

CS5785

Applied Machine Learning

HW1

Due Date: 09/14/2016

Siyadong (Sid) Xiong

sx225@cornell.edu

Teammates: N/A

1 Programming Exercises

1.1 Digit Recognizer

1.1.1 Imported Libraries & Helper Functions

We firstly imported common ml libraries and wrote a helper logging function.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import matplotlib.cm as cm
4 from sklearn import cross_validation
5
6 def mlog(s, level="INFO"):
7     print "[%s]: %s" % (level, s)
8
9 data = np.loadtxt('train.csv', delimiter=',', skiprows=1,
10                  dtype=np.int32)
11 hei = wid = int((data.shape[1] - 1) ** 0.5)
```

1.1.2 Display Digits

We used the following code to demonstrate each digit in one figure as demonstrated in Fig.1

```
1 def show(data):
2     plt.figure(figsize=(10, 5))
3     for y in range(10):
4         x = data[data[:,0] == y,
5                  :][np.random.randint(50),1:].reshape((hei, wid))
6         ax = plt.subplot(2, 5, y+1)
7         ax.set_title("number: %d" % y)
8         ax.imshow(x, cmap=cm.binary)
9     plt.tight_layout()
```

```

9
10 show(data)

```

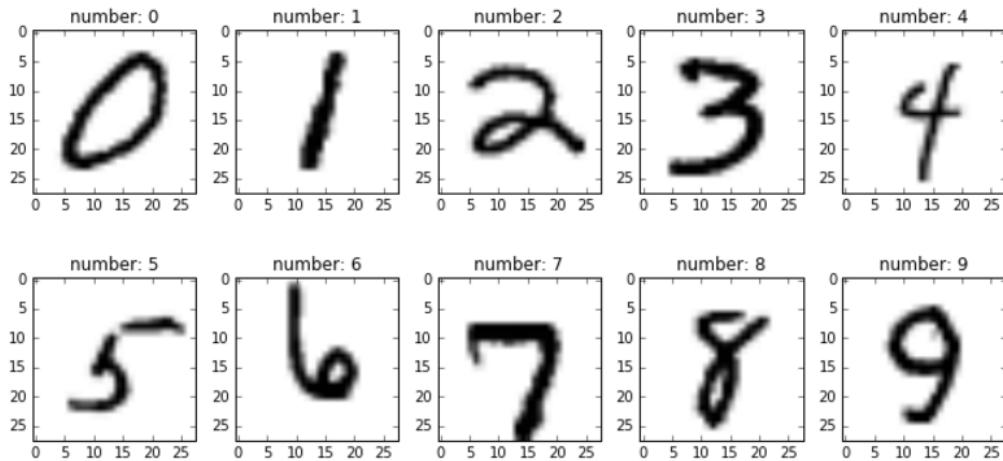


Figure 1: Plot of each digit

To avoid duplicates, code for plotting is omitted in the remaining report. Details could be found in `.ipynb` files.

1.1.3 Prior Probabilities of Labels

By the following code,

```

1 plt.figure(figsize=(10, 3))
2 _a, _b = np.histogram(data[:, 0], 10, normed=True)

```

we get the prior probabilities of all digit labels, which can be demonstrated via a histogram as Fig.2 demonstrated. The histogram indicated that the distribution of digits is rather uniform.

1.1.4 Nearest Neighbor under L2 Distance

Nearest neighbor is a particular case of kNN when k equals 1. The implementation of kNN model is described in section.1.1.7. After indexing a digit and getting its complementary indice in the dataset via this code,

