EECS 545 Machine Learning Homework 2

Tsung-Hung Yao

February 9, 2016

1) Linear Regression

In this problem, you will implement linear regression (using polynomial features) to reproduce curves similar to those you have seen in lecture, exploring how various choices of model-complexity parameters affect training and test error.

Problem (a). The training and testing data is provided to you, named **train_graphs_f16_autopilot_cruise.csv** and **test_graphs_f16_autopilot_cruise.csv**. Every row in these csv files correspond to a single datapoint $\mathbf{x}$, $\mathbf{t}$. Columns of this graphs are named id, rolling speed, elevation speed, elevation jerk, elevation, roll, elevation acceleration, controller input. Your task is to predict controller input from rolling speed, elevation speed, elevation jerk, elevation, roll, and elevation acceleration. Note that id is a primary key, and is unique to each datapoint, so you'd want to remove it from the feature set. Ponder why this is necessary, and what would happen otherwise. Also, you might need to append an additional feature that is constant for all data points, for example a feature with a value of 1, in order to model the intercept term.

(i) Fit the data by applying the psuedo-inverse approach of linear regression using x_j^i, $i = 1, \ldots, M$ as features (try $M = 1, 2, \ldots, 6$), where x_j represents the j^{th} component of vector $\mathbf{x}$. In other words, you will need to raise every element in vector $\mathbf{x}$ to the power of i, for $i = 1, 2, \ldots, M$ to get the set of features. For example, if $\mathbf{x} = [x_1, x_2]'$ and $M = 2$, then you'd have 4 features, $x_1^1, x_1^2, x_2^1, x_2^2$ along with an additional feature 1 for the intercept term. Plot training error and test error as Root Mean Square Error (RMSE) against M, the order of the polynomial features. Name the generated plot 1.png.

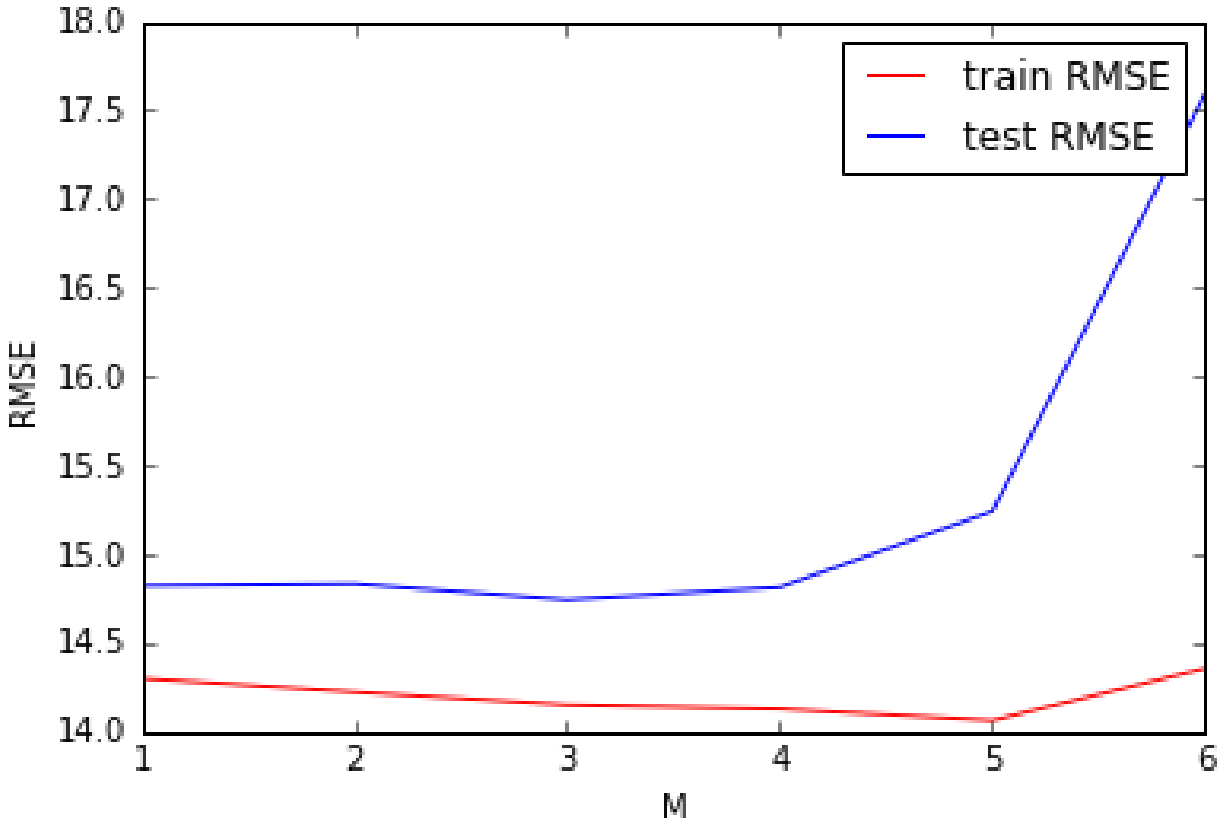

Figure 1.png

(ii) Fit the data by using psuedo-inverse approach of regularized linear regression. For this, you need to use all polynomial features (i.e. $M = 6$), and choose values of $\ln \lambda$ using a sweep as follows: $-40, -39, \ldots, 18, 19, 20$. Plot training error and test error as Root Mean Square Error (RMSE) against $\ln \lambda$, the regularization coefficient. Name the generated plot 2.png.

