CS 5785 Applied Machine Learning
Homework 1

Wenyi Chu wc625
Qixin Ding qd49

September 26, 2019

1 PROGRAMMING EXERCISES

1.1 Digit Recognizer Competition

In this assignment, we conducted lazy learning algorithms (Nearest and k-nearest neighbor) on the famous MNIST digit data set. A general idea of this method is to store lots of 28×28 pixel data in the memory and when an unknown digit image (test instance) comes in, simply compare it to all of the stored values, supply a distance function and return a label based on nearest neighbor(s).

Write a function to display an MNIST digit. Display one of each digit.

See the source code attached below:

```
# a function to display an MNIST digit
def display_digit(index):
    plt.figure(figsize=(1, 1))
    ax = plt.subplot(1,1,1)
    ax.imshow(train_data[index].reshape(28,28), cmap='gray')

# a function to display all MNIST digits, from 0 to 9
def display_digits():
#    digit_index = [1,0,16,7,3,8,21,6,10,11]
    plt.figure(figsize=(10, 5))
    for i in range(10):
        ax = plt.subplot(2, 5, i+1)
        ax.imshow(train_data[all_digit_index[i]].reshape(28,28),
    cmap='gray')
```

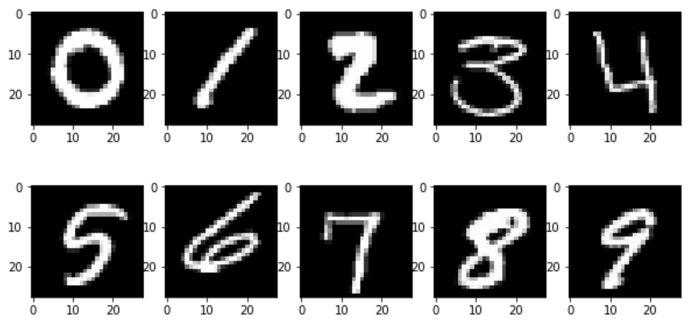


Figure 1: Displays one of each digit.

Prior probability

In this case, prior for a given class (e.g. 0,1,2,...9) = (number of samples in the class) / (total number of samples). See the source code attached below:

```python
# display a normalized histogram of digit counts
def display_distribution():
    plt.figure(figsize=(10, 5))
    plt.hist(labels, range(11), alpha=0.5, density=True, color='grey
  ', width=0.8)
    plt.xticks(np.arange(0, 10, 1))
    plt.ylim(0, 0.2)
    plt.grid(True)

    plt.xlabel("digits")
    plt.ylabel("distribution")
```

Seen from the histogram, it is kind of uniform across the digits, and evenly distributed.

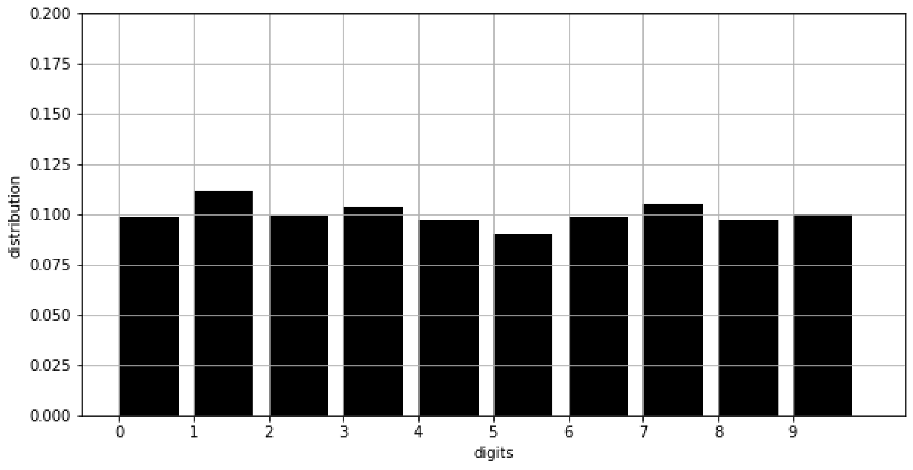


Figure 2: Normalized histogram counting each digit

Nearest Neighbor

Pick one example of each digit from your training data Compute and show the best match (nearest neighbor) in the rest of the training data. We used Eeuclidean to compute the distance between two images and picked the best match (nearest neighbor) based on distances.

