol. 49, no. 1, pp. 415–432, Jan. 2010.
[13] R. Tomioka and K. Aihara, “Classifying matrices with a spectral regularization,” in Proc. 24th Int. Conf. Mach. Learn., Corvallis, OR, USA, Jun. 2007, pp. 895–902.
[14] P. von Bünau, F. C. Meinecke, F. C. Király, and K.-R. Müller, “Finding stationary subspaces in multivariate time series,” Phys. Rev. Lett., vol. 103, no. 21, p. 214101, Nov. 2009.
[15] B. Blankertz, M. Kawanabe, R. Tomioka, F. Hohlefeld, K.-R. Müller, and V. V. Nikulin, “Invariant common spatial patterns: Alleviating nonstationarities in brain-computer interfacing,” in Proc. 21st Annu. Conf. Neural Inf. Process. Syst., Vancouver, BC, Canada, Dec. 2007, pp. 113–120.
[16] W. Samek, C. Vidaurre, K.-R. Müller, and M. Kawanabe, “Stationary common spatial patterns for brain–computer interfacing,” J. Neural Eng., vol. 9, no. 2, p. 026013, 2012.
[17] M. Arvaneh, C. Guan, K K. Ang, and C. Quek, “Optimizing spatial filters by minimizing within-class dissimilarities in electroencephalogrambased brain–computer interface,” IEEE Trans. Neural Netw. Learn. Syst., vol. 24, no. 4, pp. 610–619, Apr. 2013.
[18] A. Beck and M. Teboulle, “A fast iterative shrinkage-thresholding algorithm for linear inverse problems,” SIAM J. Imag. Sci., vol. 2, no. 1, pp. 183–202, 2009.
[19] J.-F. Cai, E. J. Candès, and Z. Shen, “A singular value thresholding algorithm for matrix completion,” SIAM J. Optim., vol. 20, no. 4, pp. 1956–1982, 2010.
[20] D. P. Bertsekas, Nonlinear Programming. Belmont, MA, USA: Athena Scientific, 1999.
[21] V. Arsigny, P. Fillard, X. Pennec, and N. Ayache, “Log-Euclidean metrics for fast and simple calculus on diffusion tensors,” Magn. Reson. Med., vol. 56, no. 2, pp. 411–421, Aug. 2006.

[22] H. Zhang, H. Yang, and C. Guan, “Bayesian learning for spatial filtering in an EEG-based brain–computer interface,” IEEE Trans. Neural Netw. Learn. Syst., vol. 24, no. 7, pp. 1049–1060, Jul. 2013.
[23] N. Parikh and S. Boyd, “Proximal algorithms,” Found. Trends Optim., vol. 1, no. 3, pp. 123–231, 2014.
[24] Y. Cui, C. Leng, and D. Sun, “Sparse estimation of high-dimensional correlation matrices,” Comput. Statist. Data Anal., to be published.
[25] X. V. Doan, K.-C. Toh, and S. Vavasis, “A proximal point algorithm for sequential feature extraction applications,” SIAM J. Sci. Comput.,vol. 35, no. 1, pp. A517–A540, 2013.
[26] G. Dornhege, B. Blankertz, G. Curio, and K.-R. Müller, “Boosting bit rates in noninvasive EEG single-trial classifications by feature combination and multiclass paradigms,” IEEE Trans. Biomed. Eng., vol. 51, no. 6, pp. 993–1002, Jun. 2004.
[27] K. K. Ang, Z. Y. Chin, C. Wang, C. Guan, and H. Zhang, “Filter bank common spatial pattern algorithm on BCI competition IV datasets 2a and 2b,” Frontiers Neurosci., vol. 6, no. 1, p. 39, 2012.
[28] M. Tangermann et al., “Review of the BCI competition IV,” Frontiers Neurosci., vol. 6, no. 1, p. 55, 2012.
[29] B. Blankertz et al., “The BCI competition III: Validating alternative approaches to actual BCI problems,” IEEE Trans. Neural Syst. Rehabil. Eng., vol. 14, no. 2, pp. 153–159, Jun. 2006.
[30] H. Zeng, A. Song, R. Yan, and H. Qin, “EOG artifact correction from EEG recording using stationary subspace analysis and empirical mode decomposition,” Sensors, vol. 13, no. 11, pp. 14839–14859, 2013.
[31] W. Wu, X. Gao, B. Hong, and S. Gao, “Classifying single-trial EEG during motor imagery by iterative spatio-spectral patterns learning (ISSPL),” IEEE Trans. Biomed. Eng., vol. 55, no. 6, pp. 1733–1743, Jun. 2008.\

TITLE : Orthogonal Decision Trees

Kargupta, K., Park, B. H., & Dutta, H. (2006). Orthogonal decision trees. IEEE Transactions on Knowledge and Data Engineering, 18(8), 1028-1042.

