“CS5785 Homework 1”

Submitted as part of Homework
required for
Applied Machine Learning (CS5785)

SUBMITTED BY

TANUJ AHUJA – ta364
JAMIE YU – jky32

SUBMITTED TO

Cornell Tech, Cornell University

2 W Loop Road,
New York, NY 10044

PROGRAMMING EXCERCISES

Q1. Digit Recognizer

Q1(a). Join the Digit Recognizer competition on Kaggle. Download the training and test data. The competition page describes how these files are formatted.

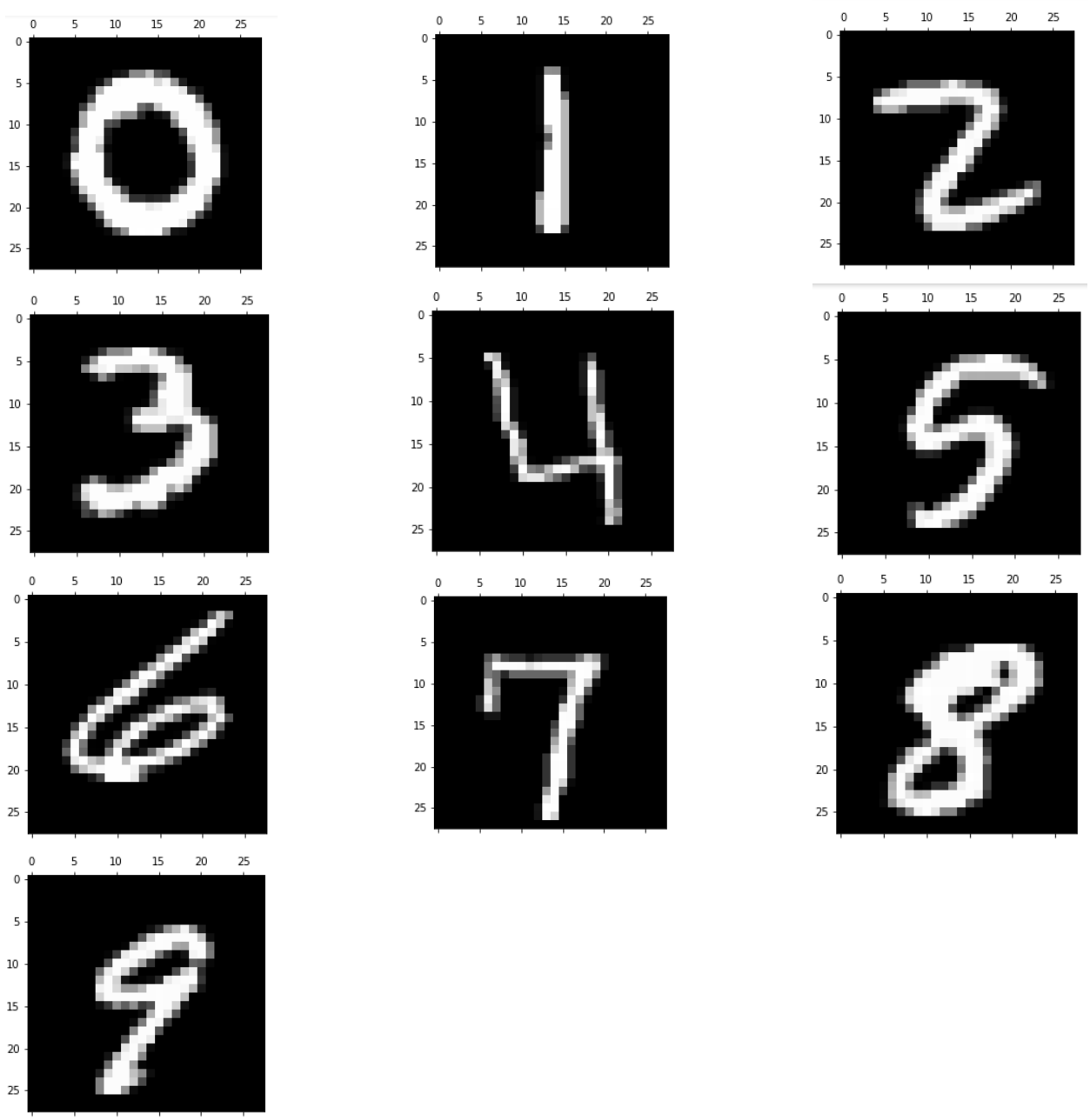
Ans1(a). Kaggle IDS

Q1(b). Write a function to display an MNIST digit. Display one of each digit.