Plot the comparison in the Figure.3 and we added an asterisk next to the erroneous examples:

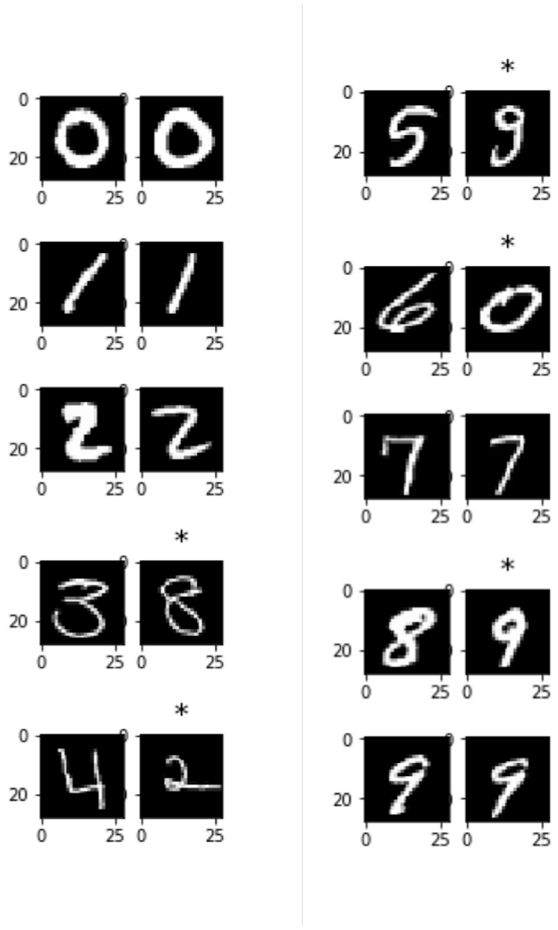


Figure 3: Normalized histogram counting each digit

Since we used L2 distance between the two images' pixel values as the metric, the classification results were not that perfect.

Histogram of the genuine and impostor distance

We added pairwise distances for all genuine matches (0 to 0 and 1 to 1, and imposters matches (0 to 1) and plot distances into a histogram.

See the source code for details:

```
    genuine_dist = []
    impostor_dist = []

    # pairwise distances for all genuine matches
    comb_zeros = list(combinations(zeros, 2))
    comb_ones = list(combinations(ones, 2))
    for i in comb_zeros:
        d = find_distance(i[0],i[1])
        genuine_dist.append(d)
    for i in comb_ones:
        d = find_distance(i[0],i[1])
        genuine_dist.append(d)

    # pairwise distances for all imposter matches
    for i in zeros:
        for j in ones:
            d = find_distance(i,j)
            impostor_dist.append(d)

    # Plot histograms of the genuine and impostor distances on the same set of axes.
    custom_bins = range(500, 4000, 50)

    plt.hist(genuine_dist, alpha=0.5, bins=custom_bins, label='Genuine', color="lightblue",width=50)
    plt.hist(impostor_dist, alpha=0.5, bins=custom_bins, label='Imposter', color="lightgrey",width=50)
```

Seen from the histogram, imposters generally have a bigger distance values than those genuine ones.

