Spam Mail Classification

R05942078，陳偉寧

1. Logistic Regression

A. Probability Model

Logistic regression 的機率模型課堂上已經講解得非常清楚，故在這邊僅簡略提一下我們的ERM rule (empirical risk minimization)、loss function、以及gradient descent 的實作。

首先，logistic regression的probability model 如下：

$P(X \mid y=1)=\frac{1}{1+e^{-(w\cdot X+\beta)}}$, 其中我們定 $f(x)=\frac{1}{1+e^{-x}}$ 為sigmoid function

在此機率模型下，其 negative log likelihood 則是cross entropy:

$H((X,y))=-\sum(y_i \log(f(w\cdot x_i+\beta))+(1-y_i)\log(f(w\cdot x_i+\beta))$, 故minimize cross entropy即為ML estimator。值得注意的是，logistic loss (i.e. cross entropy)是convex function（詳細可見Appendix A），因此gradient descent是個求出最佳解的合適方式。

B. Implementation and Optimization

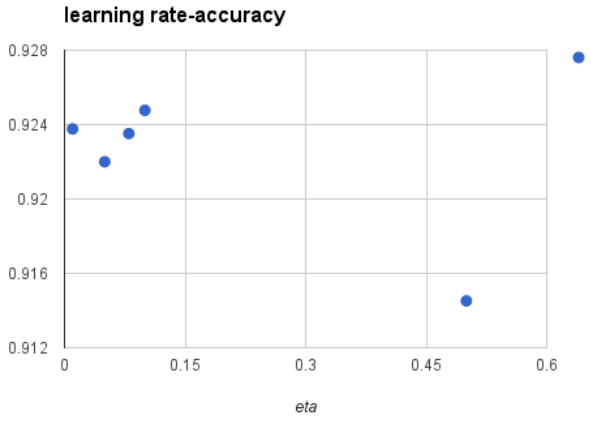
要minimize cross entropy，我們用採用gradient descent。為了避免每個iteration都要更新bias，我們先將raw data增加一個dimension並令其為1，如此可假設bias term為零。而其update的方式如下：

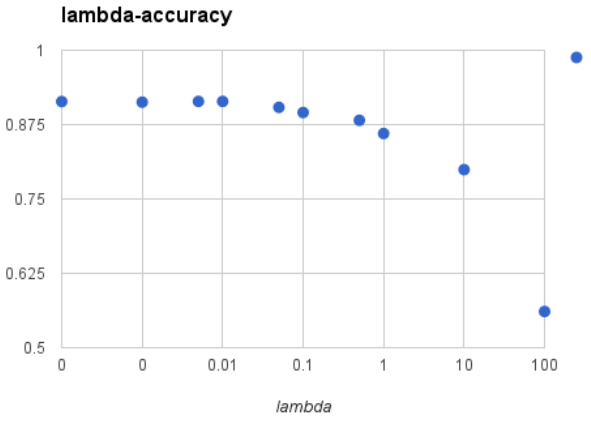
$$w_{i+1}=w_i-\eta\cdot\frac{1}{m}\sum(h_w(x_i)-y_i)\cdot x_i$$

，其中 $h_w(x_i)=\frac{1}{1+e^{-w\cdot x_i}}$ 。 實作上為了更有效率，我使用了adagrad，詳細的code在Appendix B。

C. Model Selection: Cross Validation

如先前在linear regression一般，為了避免overfitting，我們會加上一個regularizer防止得到過於貼合training data的解，而經過作業一到慘痛教訓，在這次作業中我執行了4-folded cross validation來決定出最合適的lambda。另外，我也測試了一下learning rate對於logistic regression的影響如下（詳細的code及數據別附於AppendixB、Appendix C）：