Ans1(b). **CODE**

```
%pylab inline
import pandas as pd
import numpy
file = pd.read_csv("MNIST/train.csv")

labels = file["label"]
pixels = file.iloc[:,1:]

pixels = pixels.as_matrix()
def displayNumber(pixels):
matshow(pixels.reshape(28,28), cmap='gray')

displayNumber(pixels[1])
displayNumber(pixels[2])
displayNumber(pixels[24])
displayNumber(pixels[9])
displayNumber(pixels[3])
displayNumber(pixels[8])
displayNumber(pixels[21])
displayNumber(pixels[6])
displayNumber(pixels[10])
displayNumber(pixels[11])
displayNumber(pixels[7])
```

OUTPUT



Q1(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?

Ans1(c). **CODE**

```
digits=[0]*10
for i in labels:
        digits[i]=digits[i]+1

prior_prob=[]
for i in digits:
        prior_prob.append(i/len(labels))
print(prior_prob)
```

OUTPUT

[0.09838095238095237, 0.11152380952380953, 0.09945238095238096, 0.1035952380952381, 0.09695238095238096, 0.09035714285714286, 0.0985, 0.10478571428571429,

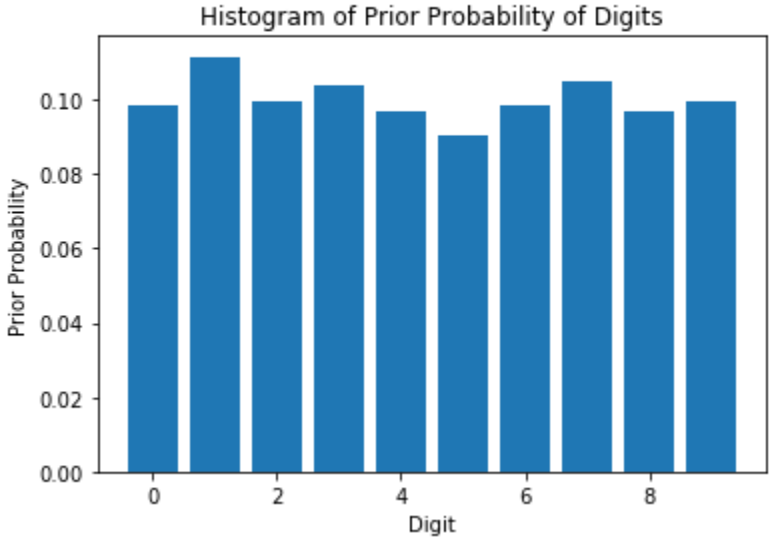
0.09673809523809523, 0.09971428571428571]

CODE

```
plt.title("Histogram of Prior Probability of Digits")
x_axis_digit=[0,1,2,3,4,5,6,7,8,9]
plt.bar(x_axis_digit,prior_prob)
plt.xlabel("Digit")
plt.ylabel("Prior Probability")
```

OUTPUT

```
<matplotlib.text.Text at 0x2ab522828>
```



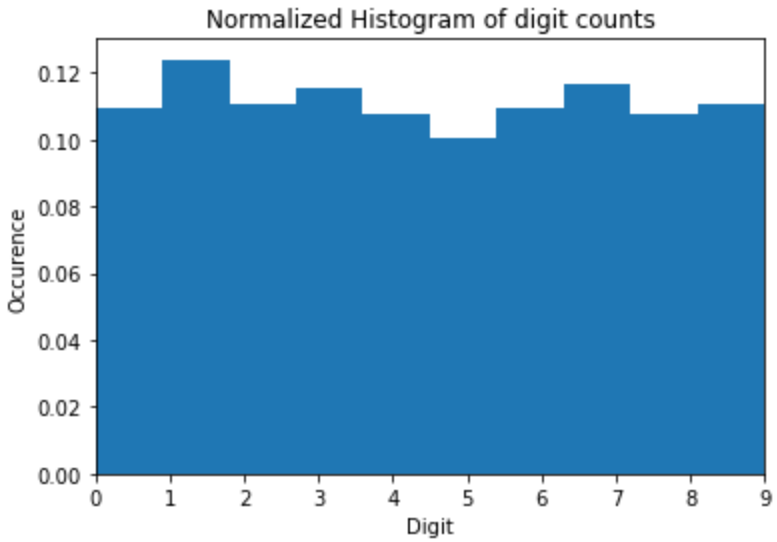
The graph clearly indicates that the Prior Probability is not uniform across the digits.

CODE

```
plt.title("Normalized Histogram of digit counts")
plt.hist(labels,normed = 1)
plt.xlim(0,9)
plt.xlabel("Digit")
plt.ylabel("Occurence")
```

OUTPUT

```
<matplotlib.text.Text at 0x2ab84b0f0>
```



The histogram of Digit Count is uneven indicating a non-uniform distribution in initial dataset.