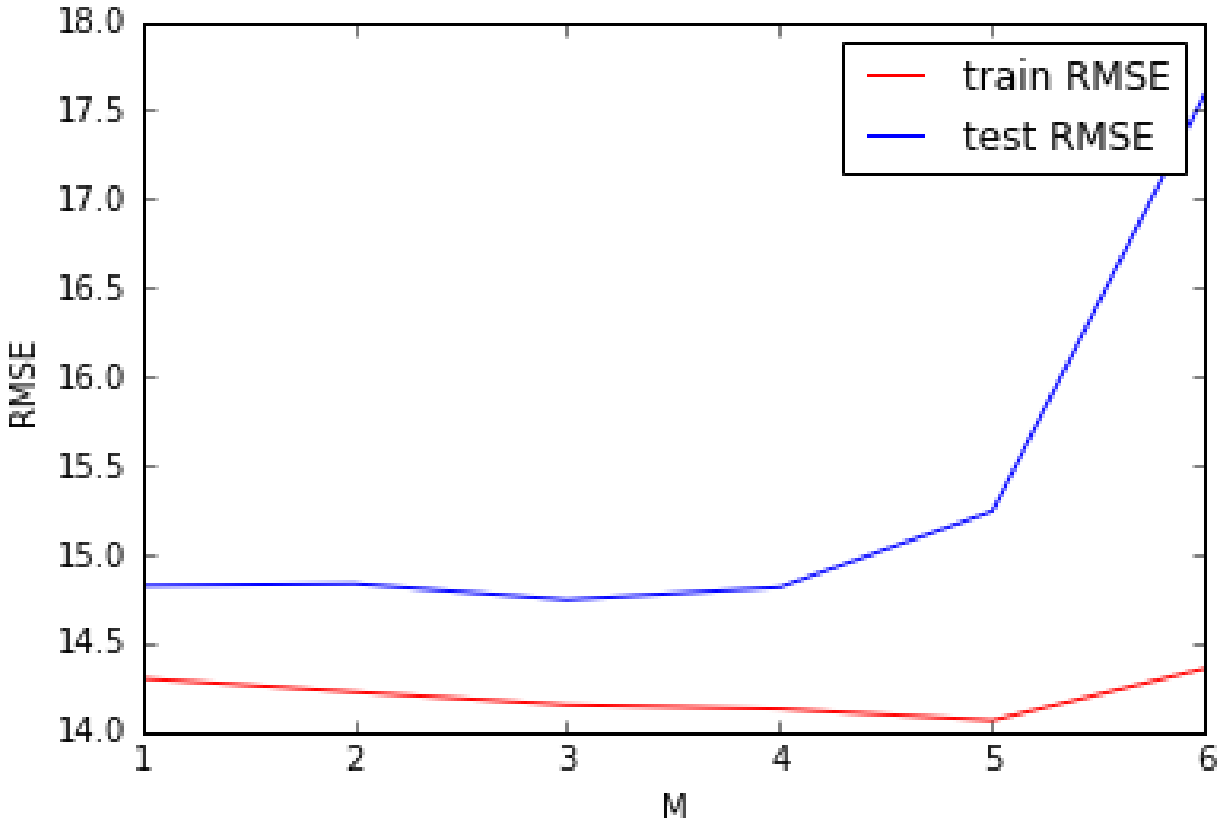

Figure 2.png

Problem (b). Because locally weighted linear regression is computationally more expensive, this problem is provided with a smaller test dataset. The training and testing data is provided to you, named **train_graphs_f16_autopilot_cruise.csv** and **test_locreg_f16_autopilot_cruise.csv**. For this part, you will not create the features by yourself, but rather use the features given in the dataset.

Fit the data by applying the psuedo-inverse approach for solving locally weighted linear regression. Use as weights the similarity between the point of interest and the example point as defined by

$$r_i(\mathbf{x}) = \exp(-\frac{||\mathbf{x}^{(i)} - \mathbf{x}||^2}{2\tau^2})$$

and choose values of τ using a geometric sweep as follows: Generate 10 evenly spaced numbers on a logarithmic scale between 2^{-2} and 2^1 (start and end values inclusive). Plot only the test error as Root Mean Square Error (RMSE) against τ, a hyperparameter that affects the performance of the algorithm. Name the generated plot 3.png. Note: You might want to think about formulating efficient matrix multiplications in your code to implement locally weighted linear regression for this problem.