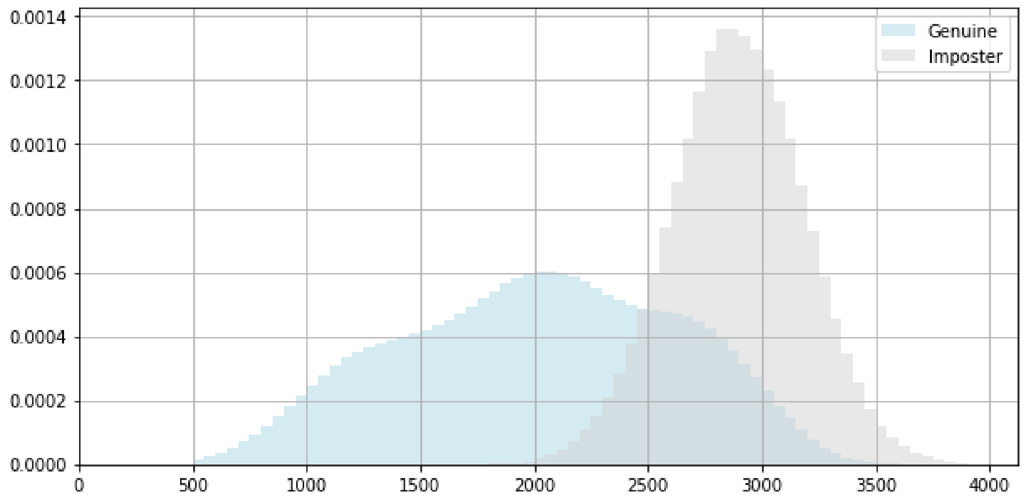


Figure 4: Histograms of the genuine and impostor distance

ROC curve

The ROC curve is a graphical means of representing a binary confusion matrix as a function of classifier threshold.Once committing to a threshold, we generated a corresponding confusion matrix given that threshold and connected into a curve.

See source code below:

```python
# Generate an ROC curve from the above sets of distances
def display_ROC(genuine,impostor):
    TPRs = []
    FPRs = []

    genuine.sort()
    impostor.sort()

    thresholds = range(0, len(impostor), int(len(impostor))/100)

    for i in thresholds:
        fpr = (impostor < impostor[i]).sum()/float(len(impostor))
        tpr = (genuine < impostor[i]).sum()/float(len(genuine))

        TPRs.append(tpr)
        FPRs.append(fpr)
#         print(tpr,fpr)

    plt.figure(figsize=(10, 5))
    plt.plot(FPRs,TPRs, color="lightblue")

    plt.title("ROC Curve")
    plt.xlabel("FPR: False Positive")
    plt.ylabel("TPR: True Positive")
```

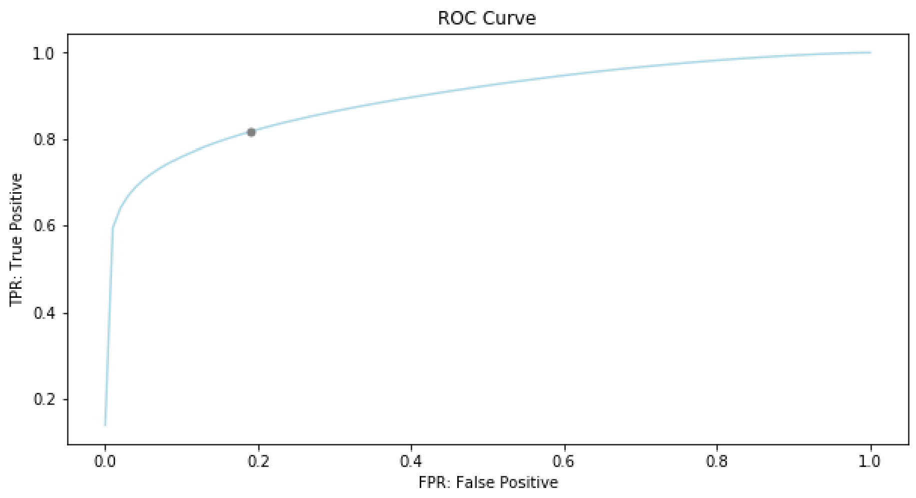


Figure 5: ROC curve marked with ERR=0.1899975

Source code for computing ERR:

```python
    tpr_index = np.abs(np.add(FPRs,TPRs)-1).argmin()

    x = FPRs[tpr_index] #err
    y = TPRs[tpr_index]

    # mark ERR on the curve
    plt.plot([x], [y], marker='o', markersize=5, color="grey")
    plt.show()
```

ERR (Equal Error Rate) = **0.1899975**, but Random Guess Equal Error Rate = **0.5**.