Q1(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 *L*2 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).

Ans1(d). **CODE**

```
def l2d(a,b):
distances = (a-b)**2
distances = distances.sum()
distances = numpy.sqrt(distances)
return distances

series=[7,1,2,24,9,3,8,21,6,10,11]
for j in series:
        distance =l2d(pixels[j],pixels[0])
        index =0
        for i,value in enumerate(pixels):
                temp_distance=l2d(pixels[j],pixels[i])
                if i!=j and temp_distance<distance:
                        distance= temp_distance
                         index =i
        if labels[j] != labels[index] :
                print ("Actual",labels[j],"Nearest",labels[index],"*")
         else:
                print ("Actual",labels[j],"Nearest",labels[index])
```

OUTPUT

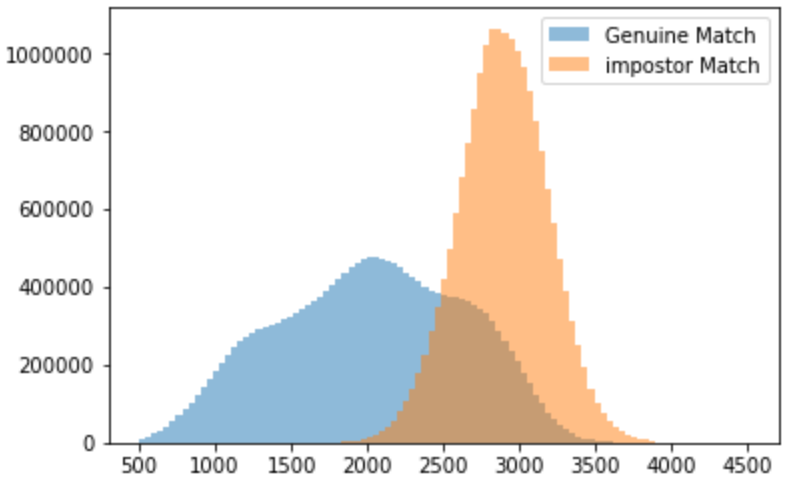
```
Actual 3 Nearest 5 *
Actual 0 Nearest 0
Actual 1 Nearest 1
Actual 2 Nearest 2
Actual 3 Nearest 3
Actual 4 Nearest 4
Actual 5 Nearest 5
Actual 6 Nearest 6
Actual 7 Nearest 7
Actual 8 Nearest 8
Actual 9 Nearest 9
```

Q1(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.

Ans1(e). **CODE**

```
def binaryComparison(data):
    #load in occurences of 0 and 1
    zero = data[data[:,0]==0]
    one = data[data[:,0]==1]

    #calculate pairwise distance for genuine matches(0 with 0, 1
    #with 1)
    genuine = np.concatenate((distance.pdist(one, 'euclidean'),
    distance.pdist(zero, 'euclidean')))

    #cdist calculates pairwise distance between the digits 0 and 1
    imposter = distance.cdist(zero, one, 'euclidean').flatten()

    bins = np.linspace(500, 4500, 100)
    plt.hist(genuine, bins, alpha= 0.5, label='Genuine Match')
    plt.hist(imposter, bins, alpha= 0.5, label='Imposter Match')
    plt.savefig('binaryComparison.png')

    return genuine, imposter;
```

OUTPUT



Q1(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?

Ans1(f). **CODE**

```
def tpr_fpr(threshold):
        tp=0
        fn=0
        for i in gen:
                if i < threshold:
                        tp=tp+1
                else:
                        fn=fn+1

        fp=0
        tn=0
        for i in impostor:
                if i < threshold:
                        fp=fp+1
                else:
                        tn=tn+1

        tpr=(tp)/(tp+fn)
        fpr=(fp)/(fp+tn)
        return tpr,fpr
```

Q1(g). Implement a K-NN classifier. (You cannot use external libraries for this question; it should be your own implementation.)