```

1 for num in xrange(10):
2     current_number_idx = (data[:, 0] == num).nonzero()[0]
3     sample_idx =
        current_number_idx[np.random.randint(len(current_number_idx))]

```

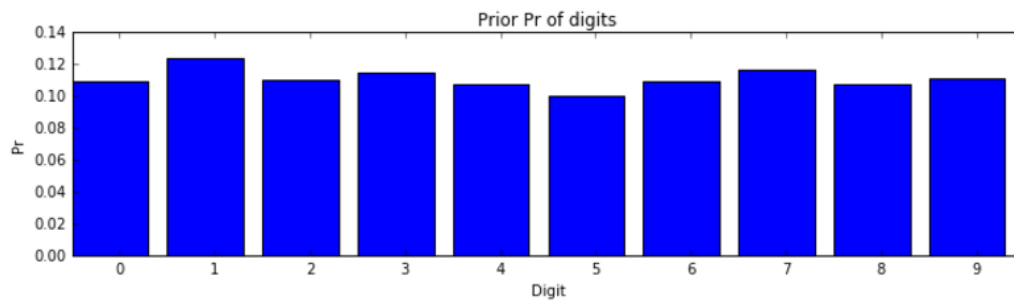


Figure 2: Normalized histogram of prior probabilities of labels

```

4 other_idx = np.ones(data.shape[0], dtype=bool)
5 other_idx[sample_idx] = False
6
7 mdl = kNN()
8 mdl.fit(data[other_idx, :])
9 sample_x = data[sample_idx, 1:]
10 nn, l2_dist = mdl._get_nn(sample_x)

```

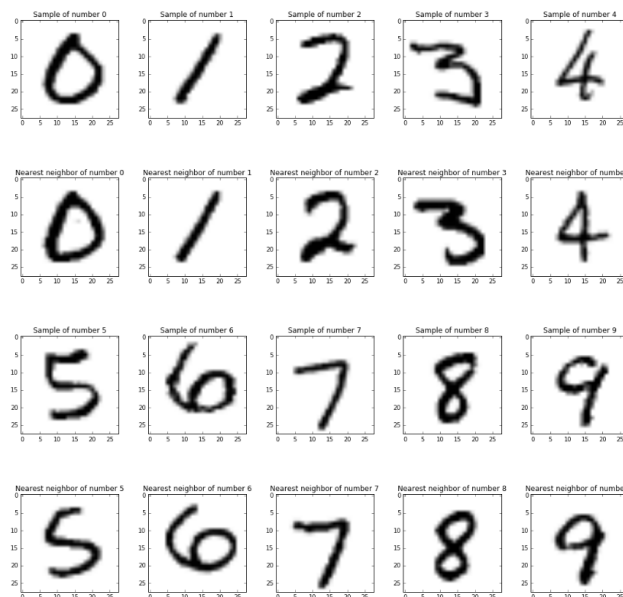


Figure 3: Comparison of 1-NN

we get the digit's nearest neighbor. Fig.3 shows the comparison of each sample digit and its nearest neighbor. In my opinion, as *mnist* is a hand writing image dataset, the probability density functions among digits have

little overlaps, hence performances pretty well the 1-NN model.

1.1.5 Genuine and Impostor Distances

In this section, we compared all the pairwise L_2 distance over images of digits 0 and 1. Following the definition in the problem statement, via this code,

```
1 _n, _bins, _ = plt.hist(genuine_dist, label='genuine',
    alpha=0.3, range=(0, 4500), bins=9)
2 _n2, _bins2, _ = plt.hist(impostor_dist, label='impostor',
    alpha=0.3, range=(0, 4500), bins=9)
```

we plot the histogram of *genuine match distance* and *impostor match distance* in one figure, as shown in Fig.4. As we expected, the distributions of genuine

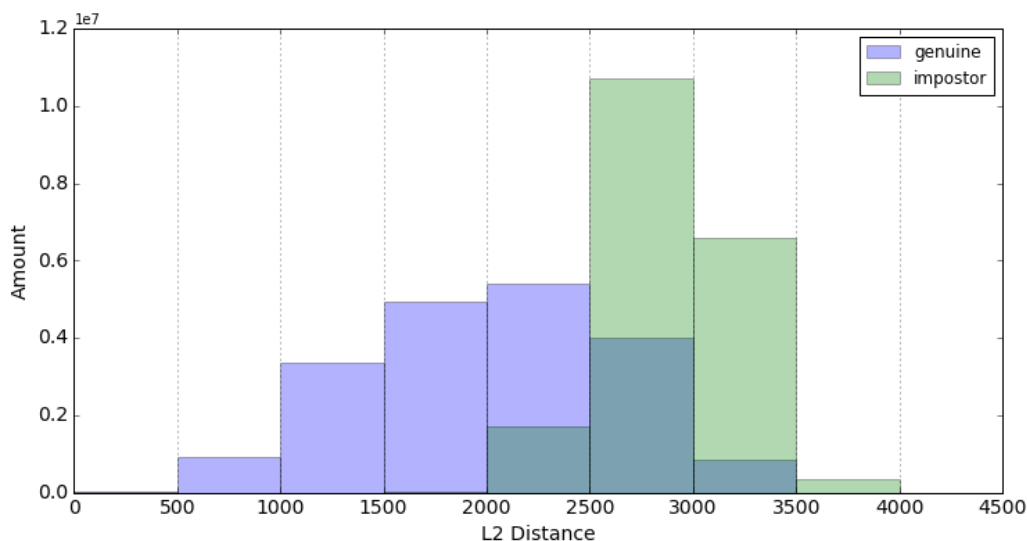


Figure 4: Distribution of *genuine distances* and *impostor distances*

matches and impostor matches overlapped little, which make it feasible for us to use kNN model to classify digits.

1.1.6 ROC Curve

By setting classification thresholds by *step* = 5 on above sets of distances, ranging from [250, 4500), we got the TPRs and FPRs.