KNN

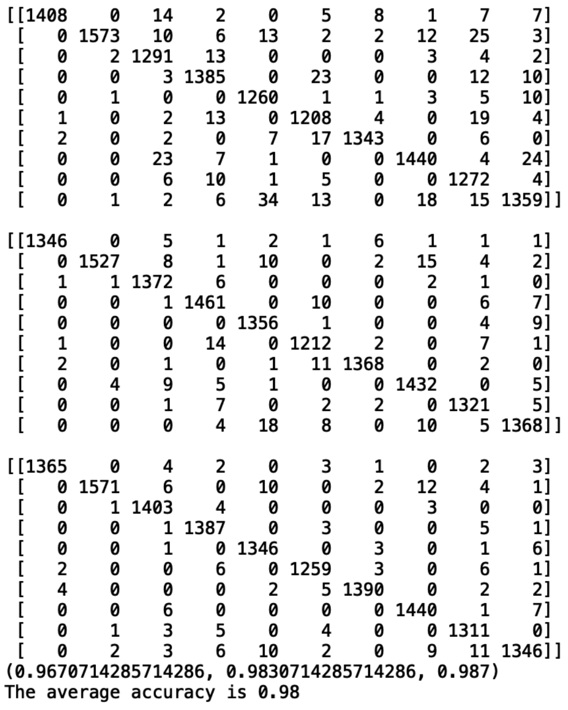
The basic idea of k-nearest neighbors (KNN) is to classify using majority vote among the k neighbors. Our source code is very self-explanatory as below:

```python
def KNN(test, train_data, train_labels, k):
    output = []
    print("Total test case %d" % len(test))
    for i in range(len(test)):
        test_case = test[i]

        # for each test case, iterate in the training data and find
   all distances
        ds = []

        for i in range(len(train_data)):
            d = np.linalg.norm(train_data[i] - test_case)
            ds.append((i, d))

        ds.sort(key=lambda elem: elem[1])

        # find k neighbors
        neighbors = []
        for i in range(k):
            neighbors.append(ds[i][0])

        # find freq
        freq = {}
        for i in range(len(neighbors)):
            response = labels[neighbors[i]]
            if response in freq:
                freq[response] += 1
            else:
                freq[response] = 1
        freq = sorted(freq.items(), key=lambda elem: elem[1],
   reverse=True)

```

```python
        # Classify using majority vote among the k neighbors
        match = freq[0][0]
        output.append(match)
    return output
```

We repeated the 3fold-cross-validation across a range of set a k values, from 1 to 10, and we noticed that an optimal k achieved at **k = 3** with accuracy of **(0.967, 0.983, 0.987)** for all three folds. Overall, the average accuracy is around **98%**.

Below are the snapshots for the confusion matrices:

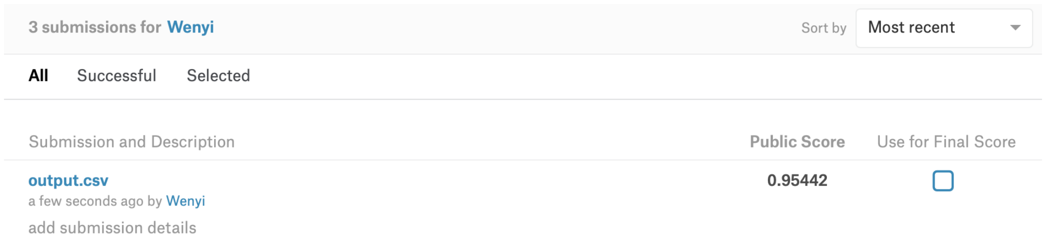


```
[[1408    0   14    2    0    5    8    1    7    7]
 [   0 1573   10    6   13    2    2   12   25    3]
 [   0    2 1291   13    0    0    0    3    4    2]
 [   0    0    3 1385    0   23    0    0   12   10]
 [   0    1    0    0 1260    1    1    3    5   10]
 [   1    0    2   13    0 1208    4    0   19    4]
 [   2    0    2    0    7   17 1343    0    6    0]
 [   0    0   23    7    1    0    0 1440    4   24]
 [   0    0    6   10    1    5    0    0 1272    4]
 [   0    1    2    6   34   13    0   18   15 1359]]

[[1346    0    5    1    2    1    6    1    1    1]
 [   0 1527    8    1   10    0    2   15    4    2]
 [   1    1 1372    6    0    0    0    2    1    0]
 [   0    0    1 1461    0   10    0    0    6    7]
 [   0    0    0    0 1356    1    0    0    4    9]
 [   1    0    0   14    0 1212    2    0    7    1]
 [   2    0    1    0    1   11 1368    0    2    0]
 [   0    4    9    5    1    0    0 1432    0    5]
 [   0    0    1    7    0    2    2    0 1321    5]
 [   0    0    0    4   18    8    0   10    5 1368]]

[[1365    0    4    2    0    3    1    0    2    3]
 [   0 1571    6    0   10    0    2   12    4    1]
 [   0    1 1403    4    0    0    0    3    0    0]
 [   0    0    1 1387    0    3    0    0    5    1]
 [   0    0    1    0 1346    0    3    0    1    6]
 [   2    0    0    6    0 1259    3    0    6    1]
 [   4    0    0    0    2    5 1390    0    2    2]
 [   0    0    6    0    0    0    0 1440    1    7]
 [   0    1    3    5    0    4    0    0 1311    0]
 [   0    2    3    6   10    2    0    9   11 1346]]
(0.9670714285714286, 0.9830714285714286, 0.987)
The average accuracy is 0.98
```

Figure 6: Confusion Matrix (3-fold)

Seen from the second matrix, we noticed that digit 8 is particularly hard to classify; from the second matrix, we noticed that digits such as 2 and 9 are relatively tricky to classify.

Then we submitted our results to Kaggle and it reported an accuracy of **95.4**%.



3 submissions for Wenyi | Sort by: Most recent

All | Successful | Selected

Submission and Description	Public Score	Use for Final Score
output.csv a few seconds ago by Wenyi add submission details	0.95442	☐

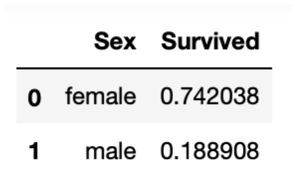
Figure 7: Submission to Kaggle

1.2 The Titanic Disaster

1.2.1 Logistic regression

Using logistic regression, try to predict whether a passenger survived the disaster.
Selecting features: To confirm some of our observations and assumptions, we pull out some correlations by pivoting certain features against each other. Before that, we filled in cells with empty values (e.g. fill empty ages with mean, etc.). Here's the breakdown of what features we picked:

Sex: We also noticed that Sex=female had very high survival rate at 74%.



	Sex	Survived
0	female	0.742038
1	male	0.188908

Figure 8: Sex vs survival rate

```
    # converted Age feature into a numerical (int)
    for dataset in combine:
        dataset[’Sex’] = dataset[’Sex’].map( {’female’: 1, ’male’:
 0} ).astype(int)
```

To fully use the sex data, we did a bit of data cleaning to convert the string variables into numerical ones, showing as above.

Pclass: We observed a significant correlation between Pclass=1 and Survived and we decided to include this feature in our model.

