

## Homework 1

# CS 5785 Applied Machine Learning

---

Xialin Shen <[xs293@cornell.edu](mailto:xs293@cornell.edu)>

An Le <[aql6@cornell.edu](mailto:aql6@cornell.edu)>

13th September, 2017

## PART A - PROGRAMMING EXERCISE

### Question 1 - Digit Classifier

(a) Join the Digit Recognizer competition on Kaggle. Download the training and test data. The competition page describes how these files are formatted.

```
In [1]: 1 import pandas as pd
        2 import matplotlib.pyplot as plt, matplotlib.image as mpimg
        3 import numpy as np
        4 %matplotlib inline
```

#### 1 a) Reading data ¶

```
In [2]: 1 labeled_images = pd.read_csv('./data/train.csv')
        2 images = labeled_images.iloc[0:,1:]
        3 labels = labeled_images.iloc[0:,:1]
```

```
In [3]: 1 images.shape
```

```
Out[3]: (42000, 784)
```

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

## 2 b) Display an MNIST digit

```
: 1 def display_image_at(i, images, labels):  
2     img = images.iloc[i].as_matrix()  
3     img = img.reshape((28,28))  
4     plt.imshow(img,cmap='gray')  
5     plt.title(labels.iloc[i,0])
```

### 2.1 Display one of each digit

```
: 1 def search_image_index(n):  
2     if(n < 0 or n > 9):  
3         print("Data not available")  
4     for i in range(0,len(labels)):  
5         if(n == labels.iloc[i,0]):  
6             return i  
7  
8 def show_digit(n):  
9     display_image_at(search_image_index(n), images, labels)
```

```
: 1 index = 0  
2 while(index < 10):  
3     plt.subplot(1,10,index+1)  
4     show_digit(index)  
5     index = index + 1
```



(c) Examine the prior probability of the classes in the training data. Is it uniform across the digits? Display a normalized histogram of digit counts. Is it even?

### 3.1 Prior Probability

```
1 count_arr = [0] * 10
2 for i in range(0, len(labels)):
3     count_arr[labels.iloc[i,0]] = count_arr[labels.iloc[i,0]] + 1;
4
5 for i in range(0,10):
6     print(i, ":", (count_arr[i]/len(labels)))
7
```

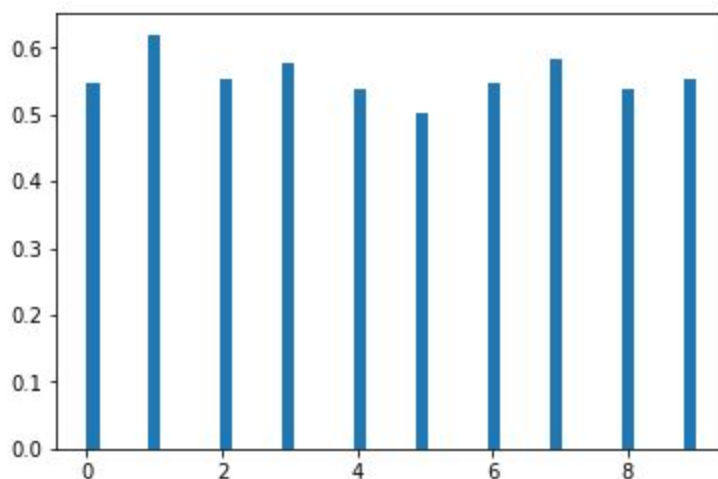
0 : 0.09838095238095237  
1 : 0.11152380952380953  
2 : 0.09945238095238096  
3 : 0.1035952380952381  
4 : 0.09695238095238096  
5 : 0.09035714285714286  
6 : 0.0985  
7 : 0.10478571428571429  
8 : 0.09673809523809523  
9 : 0.09971428571428571

**3.2 As the hist shows, the digits are uniform distributed.**

**3.3 It's also even since the frequencies of labels are roughly equal.**

### 3.4 Display a normalized hist

```
1 plt.hist(labels['label'], 50, normed=1)
```

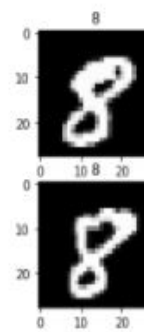
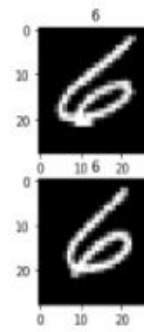
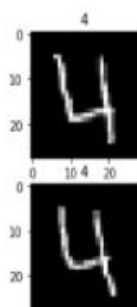
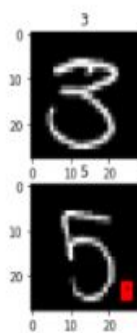
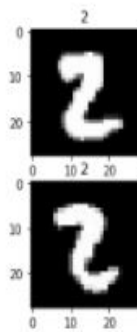
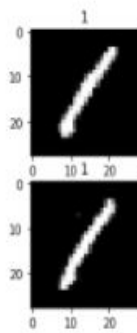
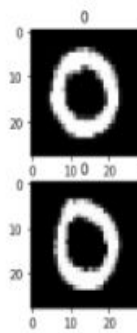


(d) Pick one example of each digit from your training data. Then, for each sample digit, compute and show the best match (nearest neighbor) between your chosen sample and the rest of the training data. Use L2 distance between the two images' pixel values as the metric. This probably won't be perfect, so add an asterisk next to the erroneous examples (if any).