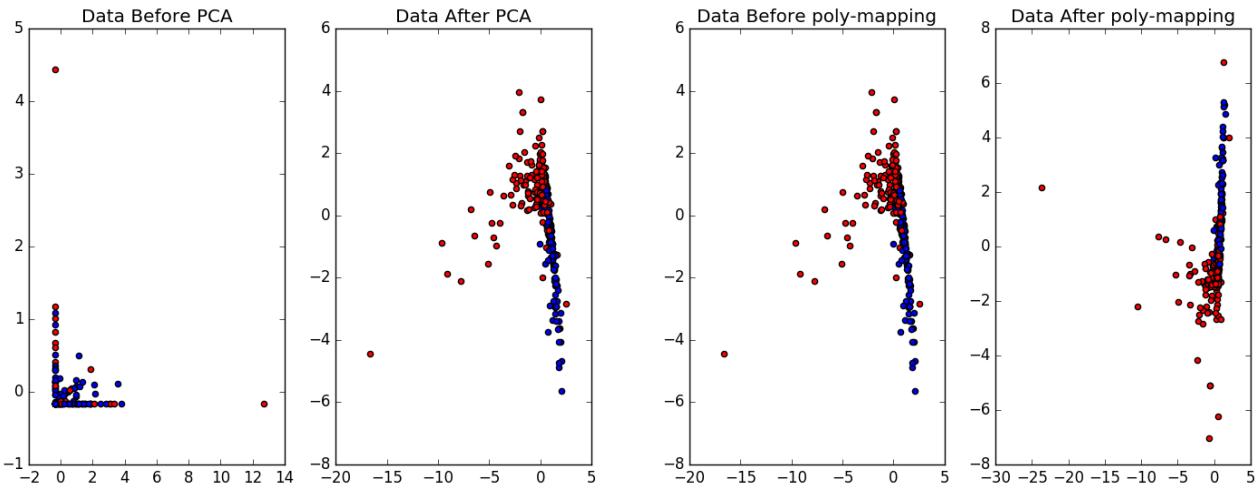


2. Method 2: Feature Extraction

首先，我從data的feature extraction下手。
對於linear non-separable的data，傳統classification通常會將data透過一個feature map將其mapping到一個high dimensional space (e.g. kernel method in SVM)。在NLP (natural language processing)中常用到的kernel 則是polynomial kernel。因spam detection本質上與NLP十分接近，因此我選擇使用**polynomial feature mapping**。然而為了避免過多的參數造成logistic regression的運算量過大、以及過多的variable所造成的overfitting，我選擇使用**PCA (principle component analysis)**進行dimension reduction，最後得到我們想要的feature。以下將詳述我的實作方法。

A. Polynomial Feature Mapping
在polynomial mapping中，為了避免過多的變數造成運算量過大的問題，我並沒有考慮cross symbol term。另外，polynomial的dimension是一個可調的參數。
以二次的polynomial mapping 為例，詳細定義如下：$[x_1,x_2,...,x_n]^T \rightarrow [x_1,x_1^2,x_2,x_2^2,...,x_n,x_n^2]^T$ 。
在我們這次的問題中，raw data的dimension是57維，經過了k-dimensional polynomial mapping後則是變成 $57 \times k$ dimensions data。

B. Principle Component Analysis
如同前所述，當經過了polynomial mapping後data的dimension可能會太高，因此這邊我使用PCA降維。概念上來說，PCA主要是將原先的data做一個basis transformation，使其在每個dimension的variance最大，也因此使data更容易被separate。而經過一些計算後（詳細內容可參閱維基百科），我們發現這些basis恰好是data covariance matrix的eigenvector，並且其重要性恰好對應於eigenvalues的大小。為了應證我們的想法，我將經過PCA轉換前後、經過polynomial mapping前後的data用二維散布圖表現出來如下：



我們可以發現，經過PCA/polynomial mapping後，data的分離程度有明顯差異。經過多次試驗後，我發現以10次polynomial mapping後，再以PCA reduce to 100 dimension效果最好。詳細code附於Appendix B。

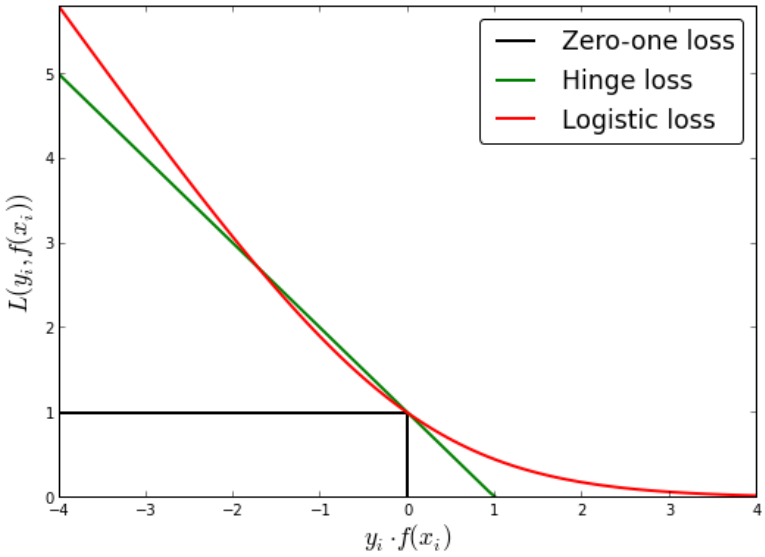
C. Discussion and Result
經過如前述的feature mapping後，我最後是以logistic regression作為classifier，並得到我的kaggle_best，但我同時也比較了幾種不同方式（有使用sklearn，但沒有送到kaggle上）