REFERENCES:

[1] J.R. Quinlan, “Induction of Decision Trees,” Machine Learning, vol. 1, no. 1, pp. 81-106, 1986.

[2] Y. Freund, “Boosting a Weak Learning Algorithm by Majority,” Information and Computation, vol. 121, no. 2, pp. 256-285, 1995.

[3] H. Drucker and C. Cortes, “Boosting Decision Trees,” Advances in Neural Information Processing Systems, vol. 8, pp. 479-485, 1996.

[4] L. Breiman, “Bagging Predictors,” Machine Learning, vol. 24, no. 2, pp. 123-140, 1996.

[5] D. Wolpert, “Stacked Generalization,” Neural Networks, vol. 5, pp. 241-259, 1992.

[6] L. Breiman, “Random Forests,” Machine Learning, vol. 45, no. 1, pp. 5-32, 2001.

[7] W. Fan, S. Stolfo, and J. Zhang, “The Application of Adaboost for Distributed, Scalable, and On-Line Learning,” Proc. Fifth ACM SIGKDD Int’l Conf. Knowledge Discovery and Data Mining, 1999.

[8] W.N. Street and Y. Kim, “A Streaming Ensemble Algorithm (Sea) for Large-Scale Classificaiton,” Proc. Seventh ACM SIGKDD Int’l Conf. Knowledge Discovery and Data Mining, 2001.

[9] H. Kargupta and B. Park, “A Fourier Spectrum-Based Approach to Represent Decision Trees for Mining Data Streams in Mobile Environments,” IEEE Trans. Knowledge and Data Eng., vol. 16, no. 2, pp. 216-229, 2002.

[10] B. Park, A. R, and H. Kargupta, “A Fourier Analysis-Based Approach to Learn Classifier from Distributed Heterogeneous Data,” Proc. First SIAM Int’l Conf. Data Mining, 2001.

[11] B.H. Park and H. Kargupta, “Constructing Simpler Decision Trees from Ensemble Models Using Fourier Analysis,” Proc. Seventh Workshop Research Issues in Data Mining and Knowledge Discovery, pp. 18-23, 2002.

[12] F. Chung, Spectral Graph Theory. Providence, R.I.: Am. Math. Soc., 1994.

[13] H. Kargupta and B. Park, “Mining Time-Critical Data Stream Using the Fourier Spectrum of Decision Trees,” Proc. IEEE Int’l Conf. Data Mining, pp. 281-288, 2001.

[14] H. Kargupta, B. Park, S. Pittie, L. Liu, D. Kushraj, and K. Sarkar, “Mobimine: Monitoring the Stock Market from a PDA,” ACM SIGKDD Explorations, vol. 3, no. 2, pp. 37-46, Jan. 2002.

[15] N. Linial, Y. Mansour, and N. Nisan, “Constant Depth Circuits, Fourier Transform, and Learnability,” J. ACM, vol. 40, pp. 607-620, 1993.

[16] E. Kushilevitz and Y. Mansour, “Learning Decision Trees Using the Fourier Spectrum,” SIAM J. Computing, vol. 22, no. 6, pp. 1331- 1348, 1993.

[17] D. Goldberg, “Genetic Algorithms and Walsh Functions: Part I, a Gentle Introduction,” Complex Systems, vol. 3, no. 2, pp. 129-152, 1989.

[18] C.J. Merz and M.J. Pazzani, “A Principal Components Approach to Combining Regression Estimates,” Machine Learning, vol. 36, nos. 1-2, pp. 9-32, 1999.

[19] C. Merz and M. Pazzani, “A Principal Components Approach to Combining Regression Estimates,” Machine Learning, vol. 36, pp. 9- 32, 1999.

[20] H. Kargupta, B. Park, D. Hershberger, and E. Johnson, “Collective Data Mining: A New Perspective towards Distributed Data Mining,” Advances in Distributed and Parallel Knowledge Discovery, H. Kargupta and P. Chan, eds., AAAI/MIT Press, 2000.

TITLE : Compression and Aggregation for Logistic Regression Analysis in Data Cubes

Xi, R., Lin, N., & Chen, Y. (2009). Compression and aggregation for logistic regression analysis in data cubes. IEEE transactions on knowledge and data engineering, 21(4), 479-492.