Ans1(g). **CODE**

```
#Source: https://www.youtube.com/watch?v=NIFEwPNyBzk
#helper function to find majority class amongst k items
def findMajorityClass(index, pixels, labels, k, sortedDistIndices):
    classCount = {}
    # iterate k times from the closest item
    for i in range(k):
        voteIlabel = labels[sortedDistIndices[i]]
        classCount[voteIlabel] = classCount.get(voteIlabel, 0)+1

    #sort the items in the classCount
    return sorted(classCount.items(), key=operator.itemgetter(1),
    reverse=True)

#main function
def KNN(index, testImages, labels, k):
    # calculate the distance between index and the current point
    imagesSize = testImages.shape[0] #returns the number of rows
    #calculate distance from index to each tested data point
    diff = np.tile(index, (imagesSize, 1)) - testImages
    sqDiff = diff ** 2
    sqDistances = sqDiff.sum(axis=1)
    distances = sqDistances ** 0.5
    sortedDistIndices = distances.argsort()

    # take k items with lowest distances to index and find the
    #majority class
    sortedClassCount = findMajorityClass(index, pixels, labels, k,
    sortedDistIndices)

    #return the first item in sortedClassCount to obtain the class
    #that got the most votes
    return sortedClassCount[0][0]
```

Q1(h). Using the training data for all digits, perform 3 fold cross-validation on your K-NN classifier and report your average accuracy.

Ans1(h). **CODE**

```
from sklearn import cross_validation
allFolds = cross_validation.KFold(len(pixels), n_folds=3)
for training, testing in allFolds:
        print("%s %s" % (training, testing))
```

Q1(i). Generate a confusion matrix (of size 10×10) from your results. Which digits are particularly tricky to classify?

Ans1(i). **CODE**

```
from sklearn.metrics import confusion_matrix
y_pred = []
for i in allFolds:
   test, train = i
   pixel_train, pixel_test = pixel[train], pixel[test]
   label_train, label_test = label[train], label[test]
   result=KNNClassifier(pixel_test,pixel_train,3,train,test)
   print(len(result))
   for i in range(0,len(result)):
      y_pred.append(result[i])
   return numpy.mean((numpy.asarray(result)==y_test))
for i in all_folds:
   print(calculate_accuracy(i))

confusion_matrix(Y_true,y_pred)
```

WHAT WE DID

This assignment makes use of python library **Pandas** and **Numpy** to load in the training*(‘train.csv’*) and testing data*(‘test.csv’*)**.** The *‘read_csv’* function of pandas is used to read and store the ‘N x p’(42,000 x 784) data into two ‘**Dataframe’**s named *‘labels* and ‘‘*pixels***.** The labels are the first column of the data set(column 0); each individual label corresponds to a set of 784 datapoints in the rest of the row. These pixels are reshaped as a 28 x 28 matrix and plotted using the **matplotlib.pyplot** library.

The prior probability was calculated by counting the number of occurrences of each label divided by the total number of labels, thus calculating the probability of each digit. This was then plotted in histogram form using the **matplotlib.pyplot** library.

One example of each digit was picked from the training data by manually selecting the indices of unique labels and putting them into an array. Then, for each sample digit, the $L2$ distance was measured between that sample digit and the rest of the individual

images in the training data. Using the distance between the two images' pixel values as a metric, the best match(or nearest neighbor) was calculated. We had mostly accurate results, with the exception of one '3' getting mistakenly matched to a '5'.

To conduct a binary comparison between the digits 0 and 1, the training data was loaded in as a numpy array, for ease of data manipulation. The pairwise distance was first calculated between the genuine matches (0 with 0 and 1 with 1) using the *'cdist'* function in the **scipy(.spatial.distance)** library. The pairwise distance between 0 and 0 and 1 and 1 were concatenated into one array using Numpys' *'concatenate'*. Then the pairwise distance between the digits 0 and 1, or the imposter matches, were calculated using the *'pdist'* function in the **scipy(.spatial.distance)** library. The pairwise distances from the genuine and imposter matches were plotted in a histogram using the **matplotlib.pyplot** library.

The KNN implementation was split into two separate functions or steps. In the first step, the distance between the query point(the datapoint being tested) and a sample of datapoints from the testing data were calculated using the Pythagorean theorem. In the second helper function, k items with lowest distances to the query point were used to find to find the majority class or the class that had the highest frequency in the k sample of datapoints from the testing data.

The 3-fold cross validation on our KNN implementation was calculated using '*cross_validation.KFold* ' from the **sklearn** library. The confusion matrix was calculated using '*confusion_matrix*' from **sklearn.metrics**.

INSIGHTS

- In 1d, the L2 distance (or the Euclidean distance) was used as a metric to calculate the nearest neighbor between two images. While this was mostly accurate, we saw that it mistook the digit '5' for a '3'. In 1e, the L2 distance was used again to plot the differences in distances between and genuine match and an imposter match. The imposter match, on average, had much higher distances than genuine matches; however there was significant overlap between the two datasets as evidenced by the overlapping area seen in the histogram. The statistical significance of the area points to the likelihood of error and the fact that there using L2 distance does not guarantee genuine matches. This points to the weakness in the KNN algorithm, as KNN relies on measuring distances to also calculate its values.