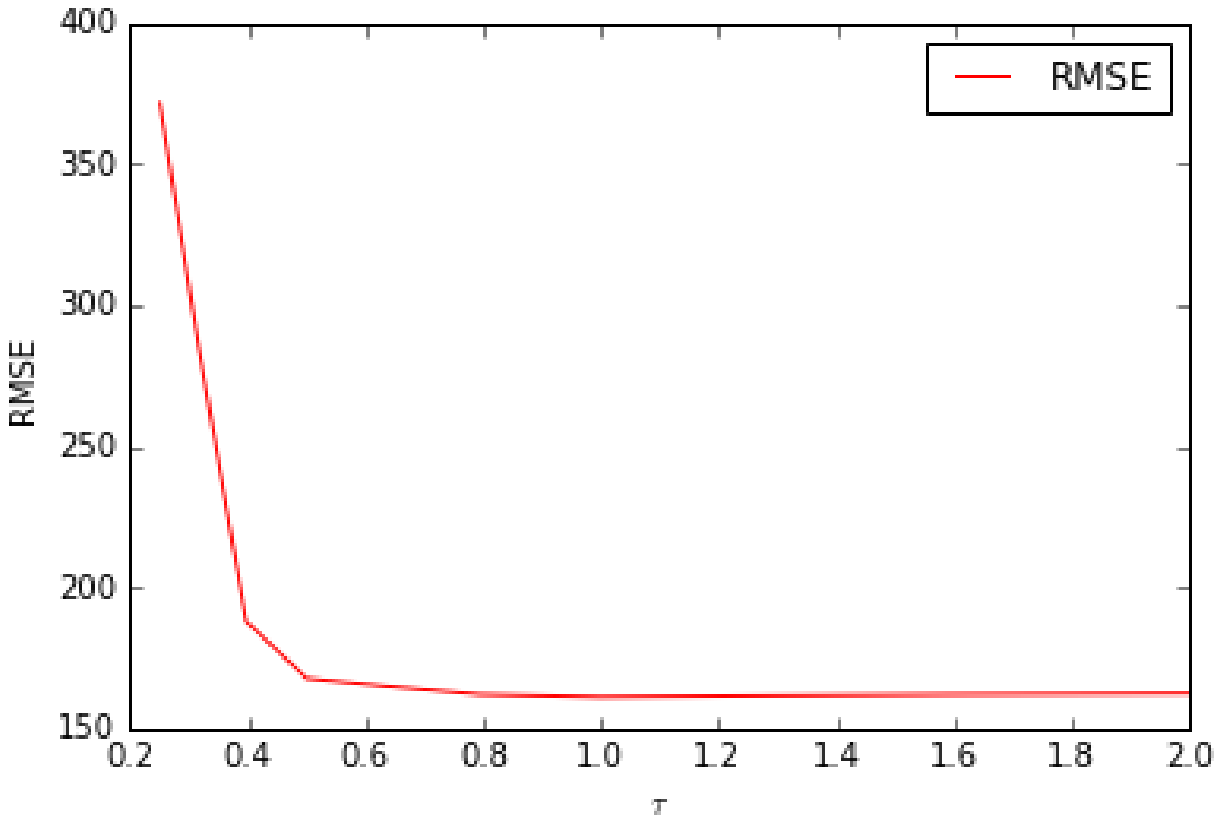

Figure 3.png

2) Open Kaggle challenge

You can use any algorithm and design any features to perform regression on the Steel Ultimate Tensile Strength Dataset. All details for this project are included in the Kaggle webpage **https://inclass.kaggle.com/c/steel-ultimate-tensile-strength**. This problem will be graded separately based on your performance on the private leaderboard.

Name: Tsung-HungYao
With score 7.199852

3) Weighted Linear Regression

Consider a linear regression problem in which we want to weigh different training examples differently. Specifically, suppose we want to minimize

$$E_D(w) = \frac{1}{2}\sum_{i=1}^{N} r_i(w^\top x_i - t_i)^2. \tag{1}$$

In class, we worked out what happens for the case where all the weights (r_i's) are one. In this problem, we will generalize some of those ideas to the weighted setting. In other words, we will allow the weights r_i to be different for each of the training examples.

Problem (a). Show that $E_D(w)$ can also be written as

$$E_D(w) = (Xw - t)^\top R(Xw - t) \tag{2}$$

for an appropriate definition of X, t and R, i.e., write the derivation as part of your submitted answer. State these three quantities clearly as part of your answer (i.e., specify the form, matrix, vector, scalar, their dimensions, and how to build them from the x_i's, t_i's and r_i's.

Proof.

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(M)} \\ x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(M)} \\ \vdots & \vdots & \ddots & \vdots \\ x_N^{(1)} & x_N^{(2)} & \cdots & x_N^{(M)} \end{bmatrix} = \begin{bmatrix} \vec{x}_1^\top \\ \vec{x}_2^\top \\ \vec{x}_3^\top \\ \vdots \\ \vec{x}_N^\top \end{bmatrix} \implies dim(\mathbf{X}) = N \times M$$

where $\vec{x}_i^\top = \begin{bmatrix} x_i^{(1)} & x_i^{(2)} & \cdots & x_i^{(M)} \end{bmatrix}$ means M features of the ith data point.

$$\vec{w}^\top = \begin{bmatrix} w^{(1)} & w^{(2)} & w^{(3)} & \cdots w^{(M)} \end{bmatrix} \implies dim(\vec{w}) = M \times 1$$

which $w^{(i)}$ means the coefficient of ith feature

$$\vec{t}^\top = \begin{bmatrix} t_1 & t_2 & t_3 & \cdots t_N \end{bmatrix} \implies dim(\vec{t}) = N \times 1$$

$$\mathbf{X}\vec{w} - \vec{t} = \begin{bmatrix} \vec{x}_1^\top \\ \vec{x}_2^\top \\ \vec{x}_3^\top \\ \vdots \\ \vec{x}_N^\top \end{bmatrix} \vec{w} - \vec{t} = \begin{bmatrix} \vec{x}_1^\top \vec{w} - t^{(1)} \\ \vec{x}_2^\top \vec{w} - t^{(2)} \\ \vec{x}_3^\top \vec{w} - t^{(3)} \\ \vdots \\ \vec{x}_N^\top \vec{w} - t^{(N)} \end{bmatrix} \implies dim(\mathbf{X}\vec{w}) = (N \times M) \times (M \times 1) = N \times 1$$

$$\mathbf{R} = \frac{1}{2} \begin{bmatrix} r_1 & 0 & 0 & \cdots & 0 \\ 0 & r_2 & 0 & \cdots & 0 \\ 0 & 0 & r_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & r_N \end{bmatrix}$$

$$(\mathbf{X}\vec{w} - \vec{t})^\top_{1\times N} \mathbf{R}_{N\times N} (\mathbf{X}\vec{w} - \vec{t})_{N\times 1}$$

$$= \tfrac{1}{2} \begin{bmatrix} \vec{w}^\top \vec{x_1} - t^{(1)} & \vec{w}^\top \vec{x_2} - t^{(2)} & \cdots & \vec{w}^\top \vec{x_N} - t^{(1)} \end{bmatrix} \begin{bmatrix} r_1 & 0 & 0 & \cdots & 0 \\ 0 & r_2 & 0 & \cdots & 0 \\ 0 & 0 & r_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & r_N \end{bmatrix} \begin{bmatrix} \vec{x}_1^\top \vec{w} - t^{(1)} \\ \vec{x}_2^\top \vec{w} - t^{(2)} \\ \vec{x}_3^\top \vec{w} - t^{(3)} \\ \vdots \\ \vec{x}_N^\top \vec{w} - t^{(N)} \end{bmatrix}$$