#### 4 d) Find neighbor and print

```
1 def display_image_with_comp(i, images, labels, n):
2     img = images.iloc[i].as_matrix()
3     img = img.reshape((28,28))
4
5     if labels.iloc[i,0] != n:
6         plt.text(24,24, '*', fontsize=8, bbox=dict(facecolor='red', alpha=1))
7
8     plt.imshow(img, cmap='gray')
9     plt.title(labels.iloc[i,0])

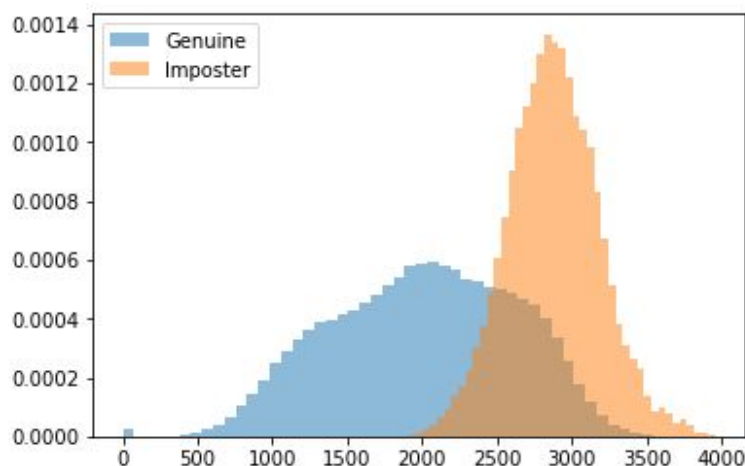
1 def nearest_neighbor(n):
2     image_index = search_image_index(n)
3     plt.subplot(2,1,1)
4     display_image_at(image_index, images, labels)
5
6     inX = images.iloc[image_index]
7     np_inX = inX.as_matrix()
8
9     images_wo_x = images.copy()
10    images_wo_x.drop(image_index, inplace=True)
11    np_images_wo_x = images_wo_x.as_matrix()
12
13    labels_wo_x = labels.copy()
14    labels_wo_x.drop(image_index, inplace=True)
15    np_labels_wo_x = labels_wo_x.as_matrix()
16
17    # Calculate distance to each pixels from inX to images
18    imagesSize = np_images_wo_x.shape[0]
19    diffMat = np.tile(np_inX, (imagesSize,1)) - np_images_wo_x
20    sqDiffMat = diffMat**2
21    sqDistances = sqDiffMat.sum(axis=1)
22    distances = sqDistances**0.5
23    sortedDistanceIndicies = distances.argsort()
24
25    plt.subplot(2,1,2)
26    display_image_with_comp(sortedDistanceIndicies[0], images_wo_x, labels_wo_x, n)
27
```



(e) Consider the case of binary comparison between the digits 0 and 1. Ignoring all the other digits, compute the pairwise distances for all genuine matches and all impostor matches, again using the L2 norm. Plot histograms of the genuine and impostor distances on the same set of axes.

### 5 e) 0 and 1 binary comparison

```
1 zero_idx = []
2 one_idx = []
3
4 for idx, val in enumerate(labels.as_matrix()):
5     if val == 0:
6         zero_idx.append(idx)
7     elif val == 1:
8         one_idx.append(idx)
9
10 zero_images = images.iloc[zero_idx]
11 one_images = images.iloc[one_idx]
12
13 from sklearn.metrics import pairwise_distances
14
15 zero_one = pairwise_distances(zero_images, one_images, metric='euclidean')
16 one_zero = pairwise_distances(one_images, zero_images, metric='euclidean')
17 zero_zero = pairwise_distances(zero_images, zero_images, metric='euclidean')
18 one_one = pairwise_distances(one_images, one_images, metric='euclidean')
19
20 genuine = np.append(zero_zero.reshape((zero_zero.size, 1)), one_one.reshape((one_one.size, 1)))
21 imposter = np.append(zero_one.reshape((zero_one.size, 1)), one_zero.reshape((one_zero.size, 1)))
22
23 fig = plt.figure()
24 ax1 = fig.add_subplot(111)
25 ax1.hist(genuine, 50, alpha=0.5, label='Genuine', normed=True)
26 ax1.hist(imposter, 50, alpha=0.5, label='Imposter', normed=True)
27 plt.legend(loc='upper left')
28 plt.savefig("Binary Comparison")
29 plt.show()
```





(f) Generate an ROC curve from the above sets of distances. What is the equal error rate? What is the error rate of a classifier that simply guesses randomly?