Q2. The Titanic Disaster

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

Ans) Kaggle ID - https://www.kaggle.com/tanuj123
https://www.kaggle.com/jamieyu226

(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.

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

Ans)

CODE

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

data = pd.read_csv("training.csv")
print(data.head(5))
print(data.describe())
```

OUTPUT

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch     Fare  Cabin Embarked
0      0   7.2500      0        S
1      0  71.2833      3        C
2      0   7.9250      0        S
3      0  53.1000      3        S
4      0   8.0500      0        S
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  891.000000  891.000000
mean    446.000000    0.383838    2.308642   29.382907    0.523008
std     257.353842    0.486592    0.836071   13.260272    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   22.000000    0.000000
50%     446.000000    0.000000    3.000000   27.000000    0.000000
75%     668.500000    1.000000    3.000000   36.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000
```

CODE

```
print(data.columns) #print the columns of our dataframe
for column in data.columns:
    print(data[column].dtype) #look at the data type of the column
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
int64
int64
int64
object
object
float64
int64
int64
float64
int64
object
```

print(pd.isnull(data))

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	\
0	False	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	
5	False	False	False	False	False	False	False	False	False	
6	False	False	False	False	False	False	False	False	False	
7	False	False	False	False	False	False	False	False	False	
8	False	False	False	False	False	False	False	False	False	
9	False	False	False	False	False	False	False	False	False	
10	False	False	False	False	False	False	False	False	False	
11	False	False	False	False	False	False	False	False	False	
12	False	False	False	False	False	False	False	False	False	
13	False	False	False	False	False	False	False	False	False	
14	False	False	False	False	False	False	False	False	False	
15	False	False	False	False	False	False	False	False	False	
16	False	False	False	False	False	False	False	False	False	
17	False	False	False	False	False	False	False	False	False	

CODE

```
for column in data.columns:
        if np.any(pd.isnull(data[column])) == True:
                print(column)

# convert female/male to numeric values (male=0, female=1)
data.loc[data["Sex"] == "male", "Sex"] = 0
data.loc[data["Sex"] == "female", "Sex"] = 1

data.loc[data["Embarked"] == "S", "Embarked"] = 0
data.loc[data["Embarked"] == "C", "Embarked"] = 1
data.loc[data["Embarked"] == "Q", "Embarked"] = 2

from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import cross_val_score
# columns we will use as features for our model
predictors = ["Pclass", "Sex", "Age", "SibSp", "Parch", "Cabin",
"Embarked"]
logreg = LogisticRegression()
print(cross_val_score(logreg, data[predictors], data['Survived'], cv=15,
```

scoring='accuracy').mean())

OUTPUT

```
0.814819793493
```

CODE

```
# reading the test data
test = pd.read_csv("test.csv")
test.loc[test["Sex"] == "male", "Sex"] = 0
test.loc[test["Sex"] == "female", "Sex"] = 1

test.loc[test["Embarked"] == "S", "Embarked"] = 0
test.loc[test["Embarked"] == "C", "Embarked"] = 1
test.loc[test["Embarked"] == "Q", "Embarked"] = 2

logreg.fit(data[predictors], data["Survived"])
prediction = logreg.predict(test[predictors])
submission = pd.DataFrame({
        "PassengerId" : test["PassengerId"],
        "Survived" : prediction
 })
print(submission)
```

OUTPUT

	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	1
5	897	0
6	898	1
7	899	0
8	900	1
9	901	0
10	902	0
11	903	0
12	904	1
13	905	0
14	906	1
15	907	1
16	908	0
17	909	0
18	910	1
19	911	0
20	912	0
21	913	0
22	914	1
23	915	1
24	916	1
25	917	0
26	918	1
27	919	0
28	920	0
29	921	0

```
submission.to_csv("submission.csv", index=False)
```

Ans2. <u>WHAT WE DID</u>

The most important step was to clean the data before feeding it to the logistic regression model. To achieve the cleaning part, we made use of *Python* and *Excel*. We essentially did the following data sanitization checks –

- Dropped the columns **Name**, **Ticket** and **Fare** since these made no relevance in deciding who should have lived and who drowned.
- We populated the missing **Ages** based upon based upon Median value derived from Ages of other members in the same *PClass* and same *Sex.*
- Columns *Sex*, *Cabin and Embarked* were converted to Boolean columns to make more sense of what output they generated.

INSIGHTS

- We received a score of **75.19%** on our submission on the Kaggle competition portal.
- Our co-variance score on training data is **81.5%**
- One interesting aspect of implementing Logistic Regression was that whenever we took fewer columns (for eg- Only Sex or Only Age) the accuracy went up suggesting that as more and more columns are made to undergo co-variance process, it becomes difficult for logistic regression to decipher the complex relationship between polynomial coefficients.