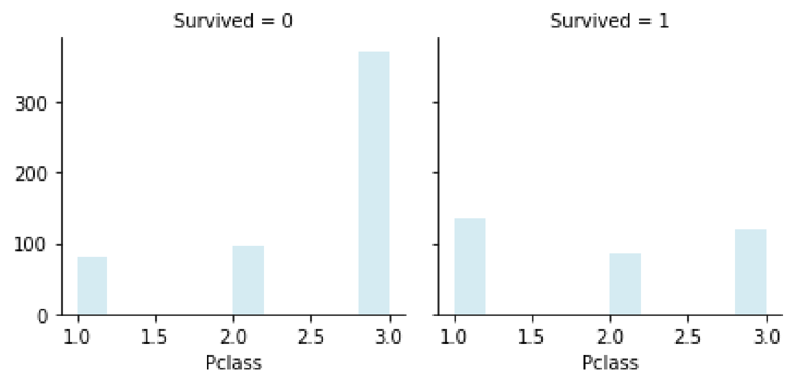


Figure 9: Pclass vs survival count

Fare: We assumed that passengers who paid higher fare had better survival and the dataset (Fig.9) confirmed our assumptions.

Embark: Very similarly, passengers who embark at Cherbourg had better survival rate (Fig.9).

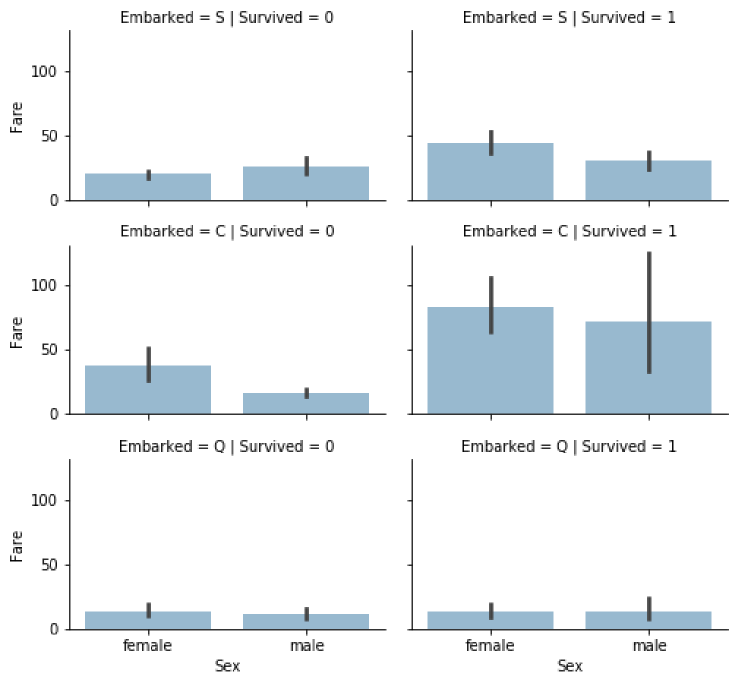


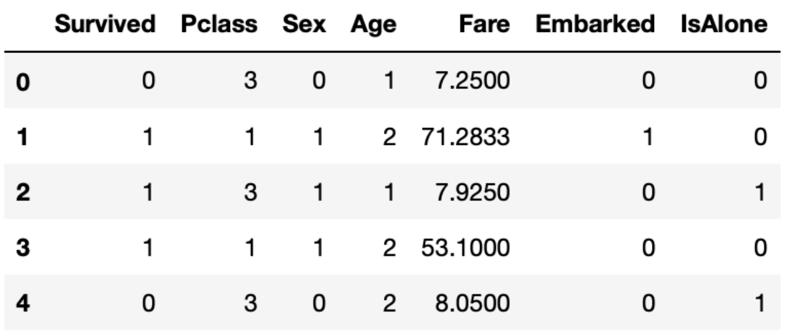
Figure 10: Fare/Embark vs survival count

SibSp and Parch: We did not see significant correlation for these two features. But it doesn't mean to be dropped right away, we though it may be better if we combine these two into a new feature: if travel **alone**. See the code attached below for details.

```
# created a new feature (if travel alone) by combining sibsp and parch
for dataset in combine:
    dataset['Family'] = dataset['SibSp'] + dataset['Parch'] + 1

for dataset in combine:
    dataset['IsAlone'] = 0
    dataset.loc[dataset['Family'] == 1, 'IsAlone'] = 1
```

Age: We converted age feature into a categorical feature: 0-10, 10-20, 20-30, 30-40, but before that, we need to fill in the empty cells with the median of all passengers' age.