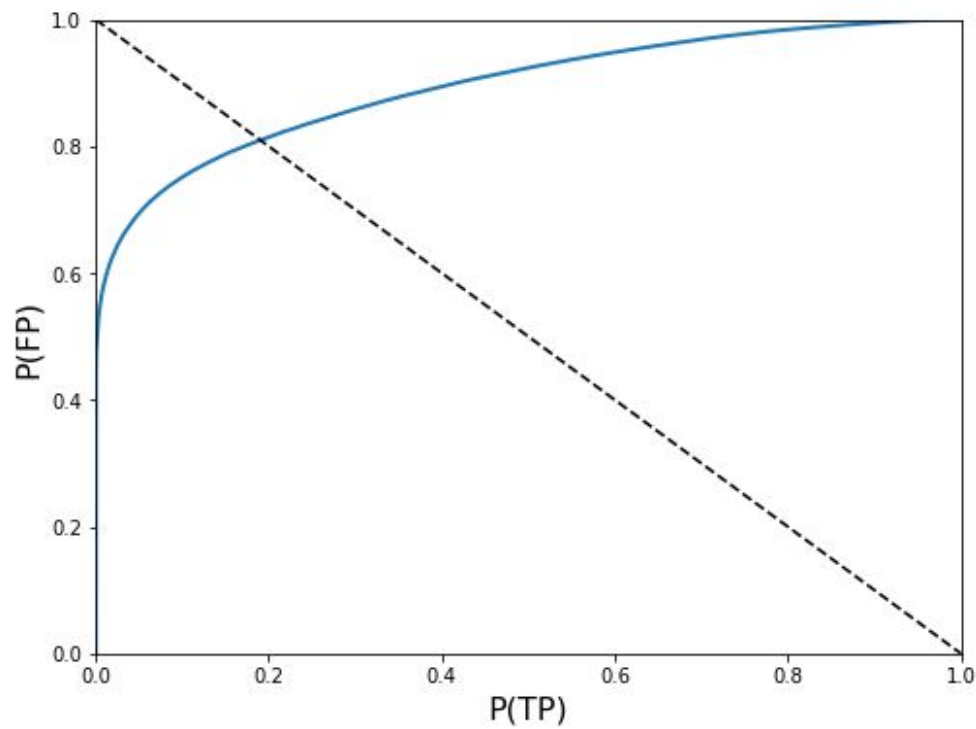
## 6 f) ROC curve

```
: 1 minDistance = min(min(genuine),min(imposter))
2 maxDistance = max(max(genuine), max(imposter))
3
```

```
: 1 TPR_values = []
2 FPR_values = []
3
4 for threshold in range(int(minDistance), int(maxDistance)):
5     TP = len([i for i,x in enumerate(genuine) if x < threshold])
6     TPR = TP / len(genuine)
7     TPR_values.append(TPR)
8
9     FP = len([i for i,x in enumerate(imposter) if x < threshold])
10    FPR = FP / len(imposter)
11    FPR_values.append(FPR)
12
```

```
: 1 def plot_roc_curve(fpr, tpr):
2     plt.plot(fpr, tpr, linewidth=2)
3     plt.plot([0, 1], [1, 0], 'k--')
4     plt.axis([0, 1, 0, 1])
5     plt.xlabel('P(TP)', fontsize=16)
6     plt.ylabel('P(FP)', fontsize=16)
7
8     plt.figure(figsize=(8, 6))
9     plot_roc_curve(FPR_values, TPR_values)
10    plt.savefig("roc_curve_plot")
11    plt.show()
```





---

**6.1 The EER is at (0.2, 0.8)**

**6.2 The EER when guessing randomly in this case is 50% (0 or 1 result)**

(g) Implement a K-NN classifier. (You cannot use external libraries for this question; it should be your own implementation.)

## 7 g) KNN Implementation

```
In [48]: 1 import operator
          2
          3 def kNN(inX, dataSet, labels, k):
          4     dataSetSize = dataSet.shape[0]
          5     diffMat = np.tile(inX, (dataSetSize,1)) - dataSet
          6     sqDiffMat = diffMat**2
          7     sqDistances = sqDiffMat.sum(axis=1)
          8     distances = sqDistances**0.5
          9     sortedDistIndices = distances.argsort()
          10
          11     classCount = {}
          12     for i in range(k):
          13         voteLabel = labels.item(sortedDistIndices[i])
          14         classCount[voteLabel] = classCount.get(voteLabel, 0) + 1
          15
          16     sortedClassCount = sorted(classCount.items(),
          17                               key=operator.itemgetter(1),
          18                               reverse=True)
          19     return sortedClassCount[0][0]

In [49]: 1 inX = images.iloc[0]
          2 dataSet = images.iloc[1:50,0:]
          3 labelSet = labels.iloc[1:50,0:]

In [83]: 1 print("Expected: ",labels.iloc[0,0])
          2 result = kNN(inX.as_matrix(), dataSet.as_matrix(), labelSet.as_matrix())
          3 print("Actual: ",result)

Expected: 1
Actual: 1
```

(h) Using the training data for all digits, perform 3 fold cross-validation on your K-NN classifier and report your average accuracy.

## 8 h) 3 fold cross-validation

```
In [170]: 1 from sklearn.model_selection import StratifiedKFold
2 from sklearn.base import clone
3
4 def kNNkFoldCV(dataSet, labelSet, k):
5     skfolds = StratifiedKFold(n_splits = k, random_state=42)
6
7     X_train = dataSet.as_matrix()
8     y_train = (labelSet.as_matrix()).reshape(labelSet.shape[0],)
9     result = []
10    y_predict = []
11    y_data = []
12
13    for train_index, test_index in skfolds.split(X_train, y_train):
14        X_train_folds = X_train[train_index]
15        y_train_folds = (y_train[train_index])
16        X_test_fold = X_train[test_index]
17        y_test_fold = (y_train[test_index])
18
19        y_pred = []
20        n_correct = 0
21        for i,x in enumerate(X_test_fold):
22            y = kNN(x, X_train_folds, y_train_folds, 5)
23            y_pred.append(y)
24            y_predict.append(y)
25            y_data.append(y_test_fold[i])
26            if y == y_test_fold[i]:
27                n_correct += 1
28
29        result.append(n_correct/len(y_pred))
30
31    return result,y_predict,y_data
```

```
In [171]: 1 kNN_Scores, y_predict, y_data = kNNkFoldCV(images, labels, 3)
```

```
In [172]: 1 sum = 0
2 for score in kNN_Scores:
3     sum += score
4
5 print("Scores: ", kNN_Scores)
6 print("Average score: ", sum/len(kNN_Scores))
7
```

```
Scores: [0.9652956298200515, 0.9643520502928989, 0.9666380911558794]
Average score: 0.9654285904229433
```

(i) Generate a confusion matrix(of size10×10)from your results.Which digits are particularly tricky to classify?