WRITTEN EXCERCISES

Q1. 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.

Ans1. Prove $var[X-Y] = var[X] + var[Y] - 2cov[X,Y]$:

Definition of variance is as such:

$$Var(W) = E(W - \mu)^2$$
$$= E(W^2 + \mu^2 - 2W\mu)$$
$$= E(W^2) - \mu^2$$

$$Definition\ of\ covariance\ is\ as\ such:$$
$$cov(W|Z) = E(WZ) - \mu_W\mu_Z$$

Substituting in 'X-Y' for 'W' in the equation for definition of variance:

$$Var(X-Y) = E\big((X-Y) - E(X-Y)\big)^2$$
$$= E((X-Y)^2 - (\mu_X - \mu_Y))^2$$
$$= E(X^2 + Y^2 - 2XY) - (\mu_X{}^2 + \mu_Y{}^2 - 2\mu_X\mu_Y)$$
$$= E(X^2) + E(Y^2) - 2E(XY) - \mu_X{}^2 - \mu_Y{}^2 + 2\mu_X\mu_Y$$

Using the definitions of variance and covariance we see that we can group the above equation into the variance of X, variance of Y, and the covariance(X|Y)

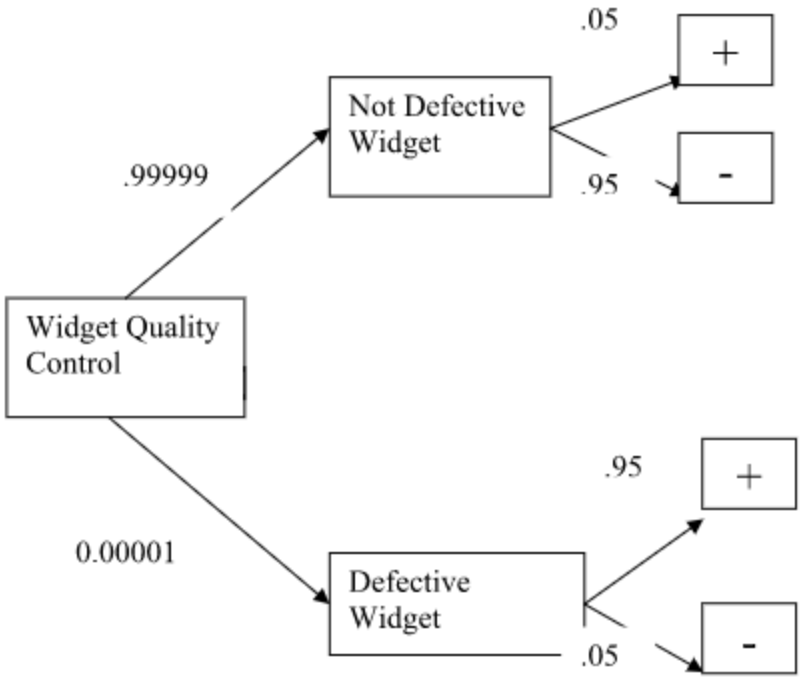
$$= (E(X^2) - \mu_X{}^2) + (E(Y^2) - \mu_Y{}^2) - 2(E|XY - \mu_X\mu_Y)$$
$$= \text{var(X)} + \text{var(Y)} - 2\text{cov(X|Y)}$$

Therefore it is proven that $var[X-Y] = var[X] + var[Y] - 2cov[X,Y]$

Q2. 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?

Ans)



To find: P(Defective Widget|Test shows that widget is defective)

Solution: Using Bayes Theorem,

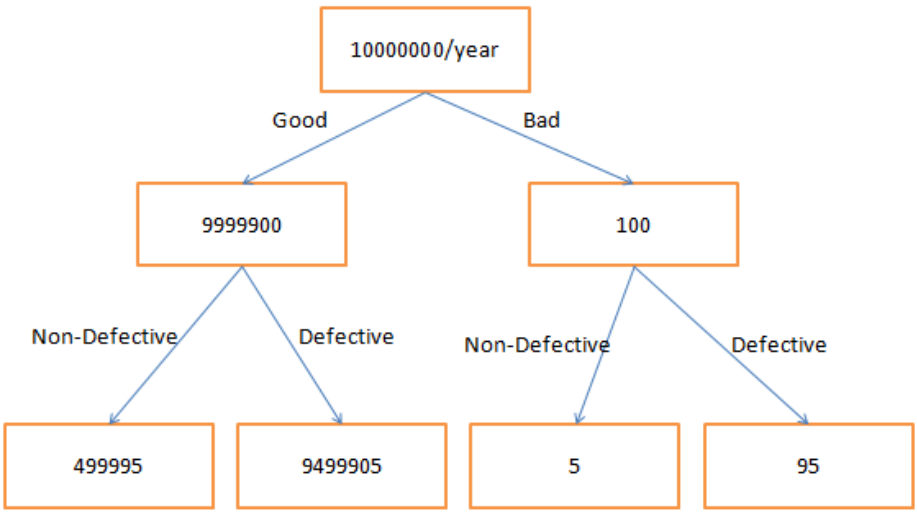
P(B|A) = (P(A|B)*P(B))/P(A)