```
1 len_gen = float(len(genuine_dist))
2 len_imp = float(len(impostor_dist))
```

```

3
4 tpr = []
5 fpr = []
6 for thr in xrange(250, 4500, 5):
7     tpr.append(np.count_nonzero(genuine_dist < thr) / len_gen)
8     fpr.append(np.count_nonzero(impostor_dist < thr) / len_imp)

```

Equal Error Rate (EER) is the point where FPR equals FNR. It is the intersecting point of ROC curve and anti-diagonal of unit square, as indicated in Fig.5. So the EER was approximately 0.185. Guessing randomly, the error rate would be 0.5.

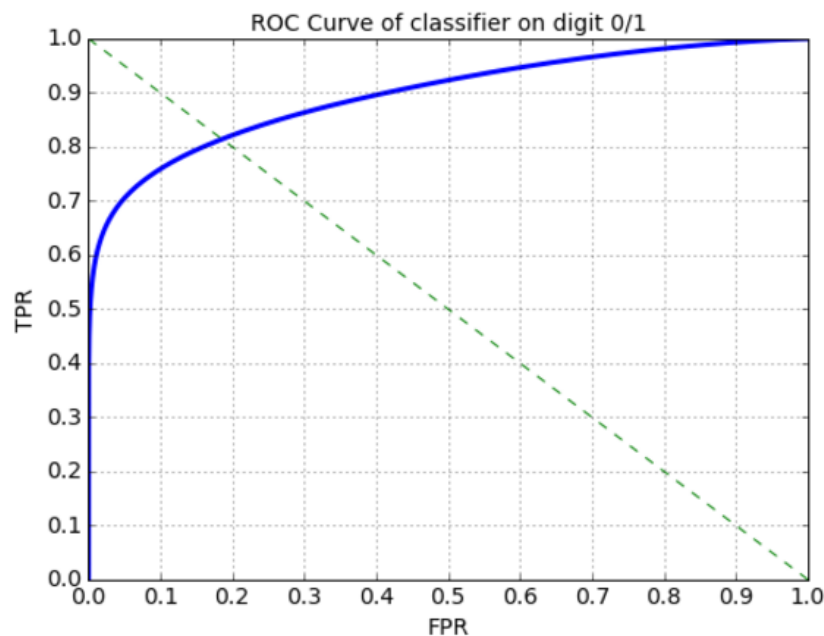


Figure 5: ROC curve.

1.1.7 kNN Implementation and Performance

Here we implemented a kNN class that had similar API to `sklearn`'s style.