We also dropped **Ticket**, **Cabin** and other unnecessary features. Here's an overview of how the new training set looks like:



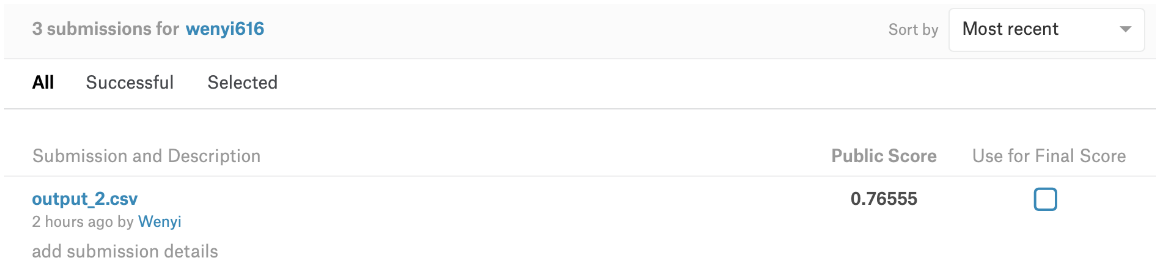
	Survived	Pclass	Sex	Age	Fare	Embarked	IsAlone
0	0	3	0	1	7.2500	0	0
1	1	1	1	2	71.2833	1	0
2	1	3	1	1	7.9250	0	1
3	1	1	1	2	53.1000	0	0
4	0	3	0	2	8.0500	0	1

Figure 11: A brief of new TrainingSet after picking/combining/dropping features

Then we applied sklearn's logisticRegresssion to train dataset.

```
# use sklearn's logisticRegresssion to train dataset
    logreg = LogisticRegression()
    logreg.fit(train, train_label)
    pred = logreg.predict(test)

    out = pd.DataFrame(columns=['PassengerId', 'Survived'])
    out['PassengerId'] = passengerId
    out['Survived'] = pred.astype(int)
    out.to_csv('output_2.csv', index=False)
```

Submission to Kaggle reported an accuracy of **76.076**%.



3 submissions for wenyi616

Sort by Most recent

All Successful Selected

Submission and Description	Public Score	Use for Final Score
output_2.csv 2 hours ago by Wenyi add submission details	0.76555	☐

Figure 12: Submission to Kaggle

2 WRITTEN EXERCISES

Exercise 2.1

Variance of a sum. Show that the variance of a sum is $var[X - Y] = var[X] + var[Y] - 2cov[X, Y]$, where $cov[X, Y]$ is the covariance between random variables X and Y.

Proof.

$$\begin{aligned} var[X] &= E(X^2) - (E(X))^2 \\ var[Y] &= E(Y^2) - (E(Y))^2 \end{aligned}$$

$$\begin{aligned} cov[X+Y] &= E((X - E(X))(Y - E(Y))) \\ &= E(XY - E(X)Y - XE(Y) + E(X)E(Y)) \end{aligned}$$

$$\begin{aligned} var[X-Y] &= E((X-Y) - E(X-Y)^2) \\ &= E(X-Y)^2 - (E(X-Y))^2 \\ &= E(X^2 + Y^2 - 2XY) - (E(X-Y))^2 \end{aligned}$$

$$\begin{aligned} var[X] + var[Y] - 2cov[X,Y] &= E(X^2) - (E(X))^2 + E(Y^2) - (E(Y))^2 \\ &- 2E(XY) - 2E(X)E(Y) - 2E(X)E(Y) - 2E(X)E(Y) \\ &= E(X^2) + E(Y^2) - 2E(XY) - ((E(X))^2 + (E(Y))^2 - 2E(X)E(Y)) \\ &= E(X^2 + Y^2 - 2XY) - (E(X) - E(Y))^2 \\ &= E(X^2 + Y^2 - 2XY) - (E(X-Y))^2 \\ &= var[X-Y] \end{aligned} \tag{1}$$

Exercise 2.2

Bayes rule for quality control. You're the foreman at a factory making ten million widgets per year. As a quality control step before shipment, you create a detector that tests for defective widgets before sending them to customers. The test is uniformly 95% accurate, meaning that the probability of testing positive given that the widget is defective is 0.95, as is the probability of testing negative given that the widget is not defective. Further, only one in 100,000 widgets is actually defective.