Dimension vs Accuracy (PCA dim = 100)								
Poly. dim	1	2	5	8	10	15	20	25
Accuracy	93.33%	94.67%	94.67%	94.33%	94.67%	93%	91%	91.33%

Algorithm vs Accuracy (PCA dim = 100, poly = 10)						
alg.	NN(2,5,2)	NN(2,2,2,2)	SVM(C=1)	SVM(C=10)	SVM(C=0.1)	k-Nearest-Neighbor
Accuracy	93.33%	91.67%	94.00%	94.67%	93.67%	81%

Appendix

A. Comparison of Logistic Loss and other loss



B. Code Section

(1) gradient of cross entropy (with regularizer)

```
def grad_cross_entropy(dataset, w, lambda):
  [data_X, data_y] = dataset
  g = 0
  for idx in range(len(data_y)):
    x = data_X[idx]
    y = data_y[idx]
    g += (sigmoid(w.T.dot(x))-y)*x
  return g / len(data_y) + 0.5*lmbda*w
```

(2) Adagrad (Minimize cross entropy)

```
def Adagrad(dataset, loss, grad_loss, model_init = 0,  eta = 0.1, it =
60000, lmbda = 0.1):
  [data_X, data_y]=dataset
  w = np.zeros(len(data_X[0]))
  gd_sum = np.zeros(len(w))+1e-8
  print "> Initialize Model"
  for i in range(it):
    gd = grad_loss(dataset, w, lmbda)
    for j in range(len(gd_sum)):
      gd_sum[j] = gd_sum[j]+gd[j]*gd[j]
      w[j] = w[j] - eta/np.sqrt(gd_sum[j])*gd[j]
  return np.array([w, gd_sum])
```

(3) Cross Validation

```
# Cross validation
  model_scores = []
  for i in [0,0.0001, 0.001, 0.005, 0.01, 0.05]:
    mod_init = np.load('model/models_12.npy')
    model = reg_logistic_regression.lr(it = 20000, eta =0.1, model_init =
0, lmbda = i)
    scores = cross_val_score(model, train_X, train_y, scoring =
'accuracy', cv = 4)
    print 'Scores:'
    print scores
    print 'Average :'+str(np.mean(scores))
    model_scores.append(np.mean(scores))
  print model_scores
```

(4) Polynomial Mapping

```
def poly_mapping(data, k = 10):
  '''
    Mapping the original data to high-dimensional space
  '''
  [num, dim] = data.shape
  feature = np.zeros([num, k*dim], dtype = float)
  for idx in range(num):
    for i in range(dim):
      for pw in range(k):
        feature[idx, k*i+pw]  = np.power(data[idx, i],pw+1)
  return feature[:,:-k]
```

(5) PCA

```
def pca(data, pc_count = None):
    """
     Principal component analysis using eigenvalues
    """
    d_mean = mean(data, 0)
    d_std = std(data, 0)
    data -= d_mean
    data /= d_std
    C = cov(data)
    E, V = eigh(C)
    key = argsort(E)[::-1][:pc_count]
    E, V = E[key], V[:, key]
    U = dot(V.T, data.T).T
    return [U, [d_mean, d_std, V]]
```

(6) PCA reconstruction

```
def pca_reconstruct(data, recst):
    data -= recst[0] # mean(data, 0)
    data /= recst[1] # std(data, 0)
    U = dot(recst[2].T, data.T).T
    return U
```

C. Experimental and Stats

Regularizer-Accuracy	
lambda	acc
0	0.914277
0.0001	0.913778
0.001	0.912778
0.005	0.914028
0.01	0.914028
0.05	0.904029
0.1	0.895331
0.5	0.882279
1	0.860032
10	0.799296
100	0.560864

Learning rate-Accuracy	
eta	acc
0.1	0.92477
0.5	0.91452
0.01	0.92377
0.05	0.922
0.08	0.92352