```

1 # Implementation of kNN class
2 class kNN(object):
3     def __init__(self, k=1):
4         self.k = k
5

```

```
6     def fit(self, yX, norm='l2'):
7         '''
8         stick to MNIST's format: N * (1 + p)
9         '''
10        self.data = yX # lazy train
11        self.norm = norm
12
13    def _get_nn(self, x, k=None):
14        if k is None:
15            k = self.k
16
17        if self.norm is 'l2':
18            penalty_list = np.sqrt(np.sum((self.data[:, 1:] -
19                                           x) ** 2, axis=1))
20        elif self.norm is 'l1':
21            penalty_list = np.sum(np.abs(self.data[:, 1:] - x),
22                                 axis=1)
23
24        if k > 1:
25            idx = np.argsort(penalty_list, axis=0)[:k]
26        else:
27            idx = np.array([np.argmin(penalty_list)]) # number
28            --> [number]
29
30        return self.data[idx, :], penalty_list[idx]
31
32    def predict(self, x):
33        neighbors, _ = self._get_nn(x)
34        if self.k == 1:
35            return neighbors[0,0]
36        else:
37            Y = neighbors[:, 0]
38            _ = {}
39            for y in list(Y):
40                _[y] = _.get(y, 0) + 1
41            ret = sorted(_.iteritems(), key=lambda kv: kv[1],
42                        reverse=True)
43            return ret[0][0]
44
45    def score(self, X, y):
46        '''
```

```

43         Keep to sklearn's API
44         '''
45         return np.average(np.apply_along_axis(self.predict, 1,
X) == y)

```

Then we need to pick appropriate parameter k under the training data.

Theoretically, kNN approximates *Bayes Classifier* with relaxed assumption in which the given X in condition probability is replaced by the neighborhood of it. Considering that the pdfs of mnist overlap little, we intuitively think that smaller k will performance better. As I used free *Amazon AWS*, which has only 1 core and little memory, to run my code, I here run this

```

1 def run_one_test(cls, yX):
2     yX_train, yX_test = cross_validation.train_test_split(yX)
3     cls.fit(yX_train)
4     return cls.score(yX_test[:, 1:], yX_test[:, 0])
5
6 best = -1
7 best_k = -1
8 for kk in xrange(3, 15, 2):
9     s = run_one_test(kNN(k=kk), data[:8000])
10    mlog("k is %d. score is %f." % (kk, s))
11    if s > best:
12        best = s
13        best_k = kk
14 mlog("best k is %d" % best_k)

```

to pick appropriate k under smaller dataset as shown in Fig.6.

```

[INFO]: k is 3. score is 0.936000.
[INFO]: k is 5. score is 0.930000.
[INFO]: k is 7. score is 0.928500.
[INFO]: k is 9. score is 0.931000.
[INFO]: k is 11. score is 0.930000.
[INFO]: k is 13. score is 0.913500.
[INFO]: best k is 3
CPU times: user 2min 44s, sys: 1min 20s, total: 4min 5s
Wall time: 4min 6s

```

Figure 6: Log of choosing k .

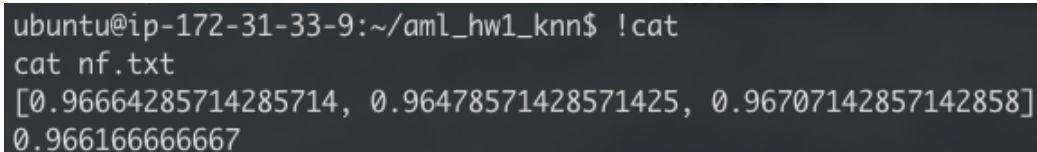
Cross Validation After setting k fixed at 3, we run 3-folds cross validation on the train set.

```

1 def run_cv_test(cls, yX, n_folds=3):
2     accu = []
3     for tr_idx, te_idx in cross_validation.KFold(len(yX),
4         n_folds=n_folds):
5         train = data[tr_idx] # yX
6         test = data[te_idx]
7         cls.fit(train)
8         accu.append(cls.score(test[:, 1:], test[:, 0]))
9     return accu
10 cls = kNN(k=3)
11 accu = run_cv_test(cls, data, n_folds=3)
12 print accu
13 print np.average(accu)

```

The average score obtained by cross validation is shown in Fig.7. From this,



```

ubuntu@ip-172-31-33-9:~/aml_hw1_knn$ !cat
cat nf.txt
[0.96664285714285714, 0.96478571428571425, 0.96707142857142858]
0.966166666667

```

Figure 7: N-fold cross validation.

we could expect that the model will get similar score on test set.

Confusion Matrix Here we generated the confusion matrix, as shown in Fig.8, over splitting dataset.