= ((0.95)(0.00001))/((0.00001*0.95)+(0.99999*0.05))

= **0.0019**

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?

Ans)



Good Widgets thrown away/year = **9499905 pcs**
Bad Widgets shipped to customers/year = **5 pcs**

Q3. 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 *xi* includes (*xi* , *yi*) as part of the example training data used by *k*NN.)

Ans) Given two overlapping classes each comprising of n/2 elements, consider the following cases-

K = 1

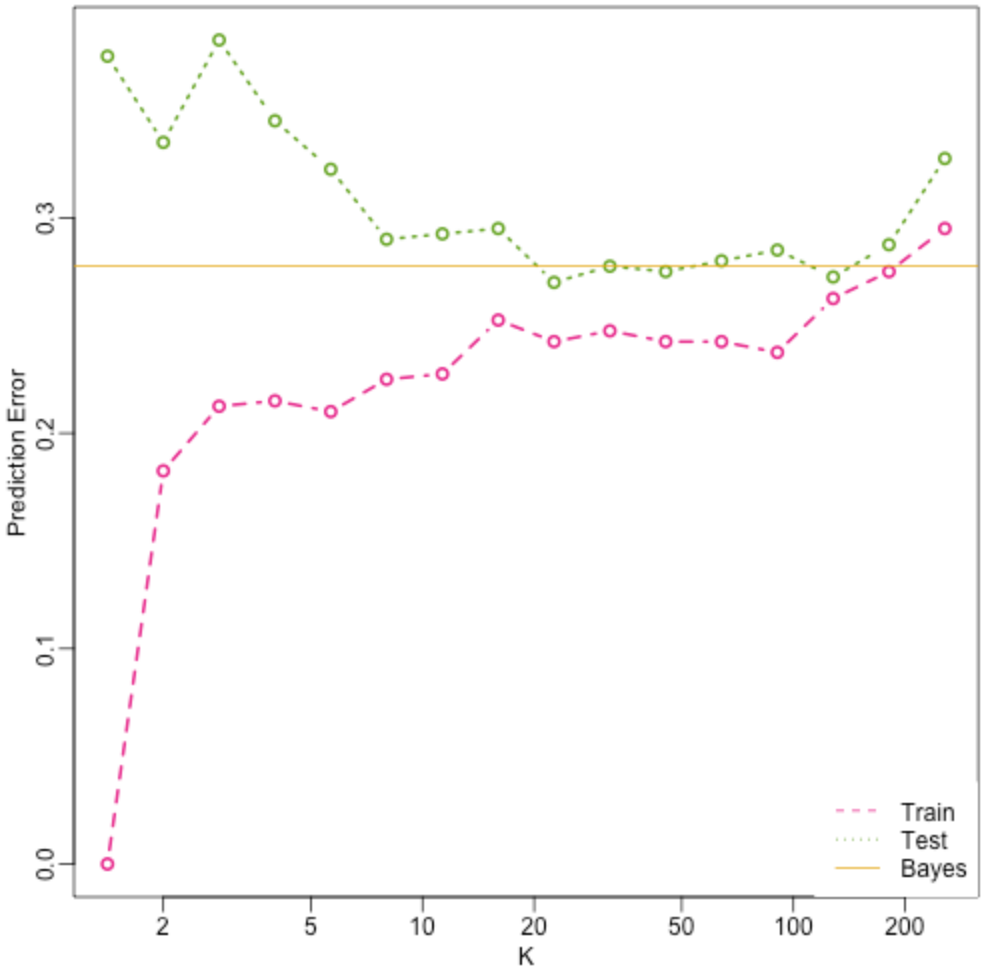
The classifier at k=1 comes out to be the most optimal choice since we are choosing our test points from the training set itself. The 1NN therefore remembers the correct label and has a **0 error rate**. In some cases, this can imply an Overfitting of data set but that is not true always.

K = n
In this case, the comparison at every step is made with ALL the **n** data points in the sample. Now given that there are two classes of n/2 each which have some overlapping involved, the error is large. In such cases, the classifier always predicts the Majority Class owing to oversimplified boundaries.