$$= \frac{1}{2}\begin{bmatrix} r_1(\vec{w}^\top \vec{x_1} - t^{(1)}) & r_2(\vec{w}^\top \vec{x_2} - t^{(2)}) & \cdots & r_i(\vec{w}^\top \vec{x_i} - t^{(i)}) & \cdots & r_N(\vec{w}^\top \vec{x_N} - t^{(1)}) \end{bmatrix} \begin{bmatrix} \vec{x}_1^\top \vec{w} - t^{(1)} \\ \vec{x}_2^\top \vec{w} - t^{(2)} \\ \vec{x}_3^\top \vec{w} - t^{(3)} \\ \vdots \\ \vec{x}_N^\top \vec{w} - t^{(N)} \end{bmatrix}$$

$= \frac{1}{2} r_1(\vec{w}^\top \vec{x_1} - t^{(1)})^2 + r_2(\vec{w}^\top \vec{x_2} - t^{(2)})^2 \cdots r_i(\vec{w}^\top \vec{x_i} - t^{(i)})^2 \cdots r_N(\vec{w}^\top \vec{x_N} - t^{(1)})^2$
$= \frac{1}{2}\sum_{i=1}^N r_i(\vec{w}^\top \vec{x_i} - t^{(i)})^2$ □

Problem (b). If all the r_i's are equal to 1, we showed in class that the normal equation is

$$X^\top X w = X^\top t \tag{3}$$

and that the value of w^* that minimizes $E_D(w)$ is given by $(X^T X)^{-1} X^T t$. By finding the derivative $\nabla_w E_D(w)$ and setting that to zero, generalize the normal equation to this weighted setting, and give the new value of w^* that minimizes $E_D(w)$ in closed form as a function of X, R and t.

Proof. $\nabla_w E_D(w)$
$= \nabla_w (Xw - t)^\top R(Xw - t)$
$= w^\top(\mathbf{X}^\top \mathbf{R}\mathbf{X})w - (t^\top \mathbf{R}\mathbf{X})w - w^\top(\mathbf{X}^\top \mathbf{R}t) + t^\top \mathbf{R}t$
$= 2\mathbf{X}^\top \mathbf{R}\mathbf{X}w - 2\mathbf{X}^\top \mathbf{R}t = 0$
$\implies \mathbf{X}^\top \mathbf{R}\mathbf{X}w = \mathbf{X}^\top \mathbf{R}t$
$\implies w = (\mathbf{X}^\top \mathbf{R}\mathbf{X})^{-1}\mathbf{X}^\top \mathbf{R}t$
When all r_i equal to 1, $\mathbf{R} = \mathbf{I_N} \implies \mathbf{X}^\top \mathbf{I_N}\mathbf{X}w = \mathbf{X}^\top \mathbf{I_N}t \implies \mathbf{X}^\top \mathbf{X}w = \mathbf{X}^\top t$ □

Problem (c). Suppose we have a training set $\{(x_i, t_i); i = 1, \cdots, N\}$ of N independent examples, but in which the t_i's were observed with different variances. Specifically, suppose that

$$p(t_i|x_i; w) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\Big(- \frac{(t_i - w^T x_i)^2}{2(\sigma_i)^2}\Big). \tag{4}$$

In other words, t_i has mean $w^T x_i$ and variance $(\sigma_i)^2$ (where the σ_i's are fixed, **known** constants). Show that finding the maximum likelihood estimate of w reduces to solving a weighted linear regression problem. State clearly what the r_i's are in terms of the σ_i's.

Proof.
$ln(\prod_{i=1}^N p(t_i|x_i; w)) = ln(\prod_{i=1}^N (\frac{1}{\sqrt{2\pi}\sigma_i} \exp\Big(- \frac{(t_i - w^T x_i)^2}{2(\sigma_i)^2}\Big)))$
$= -\frac{N}{2} ln(2\pi) - \sum_{i=1}^N ln(\sigma_i) - \sum_{i=1}^N (\frac{(t_i - w^T x_i)^2}{2(\sigma_i)^2})$
$\implies \nabla_w ln(\prod_{i=1}^N p(t_i|x_i; w)) = \nabla_w(-\frac{N}{2} ln(2\pi) - \sum_{i=1}^N ln(\sigma_i) - \sum_{i=1}^N (\frac{(t_i - w^T x_i)^2}{2(\sigma_i)^2})) = \nabla_w(\sum_{i=1}^N - \frac{(t_i - w^T x_i)^2}{2(\sigma_i)^2})$
$\implies \nabla_w \sum_{i=1}^N \frac{(t_i - w^T x_i)^2}{2(\sigma_i)^2} = 0 \implies \nabla_w(\frac{1}{2}\sum_{i=1}^N r_i(w^T x_i - t_i)^2) = 0$ where $r_i = \frac{1}{\sigma_i^2}$
$\implies \nabla_w ln(\prod_{i=1}^N p(t_i|x_i; w)) = 0$
$\implies \nabla_w(\frac{1}{2}\sum_{i=1}^N r_i(w^T x_i - t_i)^2) = \nabla_w(E_D(w)) = 0$

Maximize the likelihood function above equals to minimize the weighted error function in part (a). Therefore, we can say find the MLE of w in this problem can be reduced to solving a weighted linear regression with $r_i = \frac{1}{\sigma_i^2}$ □

4) Naive Bayes Classifier

Problem (a). Download the files **spambase.train** and **spambase.test**.
The file spambase.train contains 2000 training data and spambase.test has 2601 test data. Both datasets have 58 columns: the first 57 columns are input features, corresponding to different properties of an email, and the last column is an output label indicating spam (1) or non-spam (0). Please fit the Naive Bayes model using the training data.
As a pre-processing step, quantize each variable to one of two values, say 1 and 2, so that values below the median map to 1, and others map to 2. Please aggregate the training and test test to obtain the median value of each variable, and then use the median to quantize both data sets.

