

# CS5785 Applied Machine Learning

## HW1- Write up

Jinquan Wang (Netid: jw2657)

Panda Xu (Netid: px48)

### Coding1: Write-up

In Programming Exercises one: Digit Recognizer, in this problem, we explored and analyzed the images of handwritten digits in the MNIST ("Modified National Institute of Standards and Technology") data set. In this problem, we not only familiarized the programming environment of Python and numerous related libraries available in python but also learned a lot of data classification techniques.

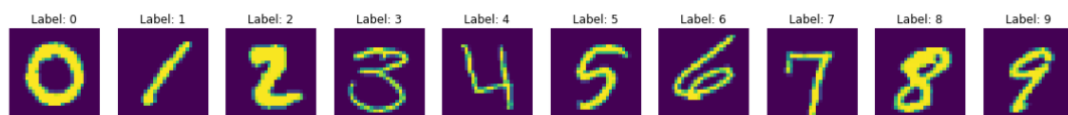
We first loaded the data using the `read_csv` in the Pandas library and then cleaned the data by separating the labels and values, normalizing the values, and reshaping the array for later analysis. Then we plotted a sample image for each digit in the dataset.

```
In [5]: plt.rcParams['figure.figsize'] = [20, 10]
_, axes = plt.subplots(1, 10)

one2ten_index = [[] for i in range(10)]

for index, digit in enumerate(Y_train):
    one2ten_index[digit].append(index)

for dig, ax in enumerate(axes):
    ax.set_axis_off()
    ax.imshow(X_train[one2ten_index[dig][0]][:, :, 0])
    ax.set_title('Label: %d' % dig)
```

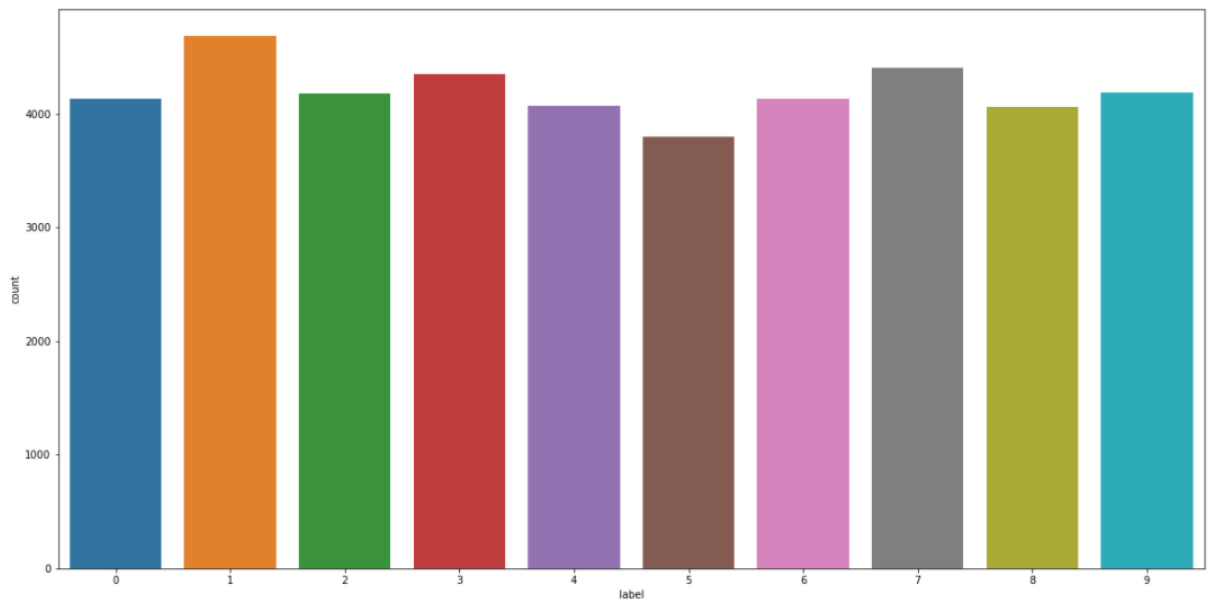


We then examined the probability across 10 digits and found out that even though they are not equally distributed but the number of images in each

digit class is similar. We plotted the distribution using the Countplot function in the *seaborn* library.

```
In [6]: g = sns.countplot(Y_train)
        Y_train.value_counts()
```