```

1 def genCFMat(yx):
2     label = yx[0]
3     pred = cls.predict(yx[1:])
4     cf_mat[label, pred] += 1
5
6 yX_train, yX_test = cross_validation.train_test_split(data)
7 cls = kNN(k=3)
8 cls.fit(yX_train)
9 cf_mat = np.zeros((10, 10), dtype=np.int32)
10 for i in xrange(len(yX_test)):
11     genCFMat(yX_test[i])

```

[[10	11	0	3	1	0	0	3	0	0	2]
[0	116	7	2	0	0	0	1	1	2	1	0]
[9	13	100	9	2	0	0	0	0	14	4	2]
[1	4	5	101	9	0	8	0	5	15	8]	
[0	11	0	0	102	8	0	1	1	2	33]	
[2	3	0	13	0	894	12	0	4	7]		
[5	1	0	0	1	4	1014	0	0	0]		
[0	18	2	0	3	0	0	1083	0	9]		
[4	9	4	16	0	14	6	3	943	9]		
[2	1	0	5	9	4	0	10	1	997]		

Figure 8: Confusion matrix.

Through the logarithmic values of confusion matrix, it would be more straightforward to get the most tricky digit to classify. As shown in Fig.9, the most tricky digit is 4, which is comparatively easy to be classified by kNN model as 9. Besides, 7 and 8 were also by mistake classified as 1 and 3, respectively.

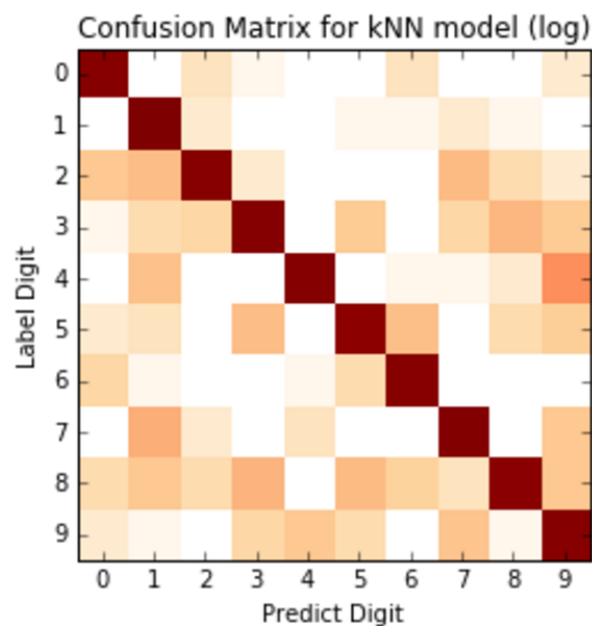


Figure 9: Logarithmic confusion matrix image.

Submit to Kaggle Finally, we ran our model on test set,

```

1 mdl_sub = kNN(k=3)
2 mdl_sub.fit(data)
3 y_pred = np.apply_along_axis(mdl_sub.predict, 1, test_data)

```

```

4 ret = np.column_stack((np.arange(1, len(y_pred) + 1),
    y_pred)).astype(np.int)
5 np.savetxt('testResult_k_3.csv', ret, delimiter=',', fmt='%d',
    header='ImageId,Label', comments='')

```

and then submitted the result to Kaggle. We received the model's performance on test dataset from Kaggle, as demonstrated in Fig.10. Obviously, kNN model is slow and not accurate enough. We could use other models such as LR or CNN to improve the accuracy as well as training/predicting speed.

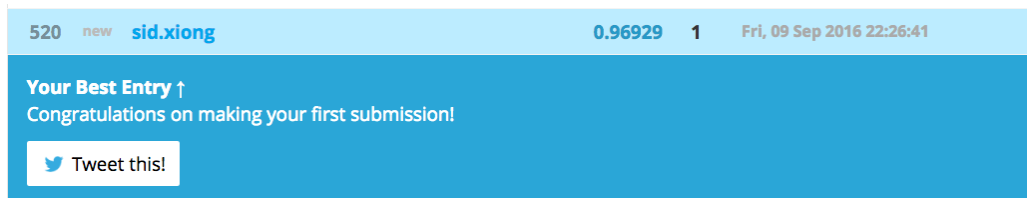


Figure 10: Performance on test set.

1.2 Titanic

The original data set has in total 11 features including obviously unnecessary ones such as {"Passenger", "Name", "Ticket", "Embarked"}. Information in feature {"Cabin"} is also conveyed by {"Pclass"}. Therefore, based on their descriptions, features we chose are

$$\{Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare'\}. \quad (1)$$

Firstly, we imported libraries that would be used and loaded original dataset.