REFERENCES:

[1] A. Agresti, An Introduction to Categorical Data Analysis. Wiley, 1996.

[2] A. Agresti, Categorical Data Analysis, second ed. John Wiley & Sons, 2002.

[3] D. Barbara and X. Wu, “Loglinear-Based Quasi Cubes,” J. Intelligent Information Systems, vol.16, pp. 255-276, 2001.

[4] C.R. Charig, D.R. Webb, S.R. Payne, and O.E. Wickham, “Comparison of Treatment of Renal Calculi by Operative Surgery, Percutaneous Nephrolithotomy, and Extracorporeal Shock Wave Lithotripsy,” British Medical J., vol. 292, pp. 882-897, 1986.

[5] B. Chen, L. Chen, Y. Lin, and R. Ramakrishnan, “Prediction Cubes,” Proc. 31st Int’l Conf. Very Large Data Bases (VLDB ’05), pp. 982-993, 2005.

[6] K. Chen, I. Hu, and Z. Ying, “Strong Consistency of Maximum Quasi-Likelihood Estimators in Generalized Linear Models with Fixed and Adaptive Designs,” The Annals of Statistics, vol. 27, pp. 1155-1163, 1999.

[7] Y. Chen, G. Dong, J. Han, J. Pei, B. Wah, and J. Wang, “Regression Cubes with Lossless Compression and Aggregation,” IEEE Trans. Knowledge and Data Eng., vol. 18, pp. 1585-1599, 2006.

[8] Y. Chen, G. Dong, J. Han, J. Pei, B.W. Wah, and J. Wang, “OLAPing Stream Data: Is It Feasible?” Proc. ACM SIGMOD ’02 Workshop Research Issues in Data Mining and Knowledge Discovery, pp. 53-58, 2002.

[9] Y. Chen, G. Dong, J. Han, B.W. Wah, and J. Wang, “Multi- Dimensional Regression Analysis of Time-Series Data Streams,” Proc. 28th Int’l Conf. Very Large Data Bases (VLDB ’02), pp. 323-334, 2002.

[10] Y. Chow and H. Teicher, Probability Theory, second ed. Springer, 1988.

[11] L. Fahrmeir and H. Kaufmann, “Consistency and Asymptotic Normality of the Maximum Likelihood Estimator in Generalized Linear Models,” The Annals of Statistics, vol. 13, pp. 342-368, 1985.

[12] “Centers for Disease Control and Prevention,” Behavioral Risk Factor Surveillance System Survey Data. US Dept. of Health and Human Services, Centers for Disease Control and Prevention, 2006.

[13] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh, “Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross- Tab and Sub-Totals,” Data Mining and Knowledge Discovery, vol. 1, pp. 29-54, 1997.

[14] J. Han, Y. Chen, G. Dong, J. Pei, B.W. Wah, J. Wang, and Y. Cai, “Stream Cube: An Architecture for Multi-Dimensional Analysis of Data Streams,” Distributed and Parallel Databases, vol. 18, no. 2, pp. 173-197, 2005.

[15] V. Harinarayan, A. Rajaraman, and J.D. Ullman, “Implementing Data Cubes Efficiently,” Proc. ACM SIGMOD ’96, pp. 205-216, 1996. [16] S.A. Julious and M.A. Mullee, “Confounding and Simpson’s

[17] H. Lenz and B. Thalheim, “OLAP Databases and Aggregation Functions,” Proc. 13th Int’l Conf. Scientific and Statistical Database Management (SSDBM ’01), pp. 91-100, 2001.

[18] C. Liu, M. Zhang, M. Zheng, and Y. Chen, “Step-by-Step Regression: A More Efficient Alternative for Polynomial Multiple Linear Regression in Stream Cube,” Proc. Seventh Pacific-Asia Conf. Knowledge Discovery and Data Mining (PAKDD ’03), pp. 437-448, 2003.

[19] P. McCullagh and J.A. Nelder, Generalized Linear Models, second ed. Chapman and Hall, 1989.

[20] T. Palpanas, N. Koudas, and A.O. Mendelzon, “Using Datacube Aggregates for Approximate Querying and Deviation Detection,” IEEE Trans. Knowledge and Data Eng., vol. 17, no. 11, pp. 1465-1477, Nov. 2005.