(i) Look up definitions of nominal/ordinal/interval/ratio variables. Which one(s) of them are suitable for the pre-processing described above? Look up what features are used in spambase data. Are they all suitable? Report briefly.

Answer:
nominal: variables are mutually exclusive but not ordered.
ordinal: order of variables matters, but not the difference.
interval: Not only the order of the variable matters, but the difference of the values
ratio: variables have all properties of interval and also have a well defined value zero.

Since we need to calculate the median of each variable in the feature, we require each variable at least ordinal. (Interval and ratio variables would also work in the pre-process mentioned above)

(ii) Report the test error (misclassification percentage) of Naive Bayes classifier. As a sanity check, what would be the test error if you always predicted the same class, namely, the majority class from the training data?

Proof. test error=31.6801% and the test error from the sanity check(predict all test y are spam) is 38.5621% □

Problem (b). Open Kaggle challenge: you can design any features to perform Naive Bayes classification on the Naive Bayes Spam Filter Dataset. All details for this project are included in the Kaggle webpage **https://inclass.kaggle.com/c/naive-bayes-spam-filter**.
A utility script `extract_feature.py` that performs lemmatization and stopwords removal of email texts is provided. You may adapt it for your feature extraction.
This problem will be graded separately based on your performance on the private leaderboard. Besides submitting your solutions to Kaggle, briefly describe how you extract features from email texts.

Name Tsung-HungYao with score 95.54189

5) Softmax Regression

In this problem, you will generalize logistic regression (for binary/2-class classification) to allow more classes (> 2) of labels (for multi-class classification).

In logistic regression, we had a training set $\{(\phi(\mathbf{x}_n), t_n)\}$ of N examples, where $t_n \in \{0, 1\}$, and the likelihood function is

$$p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^{N} y_n^{t_n}(1 - y_n)^{1-t_n} \tag{5}$$

where $y_n = p(C_1|\phi(\mathbf{x}_n)) = \sigma(\mathbf{w}^T\phi(\mathbf{x}_n))$ and $1 - y_n = p(C_0|\phi(\mathbf{x}_n)) = 1 - \sigma(\mathbf{w}^T\phi(\mathbf{x}_n))$.

Now we generalize the logistic regression to multi-class classification, where the label t can take on K different values, rather than only two. We now have a training set $\{(\phi(\mathbf{x}_n), t_n)\}$ of N examples, where $t_n \in \{0,1, ..., K-1\}$. We apply a softmax transformation of linear functions of the feature variables ϕ for the posterior probabilities $p(C_k|\phi)$, so that

$$p(C_k|\phi) = \frac{\exp(\mathbf{w}_k^T\phi)}{\sum_{k=0}^{K-1}\exp(\mathbf{w}_k^T\phi)}$$

where $\mathbf{w}_k$ are the parameters of the linear function for class k, and $\mathbf{w} = \{\mathbf{w}_k\}_{k=0}^{K-1}$ are the parameters for softmax regression. The likelihood function is then given by

$$p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^{N}\prod_{k=0}^{K-1} p(C_k|\phi(\mathbf{x}_n))^{\mathbf{1}(t_n=k)} \tag{6}$$

where $\mathbf{1}(.)$ is an indicator function so that $\mathbf{1}$(a true statement)=1 and $\mathbf{1}$(a false statement)=0. In the likelihood function, $\mathbf{1}(t_n = k)$ returns 1 if t_n equals k and 0 otherwise.

Problem (a). Gradient descent for softmax regression. We can optimize the softmax regression model in the same way as logistic regression using gradient descent. Please write down the error function for softmax regression $E(\mathbf{w})$ and derive the gradient of the error function with respect to one of the parameter vectors $\mathbf{w}_j$, i.e. $\nabla_{\mathbf{w}_j}E(\mathbf{w})$. (Show your derivation and final result using the given notations. The final result should not contain any partial derivative or gradient operator.)

Proof. $E(\mathbf{w}) = -\ln p(\mathbf{t}|\mathbf{w}) = -\ln(\prod_{n=1}^{N}\prod_{k=0}^{K-1} p(C_k|\phi(\mathbf{x}_n))^{\mathbf{1}(t_n=k)})$
$= -\sum_{n=1}^{N}(\sum_{k=0}^{K-1}\mathbf{1}(t_n = k)\ln p(C_k|\phi(\mathbf{x}_n)))$

$\nabla_{\mathbf{w}_j}\sum_{k=0}^{K-1}\mathbf{1}(t_n = k)\ln p(C_k|\phi)$
$= \nabla_{\mathbf{w}_j}\sum_{k=0}^{K-1}\mathbf{1}(t_n = k)[(\mathbf{w}_k^T\phi) - \ln(\sum_{k=0}^{K-1}\exp(\mathbf{w}_k^T\phi))]$
$= \nabla_{\mathbf{w}_j}\sum_{k=0}^{K-1}\mathbf{1}(t_n = k)(\mathbf{w}_k^T\phi) - \nabla_{\mathbf{w}_j}\sum_{k=0}^{K-1}\mathbf{1}(t_n = k)\ln(\sum_{k=0}^{K-1}\exp(\mathbf{w}_k^T\phi))$
$= \mathbf{1}(t_n = j)\nabla_{\mathbf{w}_j}\sum_{k=0}^{K-1}(\mathbf{w}_k^T\phi) - \nabla_{\mathbf{w}_j}\ln(\sum_{k=0}^{K-1}\exp(\mathbf{w}_k^T\phi))$
$= \mathbf{1}(t_n = j)\phi(\mathbf{x_n}) - \frac{\nabla_{\mathbf{w}_j}\sum_{k=0}^{K-1}\exp(\mathbf{w}_k^T\phi)}{\sum_{k=0}^{K-1}\exp(\mathbf{w}_k^T\phi)}$
$= \mathbf{1}(t_n = j)\phi(\mathbf{x_n}) - \frac{\exp(\mathbf{w}_j^T\phi)\phi}{\sum_{k=0}^{K-1}\exp(\mathbf{w}_k^T\phi)}$