## 9 i) Generate Confusion Matrix 10x10

```
1 from sklearn.metrics import confusion_matrix
2 confusion_matrix(y_data, y_predict)
```

```
array([[4102,    1,    4,    0,    0,    6,   13,    3,    1,    2],
       [    0, 4657,    9,    1,    1,    1,    3,    8,    1,    3],
       [   30,   55, 3964,   16,    3,    4,    4,   85,   11,    5],
       [    3,   11,   26, 4186,    0,   48,    3,   27,   27,   20],
       [    2,   49,    0,    0, 3898,    0,   16,    8,    2,   97],
       [   10,    5,    1,   62,    3, 3628,   50,    5,    6,   25],
       [   21,    7,    1,    0,    4,   18, 4085,    0,    1,    0],
       [    2,   59,   11,    3,   11,    0,    0, 4267,    0,   48],
       [   14,   47,   14,   64,   14,   63,   20,   13, 3764,   50],
       [   14,    9,    3,   31,   42,   11,    2,   66,   13, 3997]])
```

### 9.1 Which digits are tricky to classify? ¶

As we can see from the confusion matrix, number 8 is the most tricky since it has the highest count of mapping to wrong number (~270).

(j) Train your classifier with all of the training data, and test your classifier with the test data. Submit your results to Kaggle.

## 10 j) Train

```
: 1 test_labeled_images = pd.read_csv('./data/test.csv')
  2 test_images = test_labeled_images.iloc[0:,1:]
  3 test_labels = test_labeled_images.iloc[0:,:1]

: 1 test_labeled_images

...

: 1 arr = []
  2 for i,img in test_labeled_images.iterrows():
  3     y = kNN(img.as_matrix(), images.as_matrix(), labels.as_matrix(), 10)
  4     arr.append([i,y])

: 1 df_result = pd.DataFrame.from_records(arr, columns=["ImageId","Label"])

: 1 df_result.to_csv("kaggle_digit_reg.csv", index=False)
```

### Your most recent submission

Name	Submitted	Wait time	Execution time	Score
kaggle_digit_reg.csv	a day ago	27 seconds	1 seconds	0.96657

Complete

[Jump to your position on the leaderboard](#) ▼

1144	▼ 28	Grant Fuhr		0.96657	1	2mo
1145	▼ 28	Saurabh Kelkar		0.96657	8	16d
1146	▼ 28	lilife		0.96657	1	3d
1147	▼ 28	Rom Cohen		0.96657	1	2d
1148	▼ 27	Yunie Mao		0.96657	1	2d
1149	new	AnLe		0.96657	1	1d

### Your Best Entry ↑

Your submission scored 0.96657, which is not an improvement of your best score. Keep trying!

1150	new	David Hachuel		0.96657	1	1h
------	-----	---------------	---	---------	---	----



## Question 2 - Titanic

(a) Join the Titanic: Machine Learning From Disaster competition on Kaggle. Download the training and test data.

### ANSWER:

After downloading the data, we did a brief check of the content of it. We checked the features from each column and also check whether there is any missing data.

### <CODE>

#### Load Data

```
In [300]: train_data = pd.read_csv('./data/titanic/train.csv')
test_data = pd.read_csv('./data/titanic/test.csv')
```

#### Check Data Content

```
In [301]: train_data.head()
```

Out[301]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
In [302]: train_data.isnull().sum()
```

```
Out[302]: PassengerId      0
Survived      0
Pclass        0
Name          0
Sex           0
Age          177
SibSp         0
Parch         0
Ticket        0
Fare          0
Cabin       687
Embarked      2
dtype: int64
```

### <RESULT>

We found that there are many missing data in 'Age', 'Cabin' and 'Embarked' attributes.

(b) Using logistic regression, try to predict whether a passenger survived the disaster. You can choose the features (or combinations of features) you would like to use or ignore, provided you justify your reasoning.

## ANSWER:

Before applying logistic regression, we made some pre-study to the given training data and tried to understand the relationship between survival and each feature.

### 1. Pre-Study: Feature Distributions by Survival

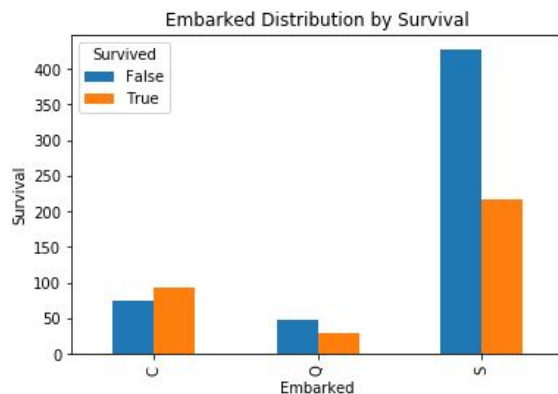
#### <CODE>

##### Embarked Distribution

```
pd.crosstab(train_labeled_data.Embarked, train_labeled_data.Survived.astype(bool)).plot(kind='bar')
plt.title('Embarked Distribution by Survival')
plt.xlabel('Embarked')
plt.ylabel('Survival')

print ("Number of people in 'C':" + str(len(train_labeled_data[train_labeled_data.Embarked=='C'])))
print ("Number of people in 'Q':" + str(len(train_labeled_data[train_labeled_data.Embarked=='Q'])))
print ("Number of people in 'S':" + str(len(train_labeled_data[train_labeled_data.Embarked=='S'])))
```