```

1 from sklearn.linear_model import LogisticRegression
2 from sklearn import cross_validation, preprocessing
3 import numpy as np
4 import pandas as pd
5 import matplotlib.pyplot as plt
6
7 df = pd.read_csv('train.csv')
8 _ = df.columns.values.tolist()
9 _.remove('PassengerId'); _.remove('Survived'); _.remove('Name')
10 _.remove('Ticket'); _.remove('Embarked'); _.remove('Cabin')

```

```
11 used_fieds = _
```

Then, we transformed text in feature *Sex* into numeric ones, i.e. {0, 1}.

```
1 le = preprocessing.LabelEncoder()
2 sex_dis = le.fit(["male", "female"]).transform(df.Sex)
3 df.Sex = sex_dis
```

It was noticeable that there existed a large amount of missing data in the original dataset. We, by intuition, replaced missing data by the mean value of feature it belonged to over the entire dataset, via this code.

```
1 for f in used_fieds:
2     df[f] = df[f].fillna(np.mean(df[f]))
```

As numeric values in columns like *SibSp*, *ParCh*, and *Fare* might be large, we normalized there columns.

```
1 scaler_sibsp = preprocessing.StandardScaler().fit(df['SibSp'])
2 df['SibSp'] = scaler_sibsp.transform(df['SibSp'])
3 scaler_parch = preprocessing.StandardScaler().fit(df['Parch'])
4 df['Parch'] = scaler_parch.transform(df['Parch'])
5 scaler_fare = preprocessing.StandardScaler().fit(df['Fare'])
6 df['Fare'] = scaler_fare.transform(df['Fare'])
```

Eventually, we trained a LR classifier with the help of *scikit*, and ran *N-Fold* validation to get the average accuracy obtained by model.

```
1 X = df[used_fieds].values[:, :-1]
2 y = df['Survived']
3 accu = []
4 for tr_idx, te_idx in cross_validation.KFold(len(X),
5     n_folds=5):
6     X_train = X[tr_idx]
7     y_train = y[tr_idx]
8     X_test = X[te_idx]
9     y_test = y[te_idx]
10     cls = LogisticRegression()
11     cls.fit(X_train, y_train)
12     accu.append(cls.score(X_test, y_test))
13 print accu
14 print np.average(accu)
```

Setting $N = 5$, the accuracy was:

```

1  accu = [0.81005586592178769, 0.797752808988764,
2         0.8033707865168539, 0.7471910112359551, 0.8370786516853933]
3  average accuracy is 0.79908982487

```

Submission	Files	Public Score
Tue, 13 Sep 2016 05:38:23 Naive LR model with ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare']. Standard scaler transformation on ['Age', 'SibSp', 'Parch', 'Fare'] Edit description	testResult_titanic.csv	0.75120

Figure 11: Performance on *Titanic* test set.

Similar to section. 1.1, we utilized our LR model to predict labels of test set, and then submitted to Kaggle, as shown in Fig.11. The reasons why the score is not good may be that we need to understand more about the features, as well as the selection of models.

2 Written Exercises

2.1 Variance of a sum

Given

$$E(X + Y) = E(X) + E(Y), \quad (2)$$

$$\text{var}(X) = E[(X - E(X))^2], \quad (3)$$

$$\text{cov}(X, Y) = E[(X - E(X))(Y - E(Y))], \quad (4)$$

we could derived that

$$\begin{aligned}
 \text{var}(X + Y) &= E[(X + Y - E(X + Y))^2] \\
 &= E[(X - E(X) + Y - E(Y))^2] \\
 &= E(X - E(X))^2 + (Y - E(Y))^2 + 2(X - E(X))(Y - E(Y)) \\
 &= \text{var}(X) + \text{var}(Y) + 2\text{cov}(X, Y).
 \end{aligned} \quad (5)$$

2.2 Bayes rule for medical diagnosis

If we denote

X : test positive, and x : test result,
 Y : have disease, and y : disease status,

then according to *Bayes Rule*, we have

$$\begin{aligned}
 Pr(Y|X) &= \frac{Pr(X|Y)Pr(Y)}{Pr(X)} \\
 &= \frac{Pr(X|Y)Pr(Y)}{\sum_y Pr(x|y)Pr(y)} \\
 &= \frac{0.99 * 0.0001}{0.99 * 0.0001 + (1 - 0.99) * 0.9999} \\
 &= 0.0098.
 \end{aligned} \tag{6}$$

2.3 Gradient and Hessian matrix

(a). **Derivative of sigmoid function** Since $\sigma(x) = \frac{1}{1+e^{-x}}$, the derivative of $\sigma(x)$ can be derived as

$$\begin{aligned}
 \frac{d\sigma(x)}{dx} &= -\frac{1}{(1+e^{-x})^2}e^{-x}(-1) \\
 &= \frac{e^{-x}}{1+e^{-x}} \frac{1}{1+e^{-x}} \\
 &= (1-\sigma(x))\sigma(x)
 \end{aligned} \tag{7}$$

(b). **Gradient of the log likelihood** The log likelihood can be expressed as

$$l(\beta) = \sum_{i=1}^N \{y_i \beta^T x_i - \log(1 + e^{\beta^T x_i})\}. \tag{8}$$

Given that

$$\begin{aligned}
 p(x; \beta) &= Pr(G = 1|X = x; \beta) \\
 &= \frac{1}{1 + e^{-\beta^T x}},
 \end{aligned} \tag{9}$$

so its derivative is

$$\begin{aligned}
\frac{dl(\beta)}{d\beta} &= \sum_{i=1}^N \left\{ y_i x_i - \frac{1}{1 + e^{\beta^T x_i}} e^{\beta^T x_i} x_i \right\} \\
&= \sum_{i=1}^N \left\{ x_i \left(y_i - \frac{1}{e^{-\beta^T x_i} + 1} \right) \right\} \\
&= \sum_{i=1}^N \{ x_i (y_i - p(x_i; \beta)) \}
\end{aligned} \tag{10}$$

(c). **Positive definite of $\mathbf{X}^T \mathbf{W} \mathbf{X}$** For any non-zero vector \mathbf{a} ,

$$\mathbf{a}^T \mathbf{X}^T \mathbf{W} \mathbf{X} \mathbf{a} = (\mathbf{X} \mathbf{a})^T \mathbf{W} (\mathbf{X} \mathbf{a}) \tag{11}$$

Since \mathbf{W} is diagonal matrix of weights $p(1-p)$ on its i^{th} element, Eq.11 can be derived as the weighted sum of all the L_2 norm of the row vectors in matrix $\mathbf{X} \mathbf{a}$, with w_i in \mathbf{W} as weights. Eq.11 can be further derived as

$$\mathbf{a}^T \mathbf{X}^T \mathbf{W} \mathbf{X} \mathbf{a} = \sum_{i=1}^N w_i \left(\sum_{j=1}^p x_{ij} a_j \right)^2. \tag{12}$$

Geometrically, as row vectors in \mathbf{X} are different samples which are apparently not all "parallel", the row vectors in $\mathbf{X} \mathbf{a}$ will not be all zeros. Therefore, with the given condition that $p(1-p) > 0$, the weighted sum is greater than zero; hence positive definite is $\mathbf{X}^T \mathbf{W} \mathbf{X}$.

3 Attachments Details

hw1_knn.ipynb and hw1_titanic.ipynb in hw1_sidxiong.zip