$= \mathbf{1}(t_n = j)\phi(\mathbf{x_n}) - p(C_j|\phi(\mathbf{x}_n))\phi(\mathbf{x}_n)$
$= (\mathbf{1}(t_n = j) - p(C_j|\phi(\mathbf{x}_n))\phi(\mathbf{x}_n)$
$\implies \nabla_{\mathbf{w}_j} E(\mathbf{w})$
$= -\sum_{n=1}^{N} \nabla_{\mathbf{w}_j} \sum_{k=0}^{K-1} \mathbf{1}(t_n = k) \ln p(C_k|\phi)$
$= -\sum_{n=1}^{N} (\mathbf{1}(t_n = j) - p(C_j|\phi(\mathbf{x}_n))\phi(\mathbf{x}_n)$

□

Problem (b). Overparameterized property of softmax regression parameterization and weight decay.

Softmax regression's parameters are overparameterized, which means that for any hypothesis we might fit to the data, there are multiple parameter settings that result in the same mapping from $\phi(\mathbf{x})$ to predictions. For example, if you add a constant to every w, the softmax regression's predictions does not change. There are multiple solutions for this issue. Here we consider one simple solution: modifying the error function by adding a weight decay term, $\sum_{k=0}^{K-1} \mathbf{w}_k^T \mathbf{w}_k$:

$$E^{\lambda}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2} \sum_{k=0}^{K-1} \mathbf{w}_k^T \mathbf{w}_k \tag{7}$$

For any parameter $\lambda > 0$, the error function $E^{\lambda}(\mathbf{w})$ is strickly convex, and is guaranteed to have a unique solution. Similarly, please derive the gradient of the modified error function $E^{\lambda}(\mathbf{w})$ with respect to one of the parameter vectors $\mathbf{w}_j$, i.e. $\nabla_{\mathbf{w}_j} E^{\lambda}(\mathbf{w})$. (Show your derivation and final result using the given notations. The final results should not contain any partial derivative or gradient operator.)

Proof. $\nabla_{\mathbf{w}_j} E^{\lambda}(\mathbf{w})$
$= \nabla_{\mathbf{w}_j} (E(\mathbf{w}) + \frac{\lambda}{2} \sum_{k=0}^{K-1} \mathbf{w}_k^T \mathbf{w}_k$
$= -\sum_{n=1}^{N} (\mathbf{1}(t_n = j) - p(C_j|\phi(\mathbf{x}_n))\phi(\mathbf{x}_n) + \frac{\lambda}{2} \nabla_{\mathbf{w}_j} (\mathbf{w}_j^T \mathbf{w}_j)$
$= -\sum_{n=1}^{N} (\mathbf{1}(t_n = j) - p(C_j|\phi(\mathbf{x}_n))\phi(\mathbf{x}_n) + \lambda \mathbf{w}_j$

□

6) Code Appendix

Problem 1) Linear Regression (a) (i) and (ii). Code

```
# -*- coding: utf-8 -*-
"""
Created on Fri Feb 05 16:34:57 2016

@author: Shaking
"""
from matplotlib import pyplot as plt
import numpy as np
train=np.genfromtxt("train_graphs_f16_autopilot_cruise.csv",delimiter=",")
train=np.delete(train,0,1)
train=np.delete(train,0,0)
test_csv=np.genfromtxt("test_graphs_f16_autopilot_cruise.csv",delimiter=",")
```

```
test_csv=np.delete(test_csv,0,0)
test_feature_noOne=test_csv[:,1:7]
test_y=test_csv[:,7]
#feature and y extracted
feature_noOne=train[:,0:6]
train_y=train[:,6]

def AddColumn(Matrix1,Matrix2):
    dimM1=np.shape(Matrix1)
    dimM2=np.shape(Matrix2)
    if len(dimM1)==1:
        dimM1+=(1,)
    if len(dimM2)==1:
        dimM2+=(1,)
    BigMatrix_flat=np.append(np.transpose(Matrix1),np.transpose(Matrix2))
    BigMatrix=BigMatrix_flat.reshape(dimM2[1]+dimM1[1],dimM1[0])
    BigMatrix=np.transpose(BigMatrix)
    return BigMatrix

RMSE_train_1a=[]
RMSE_test_1a=[]
M_lst=range(1,7)
for i in M_lst:
    one=np.ones(np.shape(train[:,0]))
    test_one=np.ones(np.shape(test_csv[:,0]))
    if i ==1:
        Phi_x=AddColumn(one,feature_noOne)
        test_Phi=AddColumn(test_one,test_feature_noOne)
    else:
        Phi_x=AddColumn(Phi_x, np.power(feature_noOne,i))
        test_Phi=AddColumn(test_Phi, np.power(test_feature_noOne,i))
    Phi_dagger=np.dot(np.linalg.inv(np.dot(np.transpose(Phi_x),Phi_x)),np.trans
    print np.shape(Phi_dagger)
    w_1=np.dot(Phi_dagger,train_y)
    print np.shape(w_1), w_1
    Error_train=0.5*np.dot(np.transpose(np.dot(Phi_x,w_1)),np.dot(Phi_x,w_1))-r
    RMSE_train=(2*Error_train/3426)**(0.5)
    print np.shape(np.transpose(np.dot(Phi_x,w_1))),np.shape(test_y)
    Error_test=0.5*np.dot(np.transpose(np.dot(test_Phi,w_1)),np.dot(test_Phi,w_
    RMSE_test=(2*Error_test/np.shape(test_y)[0])**(0.5)
    RMSE_train_1a.append(RMSE_train)
    RMSE_test_1a.append(RMSE_test)
print RMSE_train_1a
print RMSE_test_1a

plt.plot(M_lst,RMSE_train_1a,'-r',label='train RMSE')
plt.plot(M_lst,RMSE_test_1a,'-b', label='test RMSE')
```

```
plt.xlabel('M')
plt.ylabel('RMSE')
plt.legend()
plt.show()

RMSE_train_1b=[]
RMSE_test_1b=[]
for j in range(-40,21):
    Lambda=np.exp(j)
    Phi_reg_dagger=np.dot(np.linalg.inv(Lambda*np.eye(np.shape(Phi_x)[1])+np.do
    w_reg=np.dot(Phi_reg_dagger,train_y)
    Error_train=0.5*np.dot(np.transpose(np.dot(Phi_x,w_reg)),np.dot(Phi_x,w_reg
    RMSE_train=(2*Error_train/3426)**(0.5)
    Error_test=0.5*np.dot(np.transpose(np.dot(test_Phi,w_reg)),np.dot(test_Phi,
    RMSE_test=(2*Error_test/np.shape(test_y)[0])**(0.5)
    RMSE_train_1b.append(RMSE_train)
    RMSE_test_1b.append(RMSE_test)
print RMSE_train_1b
print RMSE_test_1b

plt.plot(range(-40,21),RMSE_train_1b, '-r',label= 'train RMSE')
plt.plot(range(-40,21),RMSE_test_1b,'-b', label="tets RMSE")
plt.xlabel(r'ln($\lambda$)')
plt.ylabel('RMSE')
plt.legend()
plt.show()
```