```
Number of people in 'C':168
Number of people in 'Q':77
Number of people in 'S':644
```



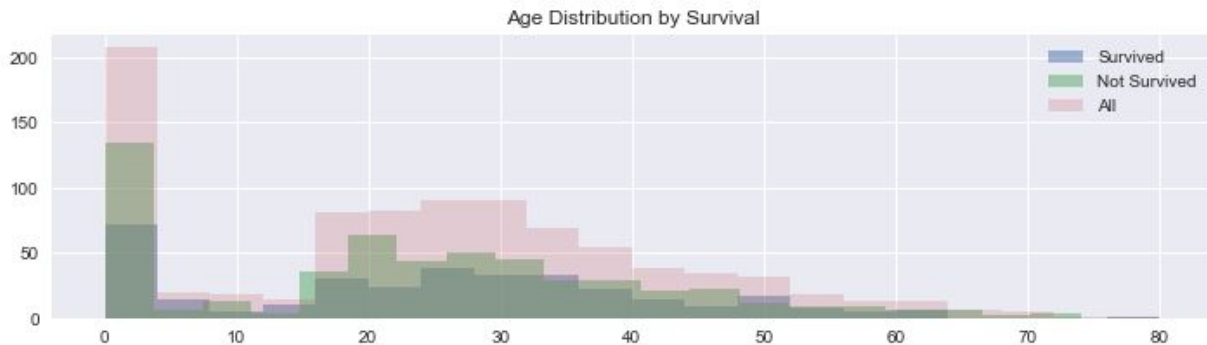
#### <RESULT>

At very beginning, we thought the “Embarked Position” is an obvious feature for us to do classification since many people who embarked at ‘Southampton’ did survive. However, we then realized that this result may due to the fact that there are more people actually embarked at ‘Southampton’ than the other two location. (644 vs 77 and 168). But when we had a closer look, we found the survival rate in ‘S’ group is actually much higher than the other two groups. In this case, we still believe it can be a good feature for us to train a classifier.

## <CODE>

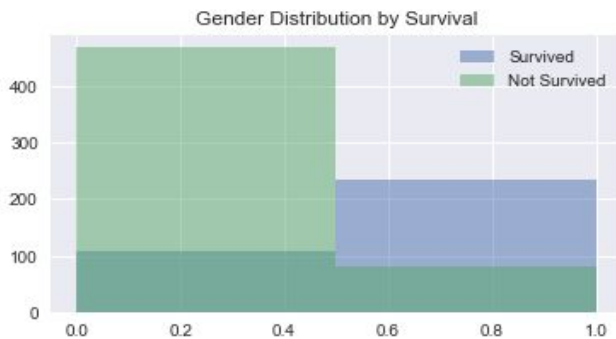
### Age Distribution

```
plt.figure(figsize=(12, 3))
plt.title("Age Distribution by Survival")
plt.hist(survived.Age, 20, alpha=0.5, label='Survived')
plt.hist(not_survived.Age, 20, alpha=0.5, label='Not Survived')
plt.hist(train_labeled_data.Age, 20, alpha=0.2, label='All')
plt.legend(loc='upper right')
plt.show()
```



### Gender Distribution

```
plt.figure(figsize=(6, 3))
plt.title("Gender Distribution by Survival")
plt.hist(survived.Gender, 2, alpha=0.5, label='Survived')
plt.hist(not_survived.Gender, 2, alpha=0.5, label='Not Survived')
plt.legend(loc='upper right')
plt.show()
```



## <RESULT>

According to our observation, the 'Gender' can be a determinant feature for deciding how likely a person survives. For the 'Age' metric, we find the survival rate for teenagers are much more higher than infants and seniors. So we decided to take both features to train the classifier.

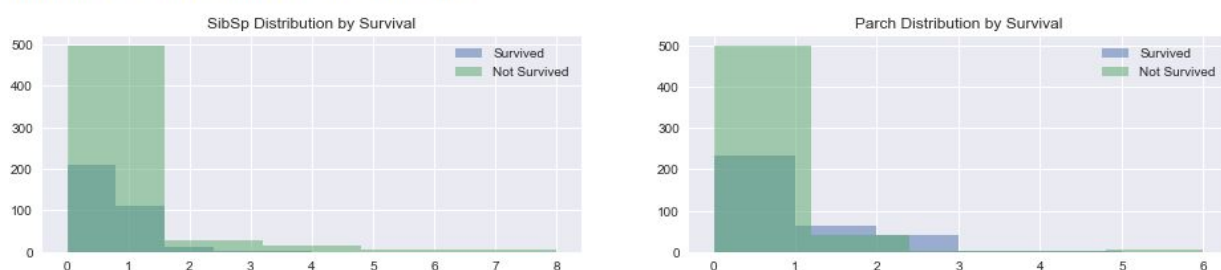
## <CODE>

### SibSp / Parch / Family Size Distribution