[21] S. Pang, S. Ozawa, and N. Kasabov, “Incremental Linear Discriminant Analysis for Classification of Data Streams,” IEEE Trans. Systems, Man, and Cybernetics, Part B, vol. 35,no. 5, pp. 905-914, 2005.

[22] G. Sathe and S. Sarawagi, “Intelligent Rollups in Multidimen- sional OLAP Data,” Proc. 27th Int’l Conf. Very Large Data Bases (VLDB ’01), pp. 531-540, 2001.

[23] P. Vassiliadis, “Modeling Multidimensional Databases, Cubes and Cube Operations,” Proc. 10th Int’l Conf. Scientific and Statistical Database Management (SSDBM ’98), pp. 53-62, 1998.

TITLE : Minimax Sparse Logistic Regression for Very High-Dimensional Feature Selection

Tan, M., Tsang, I. W., & Wang, L. (2013). Minimax sparse logistic regression for very high-dimensional feature selection. IEEE transactions on neural networks and learning systems, 24(10), 1609-1622.

REFERENCES:

[1] Liblinear. (2008) [Online]. Available: http://c2inet.sce.ntu.edu.sg/mingkui/ fgm.htm

[2] FGM. (2010) [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/ liblinear/

[3] K. L. Ayers and H. J. Cordell, “SNP selection in genome-wide and can- didate gene studies via penalized logistic regression,” Genet. Epidemiol., vol. 34, no. 8, pp. 879–891, 2010.

[4] E. J. Candes, M. Wakin, and S. Boyd, “Enhancing sparsity by reweighted l1 minimization,” J. Fourier Anal.

Appl., vol. 14, no. 5, pp. 877–905, 2007.

[5] Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, and C.-J. Lin, “Training and testing low-degree polynomial data mappings via linear SVM,” J. Mach. Learn. Res., vol. 11, pp. 1471–1490, Apr. 2010.

[6] A. Dasgupta, P. Drineas, B. Harb, V. Josifovski, and M. W. Mahoney, “Feature selection methods for text classification,” in Proc. 13th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2007, pp. 230–239.

[7] N. Ding and S. Vishwanathan, “T-logistic regression,” in Advances in Neural Information Processing Systems. Cambridge, MA, USA: MIT Press, 2010.

[8] H. Fei and J. Huan, “Boosting with structure information in the func- tional space: An application to graph classification,” in Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2010, 643–652.

[9] Y. Grandvalet and S. Canu, “Adaptive scaling for feature selection in SVMs,” in Advances in Neural Information Processing Systems. Cambridge, MA, USA: MIT Press, 2002.

[10] I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, Feature Extraction, Foundations and Applications. New York, NY, USA: Springer-Verlag, 2006.

[11] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, “Gene selection for cancer classification using support vector machines,” Mach. Learn., vol. 46, nos. 1–3, pp. 389–422, 2002.

[12] M. Heath, Scientific Computing-An Introductory Survey. New York, NY, USA: McGraw-Hill, 1997.

[13] S. S. Keerthi, K. Duan, S. K. Shevade, and A. Poo, “A fast dual algorithm for kernel logistic regression,” Mach. Learn., vol. 61, nos. 1–3, pp. 151–165, 2005.

[14] J. Liao and K.-V. Chin, “Logistic regression for disease classification using microarray data: Model selection in a large p and small n case,” Bioinformatics, vol. 23, no. 15, pp. 1945–1951, 2007.

[15] C.-J. Lin, R. C. Weng, and S. S. Keerthi, “Trust region Newton method for large-scale logistic regression,” J. Mach. Learn. Res., vol. 9, pp. 627–650, Jun. 2008.

[16] D. Lin, D. P. Foster, and L. H. Ungar, “A risk ratio comparison of L and L1 penalized regressions,” Dept. Stat., Univ. Pennsylvania, Philadelphia, PA, USA, Tech. Rep., 2010.

[17] Z. Liu, F. Jiang, G. Tian, S. Wang, F. Sato, S. J. Meltzer, and M. Tan, “Sparse logistic regression with lp penalty for biomarker identification,” Stat. Appl. Genet. Molecular Biol., vol. 6, no. 1, pp. 1544–6115, 2007.

[18] A. C. Lozano, G. Swirszcz, and N. Abe, “Group orthogonal matching pursuit for logistic regression,” in Proc. 14th Int. Conf. Artif. Intell. Stat., 2011, pp. 452–460.

[19] Z. Q. Luo and P. Tseng, “On the convergence of the coordinate descent method for convex differentiable minimization,” J. Optim. Theory Appl., vol. 72, no. 1, pp. 7–35, 1992.