Problem 1) Linear Regression (b). Code

```
 # -*- coding: utf-8 -*-
"""
Created on Sat Feb 06 23:22:52 2016

@author: Shaking
"""
from matplotlib import pyplot as plt
import numpy as np
def gaussian_kernel(x_i, query_i, decaySpeed):
    diff = x_i - query_i
    distance=np.dot(np.transpose(diff),diff)
    return np.exp(-distance/(2*np.square(decaySpeed)))

def weight_matrix(xMatrix, query_i, decaySpeed):
    weight=np.eye(np.shape(xMatrix)[0])
    for i in range(np.shape(xMatrix)[0]):
        weight[i,i]=gaussian_kernel(xMatrix[i,:],query_i,decaySpeed)
    return weight
train=np.genfromtxt('train_graphs_f16_autopilot_cruise.csv',delimiter=',')
```

```
test=np.genfromtxt('test_locreg_f16_autopilot_cruise.csv',delimiter=',')
train=np.delete(train,0,0)
test=np.delete(test,0,0)
train_y=train[:,7]
train_x=train[:,1:7]
one_train=np.ones(np.shape(train_x[:,0]))
#train_x=np.hstack((one_train.reshape(np.shape(train_x)[0],1),train_x))
train_x_mean=np.mean(train_x,axis=0)
train_x_variance=np.var(train_x,axis=0)
print train_x_mean, train_x_variance
train_x_normed=(train_x-train_x_mean)/np.sqrt(train_x_variance)
test_y=test[:,7]
test_x=test[:,1:7]
one_test=np.ones(np.shape(test_x[:,0]))
#test_x=np.hstack((one_test.reshape(np.shape(test_x)[0],1),test_x))
test_x_mean=np.mean(test_x,axis=0)
test_x_variance=np.var(test_x,axis=0)
print test_x_mean, test_x_variance
test_x_normed=(test_x-test_x_mean)/np.sqrt(test_x_variance)
tau=np.logspace(-2, 1, num=10, base=2)
print tau
RMSE_lst=[]

for i in range(len(tau)):
    RMSE=0
    for j in range(len(test_x_normed)):
        R_w_j=weight_matrix(train_x_normed,test_x_normed[j],tau[i])
        weighted_x=np.dot(np.dot(np.transpose(train_x_normed),R_w_j),train_x_no
        coeff_j=np.dot(np.dot(np.dot(np.linalg.inv(weighted_x),np.transpose(tra
        predict_y_j=np.dot(np.transpose(coeff_j),test_x_normed[j])
        RMSE+=np.sqrt((2*((predict_y_j-test_y[j])**2)/100))
    RMSE_lst.append(RMSE)

print RMSE_lst
plt.plot(tau,RMSE_lst,'-r',label="RMSE")
plt.xlabel(r'$\tau$')
plt.ylabel("RMSE")
plt.legend()
plt.show()
```