```
plt.figure(figsize=(16, 3))
plt.subplot(1, 2, 1)
plt.title("SibSp Distribution by Survival")
plt.hist(survived.SibSp, 5, alpha=0.5, label='Survived')
plt.hist(not_survived.SibSp, 5, alpha=0.5, label='Not Survived')
plt.legend(loc='upper right')

plt.subplot(1, 2, 2)
plt.title("Parch Distribution by Survival")
plt.hist(survived.Parch, 5, alpha=0.5, label='Survived')
plt.hist(not_survived.Parch, 5, alpha=0.5, label='Not Survived')
plt.legend(loc='upper right')
```

<matplotlib.legend.Legend at 0x11d39dac8>



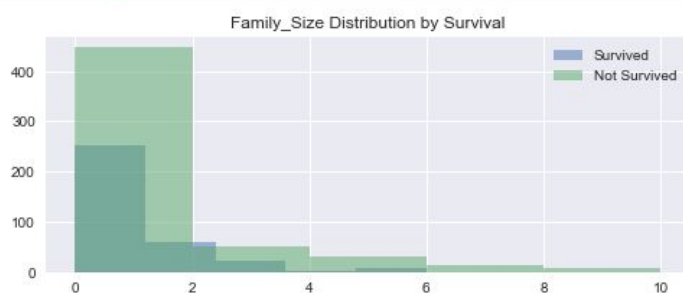
## <RESULT>

We found that the person with less family members are more likely to survive. After looking through the raw data, we guessed the reason behind it is that the big family tends to be the lower incoming people and many of them buy the 3rd class ticket.

We also noticed that the distribution of SibSp and Parch features are highly related and they actually describe the same metric of passengers. So we decided to combine these two features into one named family\_size by simply adding them together.

```
train_labeled_data['Family_Size'] = train_labeled_data['SibSp'] + train_labeled_data['Parch']

plt.figure(figsize=(8, 3))
plt.title("Family_Size Distribution by Survival")
plt.hist(survived.Family_Size, 5, alpha=0.5, label='Survived')
plt.hist(not_survived.Family_Size, 5, alpha=0.5, label='Not Survived')
plt.legend(loc='upper right')
plt.show()
```

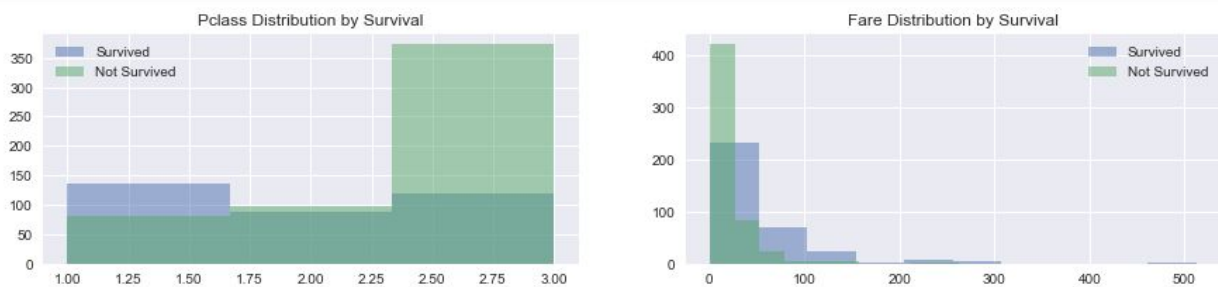


## <CODE>

### Class / Fare Distribution

```
plt.figure(figsize=(15, 3))
plt.subplot(1, 2, 1)
plt.title("Pclass Distribution by Survival")
plt.hist(survived.Pclass, 3, alpha=0.5, label='Survived')
plt.hist(not_survived.Pclass, 3, alpha=0.5, label='Not Survived')
plt.legend(loc='upper left')

plt.subplot(1, 2, 2)
plt.title("Fare Distribution by Survival")
plt.hist(survived.Fare, 10, alpha=0.5, label='Survived')
plt.hist(not_survived.Fare, 10, alpha=0.5, label='Not Survived')
plt.legend(loc='upper right')
plt.show()
```



## <RESULT>

According to our observation, the 'Class' and 'Fare' are two very related features and both of them can help us to make a decision. The group of person who belongs to the Higher level class or paid more fare fee are more likely to survive. That is make sense since there will be more escape equipments in these rooms.

## 2. Ignore some features

We found the features like 'PassengerId', 'Name', 'Ticket', 'Cabin' are either not related to the survival result or have too many missing data, so we decided to just ignore those features.

## 3. Estimate the missing data (Age)?

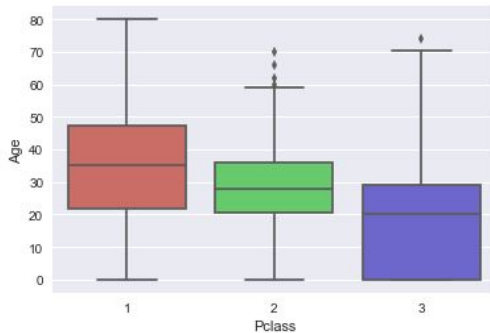
As we noticed that there are 177 missing data for age, but we believe this feature can be very useful for training. In this case, we must find a method to estimate the missing data.

We believe the age is very likely to be reflected by the class a passenger belongs to. It make sense to assume that a younger passenger may be more likely to buy a cheap ticket while the elders may can afford a first class ticket.