It can therefore be commented that as we tune the value of **k** from **n to 1,** we are **reducing 0-1 prediction error** owing to a reduction in noise due to overlapping classes (which result in simple decision boundaries in case of large values of **k**).

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) With a low value of k (typically less than 20), the training data is presumably overfitted. This overfitting allows for negligible 0-1 prediction errors for *Training Data* owing to the fact that **k** scans its very near Euclidean distances. On the hind side, since the training data gets too selective, it leads to a significant prediction error in *Test Data* .



Source: http://genomicsclass.github.io/book/pages/machine_learning.html

A larger value of k (typically >= 100) can cause the model to Underfit the data. The decision in such case is rolled to whatever class is present as Majority and therefore the *Training Data* suffers significant errors. Owing to the errors in *Training Data*, the *Testing Data* becomes almost as vulnerable to prediction.

Given the case of two classes each having **n/2** elements, the ideal prediction error is Probability of choosing a class(n/2) times probability of choosing the right neighbor (n/2) and therefore an ideal error is around 25%.

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 *k*NN and recommend an appropriate number.

Ans) Consider the following cases –

Case 1: k = N

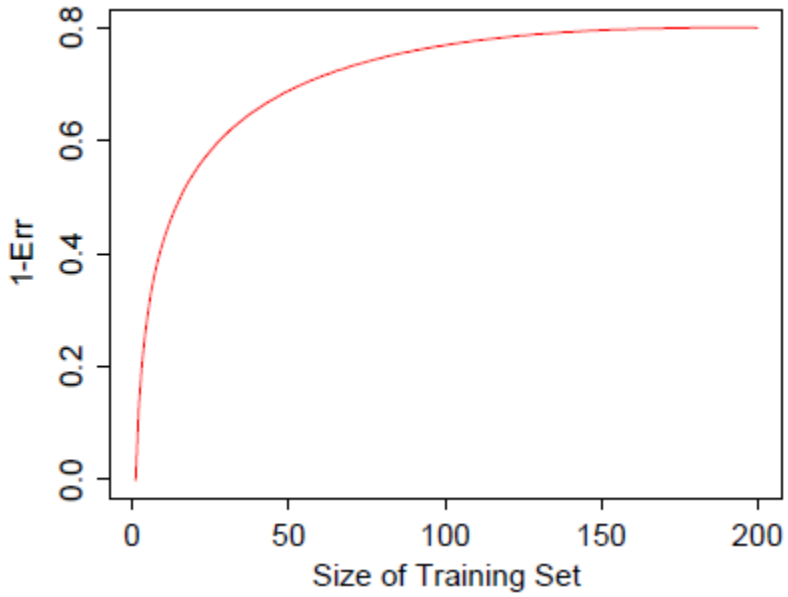
Validation Accuracy - Although it is approximately unbiased to the expected prediction error, given the fact that the **N** *Training Sets* are so similar to each other, the cross-validation estimator has a high variance.

Computational Requirements - The computational burden is also considerable, requiring N applications of the learning method.

Case 2 : k = 5

Validation Accuracy: In this case, cross-validation has lower variance but owing to a dependence of performance of learning methods on the size of the training set, bias could be a problem.

Figure below shows a hypothetical "learning curve" for a classifier on a given task, a plot of 1 − Err versus the size of the training set N. The performance of the classifier improves as the training set size increases to 100 observations; increasing the number further to 200 brings only a small benefit. If our training set had 200 observations, fivefold cross-validation would estimate the performance of our classifier over training sets of size 160, which from Figure 7.8 is virtually the same as the performance for training set size 200. Thus cross-validation would not suffer from much bias. However if the training set had 50 observations, fivefold cross-validation would estimate the performance of our classifier over training sets of size 40, and from the figure that would be an underestimate of 1 − Err. Hence as an estimate of Err, cross-validation would be biased upward. (*Source: HTF*)



Source : HTF

Therefore, **a five-fold or a ten-fold** seems like a good error estimation option which provides for an estimate trade-off between bias and variance spread.

d) In *k*NN, 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) A modification can we to use **Weighted k-NN algorithm** wherein a certain priority number can be associated with the neighbor points based on a fixed parameter(proximity, class label etc.)

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

Ans)

1) Since the Computational cost(Computation and Memory) of using a nearest neighbor classifier grows with the size of the training set, a dataset with large input dimension can be an expensive bet overall with Knn.
2) In very high dimensional space, the distance to all neighbors becomes more or less the same, and the notion of nearest and far neighbors becomes blurred thereby shunting the efficacy of the k-NN algorithm.