Problem 2) Open Kaggle Challenge. Code

```
# -*- coding: utf-8 -*-
"""
Created on Fri Feb 05 16:34:57 2016

@author: Shaking
"""
```

```
import csv
import numpy as np
train=np.genfromtxt("steel_composition_train.csv",delimiter=",")
train=np.delete(train,0,1)
train=np.delete(train,0,0)
test_csv=np.genfromtxt("steel_composition_test.csv",delimiter=",")
test_csv=np.delete(test_csv,0,0)
test_feature_noOne=test_csv[:,1:9]
#feature and y extracted
feature_noOne=train[:,0:8]
train_y=train[:,8]

def AddColumn(Matrix1,Matrix2):
    dimM1=np.shape(Matrix1)
    dimM2=np.shape(Matrix2)
    if len(dimM1)==1:
        dimM1+=(1,)
    if len(dimM2)==1:
        dimM2+=(1,)
    BigMatrix_flat=np.append(np.transpose(Matrix1),np.transpose(Matrix2))
    BigMatrix=BigMatrix_flat.reshape(dimM2[1]+dimM1[1],dimM1[0])
    BigMatrix=np.transpose(BigMatrix)
    return BigMatrix

RMSE_train_1a=[]
RMSE_test_1a=[]
M_lst=range(1,5)
for i in M_lst:
    one=np.ones(np.shape(train[:,0]))
    test_one=np.ones(np.shape(test_csv[:,0]))
    if i ==1:
        Phi_x=AddColumn(one,feature_noOne)
        test_Phi=AddColumn(test_one,test_feature_noOne)
    else:
        Phi_x=AddColumn(Phi_x, np.power(feature_noOne,i))
        test_Phi=AddColumn(test_Phi, np.power(test_feature_noOne,i))
    Phi_dagger=np.dot(np.linalg.inv(np.dot(np.transpose(Phi_x),Phi_x)),np.trans
    print np.shape(Phi_dagger)
    w_1=np.dot(Phi_dagger,train_y)
    print np.shape(w_1), w_1

predict_y=np.dot(test_Phi,w_1)
print predict_y
f=open('HW2_Q2.csv','w')
csv_w=csv.writer(f)
csv_w.writerow(predict_y)
f.close()
```

Problem 4) Naive Bayes Classifier (a)(ii). Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat Feb 06 14:09:02 2016

@author: Shaking
"""

import numpy as np
train=np.genfromtxt("spambase.train",delimiter=",")
test=np.genfromtxt("spambase.test",delimiter=",")
print np.shape(train),np.shape(test)
train_x=train[:,0:57]
train_y=train[:,57]
test_x=test[:,0:57]
test_y=test[:,57]
whole_data=np.vstack((train_x,test_x))
print np.shape(whole_data)
median=np.median(whole_data,axis=0)
print median
trainX_proed=(train_x>=median)
testX_proed=(test_x>=median)
print np.shape(trainX_proed)
print np.shape(testX_proed)
print np.shape(train_y)
print np.shape(test_y)
train_proed=np.hstack((trainX_proed,train_y.reshape(np.shape(train_y)[0],1)))
test_proed=np.hstack((testX_proed,test_y.reshape(np.shape(test_y)[0],1)))
print np.shape(train_proed)
print np.shape(test_proed)
train_spam=train_proed[train_proed[:,57]==1]
train_ham=train_proed[train_proed[:,57]==0]
test_spam=test_proed[test_proed[:,57]==1]
test_ham=test_proed[test_proed[:,57]==0]
print np.shape(train_spam), np.shape(train_ham), np.shape(test_spam), np.shape(
pi_spam=(np.shape(train_spam)[0]+1.0)/(np.shape(train_proed)[0]+2.0)
print pi_spam

train_spam_nonzero=[]
for i in range(len(train_spam[0,:])-1):
    nonzero=np.count_nonzero(train_spam[:,i])
    train_spam_nonzero.append(nonzero)
print train_spam_nonzero
theta_spam=(np.array(train_spam_nonzero)+1.0)/(len(train_spam)+2.0)
print theta_spam

train_ham_nonzero=[]
```

```
for i in range(len(train_ham[0,:])-1):
    nonzero=np.count_nonzero(train_ham[:,i])
    train_ham_nonzero.append(nonzero)
print train_ham_nonzero
theta_ham=(np.array(train_ham_nonzero)+1.0)/(len(train_ham)+2.0)
print theta_ham

predict_y=[]
for i in range(len(testX_proed)):
    datapoint=testX_proed[i,:]
    prob_spam=1.0
    prob_ham=1.0
    for j in range(len(datapoint)):
        if datapoint[j]==1:
            prob_spam=prob_spam*theta_spam[j]
            prob_ham=prob_spam*theta_ham[j]
        else:
            prob_spam=prob_spam*(1-theta_spam[j])
            prob_ham=prob_ham*(1-theta_ham[j])
    prob_spam=prob_spam*pi_spam
    prob_ham=prob_ham*(1-pi_spam)
    if prob_spam>prob_ham:
        predict_y.append(1)
    else:
        predict_y.append(0)
print np.shape(np.array(predict_y))

error_count=0
error_check_count=0
for i in range(len(predict_y)):
    if predict_y[i] != test_y[i]:
        error_count+=1
    if test_y[i]==1:
        error_check_count+=1

print error_count
print float(error_count)/2601

print error_check_count
print float(error_check_count)/2601
```

Problem 4) Naive Bayes Classifier (b). Code

```
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 08 00:47:02 2016

@author: Shaking
```

```
"""
import numpy as np
import pickle
import scipy
import csv

with open('trainFeatures.pkl', 'rb') as f:
    trainFeature = pickle.load(f)
with open('testFeatures.pkl', 'rb') as g:
    testFeature=pickle.load(g)

train_feature=scipy.sparse.csr_matrix(trainFeature).toarray()
test_feature=scipy.sparse.csr_matrix(testFeature).toarray()
print np.shape(train_feature)
print np.shape(test_feature)

f=open("spam_filter_train.txt",'r')
train_y=[]
for line in f:
    if line.split()[0]=="spam":
        train_y.append(1)
    else:
        train_y.append(0)
f.close()
print len(train_y)
train_y=np.array(train_y).reshape((len(train_y),1))
print np.shape(train_y)
print np.shape(train_feature)

train_data=np.hstack((train_feature,train_y))
print np.shape(train_data)

from sklearn.naive_bayes import MultinomialNB
mnb = MultinomialNB()
y_pred = mnb.fit(train_feature, train_y).predict(test_feature)
print y_pred, len(y_pred)

g=open('HW2_Q4b.csv','w')
csv_w=csv.writer(g)
csv_w.writerow(y_pred)
g.close()
```