```
sb.boxplot(x='Pclass', y='Age', data=train_data, palette='hls')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11e3f3710>
```



```
def age_approx(cols):
    Age = cols[0]
    Pclass = cols[1]

    if pd.isnull(Age):
        if Pclass == 1: return 37
        elif Pclass == 2: return 29
        else: return 20
    else:
        return Age
```

Based on this assumption, we plotted the diagram above to see the age distribution within each class. It helped us to create the following function to estimate the missing age data based on the class a passenger belongs to. The average age for class 1 passengers is around 37, for class 2 is around 29 and 20 for class 3 passengers.

## 4. Summary of data processing

### <CODE>

```
def process_data(input_data):
    processed_data = input_data

    # Drop useless fields
    processed_data = processed_data.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], 1)

    # Convert 'Sex'(str) to 'Gender'(int), 0 for male and 1 for female
    processed_data['Gender'] = (processed_data.Sex == "female").astype(int)
    processed_data = processed_data.drop(['Sex'], 1)

    # Estimate missing 'Age' value by 'Pclass' value
    processed_data['Age'] = processed_data[['Age', 'Pclass', 'SibSp', 'Parch']].apply(age_approx, axis=1)

    # Convert 'Embarked'(str) to 'embark_location'(int)
    processed_data['embark_location'] = processed_data[['Embarked']].apply(embarked_convert, axis=1)
    processed_data = processed_data.drop(['Embarked'], 1)

    # Create new field 'Family_Size'
    processed_data['Family_Size'] = processed_data['SibSp'] + processed_data['Parch']
    processed_data = processed_data.drop(['SibSp', 'Parch'], 1)

    processed_data['Fare'] = (processed_data.Fare).fillna(0)
    return processed_data
```