[20] A. Mutapcic and S. Boyd, “Cutting-set methods for robust convex optimization with pessimizing oracles, Optim. Methods Softw., vol. 24, no. 3, pp. 381–406, 2009.

[21] A. Nedic and A. Ozdaglar, “Subgradient methods for saddle-point problems,” J. Optim. Theory Appl., vol 142, no. 1, pp. 205–228, 2009.

[22] A. Nemirovski, “Prox-method with rate of convergence O(1/t) for variational inequalities with lipschitz continuous monotone operators and smooth convex-concave saddle point problems,” SIAM J. Optim., vol. 15, no. 1, pp. 229–251, 2005.

[23] E. Y. Pee and J. O. Royset, “On solving large-scale finite minimax prob- lems using exponential smoothing,” J. Optim. Theory Appl., vol. 148, no. 2, pp. 390–421, 2010.

[24] A. Rakotomamonjy, F. Bach, Y. Grandvalet, and S. Canu, “SimpleMKL,” J. Mach. Learn. Res., vol. 9, pp. 2491–2521, Nov. 2008.

[25] K.-Q. Shen, C.-J. Ong, X.-P. Li, and E. P. Wilder-Smith, “Feature selection via sensitivity analysis of SVM probabilistic outputs,” Mach. Learn., vol. 70, no. 1, pp. 1–20, 2008.

[26] J. Shi, W. Yin, S. Osher, and P. Sajda, “A fast hybrid algorithm for large- scale l1-regularized logistic regression,” J. Mach. Learn. Res., vol. 11, pp. 713–741, Feb. 2010.

[27] M. Tan, I. Tsang, and L. Wang, “Learning sparse svm for feature selection on very high dimensional datasets,” in Proc. 27th Int. Conf. Mach. Learn., 2010, pp. 1047–1054.

[28] M. Sion, “On general minimax theorems,” Pacific J. Math., vol. 8, no. 1, pp. 171–176, 1958.

[29] A. Tewari, P. Ravikumar, and I. S. Dhillon, “Greedy algorithms for structurally constrained high dimensional problems,” in Advances in Neural Information Processing Systems. Cambridge, MA, USA: MIT Press, 2011.

[30] P. T seng, “On accelerated proximal gradient methods for convex- concave optimization,” Dept. Math., Univ. Washington, Seattle, WA, USA, Tech. Rep., 2008.

[31] M. Varma and B. R. Babu, “More generality in efficient multiple kernel learning,” in Proc. 26th Annu. Int. Conf. Mach. Learn., 2009, pp. 1065–1072.

[32] S. Xu, “Smoothing method for minimax problems,” J. Optim. Theory App., vol. 20, no. 3, pp. 267–279, 2001.

[33] H.-F. Yu, F.-L. Huang, and C.-J. Lin, “Dual coordinate descent methods for logistic regression and maximum entropy models,” Mach. Learn., vol. 85, nos. 1–2, pp. 41–75, 2010.

[34] L. Yu and H. Liu, “Efficient feature selection via analysis of rele- vance and redundancy,” J. Mach. Learn. Res., vol. 5, pp. 1205–1224, Dec. 2004.

[35] G.-X. Yuan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin, “A comparison of optimization methods and software for large-scale L 1 -regularized linear classification,” J. Mach. Learn. Res., vol. 11, pp. 3183–3234, Jan. 2010.

[36] G.-X. Yuan, C.-H. Ho, and C.-J. Lin, “An improved GLMNET for L1-regularized logistic regression and support vector machines,” Dept. Comput. Sci., Nat. Taiwan Univ., Taipei, Taiwan, Tech. Rep., 2011.

[37] T. Zhang, “Analysis of multi-stage convex relaxation for sparse regular- ization,” J. Mach. Learn. Res., vol. 11, pp. 1081–1107, Mar. 2010.

[38] Z.-H. Zhou and M.-L. Zhang, “Multi-instance multi-label learning with application to scene classification,” in Advances in Neural Information Processing Systems. Cambridge, MA, USA: MIT Press, 2007.

[39] J. Zhu and T. Hastie, “Classification of gene microarrays by penal- ized logistic regression,” Biostatistics, vol. 5, no. 3, pp. 427–443, 2004.

[40] Z. Zhu, Y. S. Ong, and M. Dash, “Markov blanket-embedded genetic algorithm for gene selection,” Pattern Recognit., vol. 40, no. 11, pp. 3236–3248, 2007