(a) Suppose the test shows that a widget is defective. What are the chances that it's actually defective given the test result?

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2}$$

$$P(defective|positive) = \frac{P(positive|defective)P(defective)}{P(positive)} \tag{3}$$

$$(95\% \cdot 0.00001) \div (95\% \cdot 0.00001 + 5\% \cdot (1 - 0.00001)) = 0.01899\% \tag{4}$$

Therefore, the chances that it's actually defective given the test result is **0.01899%**.

(b) If we throw out all widgets that are test defective, how many good widgets are thrown away per year? How many bad widgets are still shipped to customers each year?

$$\begin{gathered} 10000000 \cdot 5\% \cdot (1 - 0.00001) = 499995 \\ 10000000 \cdot 5\% \cdot 0.00001 = \mathbf{5} \end{gathered} \tag{5}$$

There are **5** bad widgets are still shipped to customers each year.

Exercise 2.3

In k-nearest neighbors,

(a) Describe what happens to the average 0-1 prediction error on the training data when the neighbor count k varies from n to 1. (In this case, the prediction for training data point x_i includes (x_i, y_i) as part of the example training data used by kNN.)

ANS: As neighbor count k varies from n to 1, the average 0-1 prediction error goes down, decreasing from 0.5 when $k = n$ and gradually approaches to 0 when $k = 1$.

(b) We randomly choose half of the data to be removed from the training data, train on the remaining half, and test on the held-out half. Predict and explain with a sketch how the average 0-1 prediction error on the held-out validation set might change when k varies? Explain your reasoning.

ANS: As the sketch below suggests, the prediction error will decrease as the k decreases and eventually reaches an optimal point (usually happens when k=3 to k=10) and then increase (a smallar value of k means that noise will have a higher influence on the result).

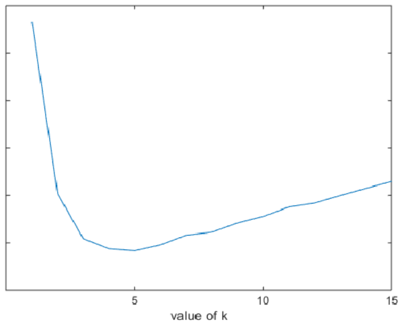


Figure 13: A sketch of relationship between hoice of k and prediction error

(c) We wish to choose k by cross-validation and are considering how many folds to use. Compare both the computational requirements and the validation accuracy of using different numbers of folds for kNN and recommend an appropriate number.

ANS: We wish to choose k using n-fold cross-validation, but before that how to choose n (aka. how many folds to use). In this assignment, we picked 3-fold but the choice of n is favorably ranging from 5 to 10. A too large k may require a relatively large computational requirement, but a small k may lead to high variance or high bias, therefore overestimating the resulting accuracy.

(d) In kNN, once k is determined, all of the k-nearest neighbors are weighted equally in deciding the class label. This may be inappropriate when k is large. Suggest a modification to the algorithm that avoids this caveat.

ANS: In the situation when k is large, the neighbor should be weighted based on its distance to the data point. Neighbors with lower distance need to be associated with a heavier weight.

(e) Give two reasons why kNN may be undesirable when the input dimension is high.

ANS: Since kNN is memory-based - it scans historical database each time to generate a prediction for each test instances. Therefore, when the input dimension is high, it requires a way larger training set to help make accurate decision, which may or not always be available. Even if the training set is large enough, singly examining the distances can be demanding and slow. Also, when input dimension is high, points in the space tend not to be close to each other and the notion of nearest and far neighbors becomes even harder to define.