```
processed_train_data = process_data(train_data)
processed_train_data.isnull().sum()
processed_train_data.head()
```

	Survived	Pclass	Age	Fare	Gender	embark_location	Family_Size
0	0	3	22.0	7.2500	0	2	1
1	1	1	38.0	71.2833	1	0	1
2	1	3	26.0	7.9250	1	2	0
3	1	1	35.0	53.1000	1	2	1
4	0	3	35.0	8.0500	0	2	0



(c) Train your classifier using all of the training data, and test it using the testing data. Submit your results to Kaggle

**ANSWER:**

We applied many method to test the real performance of our classifier such as cross validation, randomly split testing and training data and take the average accuracy from 20 times running.

Eventually our prediction submitted to Kaggle got **76.55%** accuracy.

4 submissions for Xialin SHEN		Sort by	Most recent
All	Successful	Selected	
Submission and Description		Public Score	Use for Final Score
titanic_prediction_1505175243119.csv 2 days ago by Xialin SHEN <a href="#">add submission details</a>		0.76555	<input checked="" type="checkbox"/>

## ***PART B - WRITTEN EXERCISE***

### **Question 1**

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

**ANSWER:**

$$\begin{aligned}\text{var}(X - Y) &= E(X - Y)^2 - (\mu_x - \mu_y)^2 \\ &= E(X^2 - 2XY + Y^2) - (\mu_x^2 - 2\mu_x\mu_y + \mu_y^2) \\ &= EX^2 - 2E(XY) + EY^2 - \mu_x^2 + 2\mu_x\mu_y - \mu_y^2 \\ &= (EX^2 - \mu_x^2) + (EY^2 - \mu_y^2) - 2(E(XY) - \mu_x\mu_y)\end{aligned}$$

$$\begin{aligned}\text{We have :} \quad \text{var}(X) &= E(X^2) - \mu^2 \\ \text{cov}(XY) &= E(XY) - \mu_x\mu_y\end{aligned}$$

Therefore:

$$\begin{aligned}(EX^2 - \mu_x^2) + (EY^2 - \mu_y^2) - 2(E(XY) - \mu_x\mu_y) &= \text{var}(X) + \text{var}(Y) - 2\text{cov}(X, Y) \\ \Rightarrow \text{var}(X - Y) &= \text{var}(X) + \text{var}(Y) - 2\text{cov}(X, Y)\end{aligned}$$

## Question 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?**

**ANSWER:**

Let's denote  $X_1$  for "the widget is defective", and  $X_2$  for "the widget is not defective"

Let's denote  $Y_1$  for "the widget is tested as defective", and  $Y_2$  for "the widget is tested as not defective"

Then we have:  $Pr(Y_1|X_1) = 0.95$ ,  $Pr(Y_2|X_2) = 0.95$ ,  $Pr(X_1) = \frac{1}{100000}$ ,  $Pr(X_2) = \frac{99999}{100000}$

$$\begin{aligned} Pr(X_1|Y_1) &= \frac{Pr(Y_1|X_1) * Pr(X_1)}{Pr(Y_1)} = \frac{Pr(Y_1|X_1) * Pr(X_1)}{Pr(Y_1|X_1) * Pr(X_1) + Pr(Y_1|X_2) * Pr(X_2)} = \frac{0.95 * 1/100000}{0.95 * 1/100000 + (1-0.95) * (1-1/100000)} \\ &= 0.00019 \end{aligned}$$


So the chance that the widget is actually defective given the test result is 0.00019

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

**ANSWER:**

If the widgets are tested as defective but they are actually not, the possibility of this event is:

$$\begin{aligned} Pr(Y_1, X_2) &= Pr(X_2) * Pr(Y_1|X_2) = Pr(X_2) * (1 - Pr(Y_2|X_2)) = (99999/100000) * (1 - 0.95) \\ &= 0.0499995 \end{aligned}$$



The number of good widgets are thrown away per year is  $10,000,000 * 0.0499995 = 499995$

If the widgets are tested as not defective but they are actually defective, the possibility of this event is:

$$\begin{aligned} Pr(Y_2, X_1) &= Pr(X_1) * Pr(Y_2 | X_1) = Pr(X_1) * (1 - Pr(Y_1 | X_1)) = (1/100000) * (1 - 0.95) \\ &= 0.0000005 \end{aligned}$$

The number of bad widgets still shipped to customers per year is  $10,000,000 * 0.0000005 = 5$

## Question 3

In k-nearest neighbors, the classification is achieved by majority vote in the vicinity of data. Suppose our training data comprises  $n$  data points with two classes, each comprising exactly half of the training data, with some overlap between the two classes.

**(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.)**

**ANSWER:**

When we set  $K$  equals to 1, the KNN classifier will only consider the training data who has the shortest distance to the input testing data, which may lead to the overfitting problem. In this case, noise will have a dominate influence on the classification result in some specific cases.

While a larger  $K$  value can make the computation more complex and time-consuming. More importantly, if  $K$  is too big, let say set  $K$  equals to the number of training data, then the KNN classifier will always classify any input data to the label which appears the most frequently in the training set. All the information provided by training data and input data are simply ignored.

So in both edge cases, whether we set  $K$  to a too small or a too big number, the prediction error will be high. When  $K$  gradually varies from  $n$  to 1, the prediction error may becomes better and reach a peak point and then falling down again.

*However*, by considering this question's scenario. We find that the KNN classifier will be trained and then tested with all the training data (if I understand the sentence in the brackets correctly). In this case, the prediction error will still be bad when  $K$  is too big, but it will become better when  $K$ 's value become smaller, and eventually reach 0% when  $K$  equals to 1. Since the classifier can always find the exact same data in the training set, it will never misclassify when  $K = 1$ .

At the same time, since there is only two possible labels in the training set, the prediction error will be 0.5 if  $K$  equals to  $n$ .

So we can describe the prediction error when  $K$  varies from  $n$  to 1 like this:

***The prediction error will be 0.5 when  $K$  equals to  $n$ . As  $K$  varies from  $n$  to 1, the error rate will gradually become lower and eventually it will reach zero when  $K$  equals to 1.***

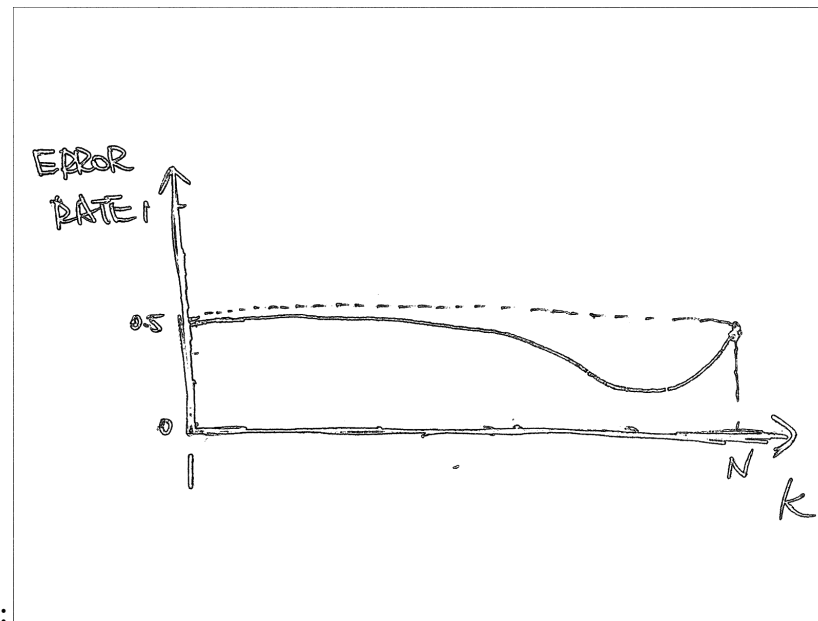
**(b) We randomly choose half of the data to be removed from the training data, train on the remaining half, and test on the held-out half. Predict and explain with a sketch how the average 0-1 prediction error on the held-out validation set might change when  $k$  varies? Explain your reasoning.**

**ANSWER:**

When we randomly remove half of the data, it will not affect the prediction error if  $K$  equals to  $n$ . Since the prediction result will still equal to the random guess about the result, it can be either correct or incorrect, thus the error rate is still 0.5.


On the other hand, let's consider the case when  $K$  equals to 1. When the KNN classifier received an input data, the possibility that the same data in the training set was removed is 0.5. In this case, the average error rate will be slightly smaller than 0.5. Since if the input data is a training data which is far away from the decision boundary, it still can be correctly classified by matching to the first nearest neighbour.

The curve of average error rate can be like that



The error rate will be 0.5 when  $K$  equals to 0.5, and will reach the lowest value at a certain point when  $K$  moves to 1. As  $K$  becomes closer to 1, the error rate will go up and eventually reach a value which is slightly lower than 0.5





**(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.**

More folds number for cross-validation can help us get a better measurement of the true accuracy. However, it will also make computation more complex and time-consuming. Taking the computational requirements and the validation accuracy into consideration, we should 5-10 as the number of fold.

**(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.**

We can apply a weighting function for voting. The neighbors with smaller Euclidean Distance value will get a higher weight when we vote to decide the prediction.

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

1. Computation will be too time consuming
2. In a very high dimensional space the data are gathered more closer to each other, in this case, the distance to all neighbors becomes more or less the same, and it becomes meaningless to choose the nearest neighbors.

---

## ***PART C - APPENDIX***

### **CODE**

[An Le](#)

[Xialin Shen](#)

### **REFERENCES**

Titanic:

<http://www.data-mania.com/blog/logistic-regression-example-in-python/>

<https://www.kaggle.com/startupsci/titanic-data-science-solutions>

kNN:

Machine Learning in Action, P.Harrington

**- END -**