Name: label, dtype: int64

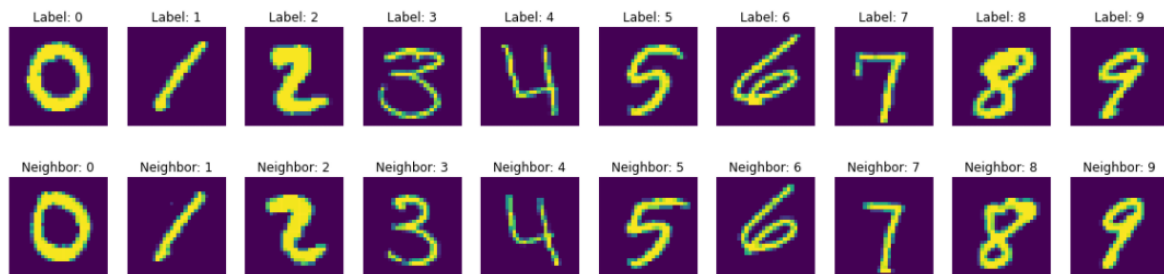


Next, we found one example of each digit from your training data and compared each of them with another of the same digit.

```
plt.rcParams['figure.figsize'] = [20, 10]
_, axes_sample = plt.subplots(1, 10)
_, axes_neighbor = plt.subplots(1, 10)

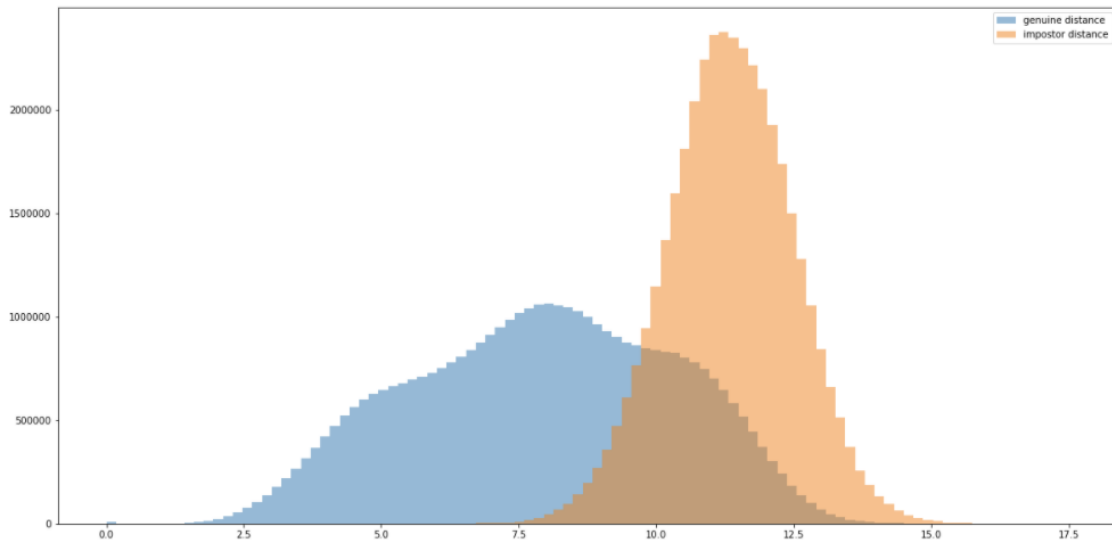
for dig, ax in enumerate(axes_sample):
    ax.set_axis_off()
    ax.imshow(X_train[one2ten_index[dig][0]][:,:,0])
    ax.set_title('Label: %d' % dig)

for dig, ax in enumerate(axes_neighbor):
    ax.set_axis_off()
    ax.imshow(X_train[minpos_list[dig]][:,:,0])
    ax.set_title('Neighbor: %d' % dig)
```

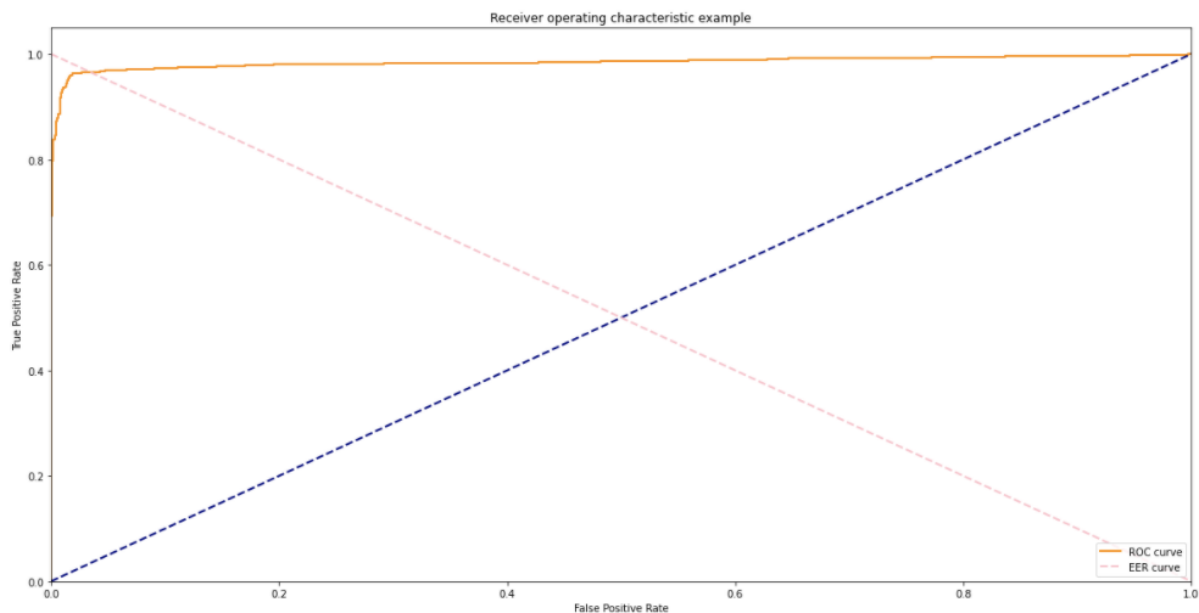


After that, we computed the L2 distances for all digit 0s and 1s and plotted a histogram graph of the genuine and impostor distances using the distances we obtained. We were having a hard time to figure out which variable should be the X-axis and Y-axis as well as the combination of genuine and impostor digits. We now know that all combinations of (0, 0) and (1, 1) should be genuine and all combinations of (0, 1) and (1, 0) should be impostors.

```
[10]: bins = np.linspace(0, 17.5, 100)
plt.gca().ticklabel_format(axis='both', style='plain', useOffset=False)
plt.hist(dist_gen, bins, alpha=0.5, label='genuine distance')
plt.hist(dist_imp, bins, alpha=0.5, label='impostor distance')
plt.legend(loc='upper right')
plt.show()
```



After finishing computing the distances, an ROC curve was generated using `roc_curve` from Sklearn library. We figured out the equal error rate is the value when  $1 - \text{True Positive Rate} = \text{False Positive Rate}$ , which is the intersection point of the  $y = x + 1$  line and ROC.



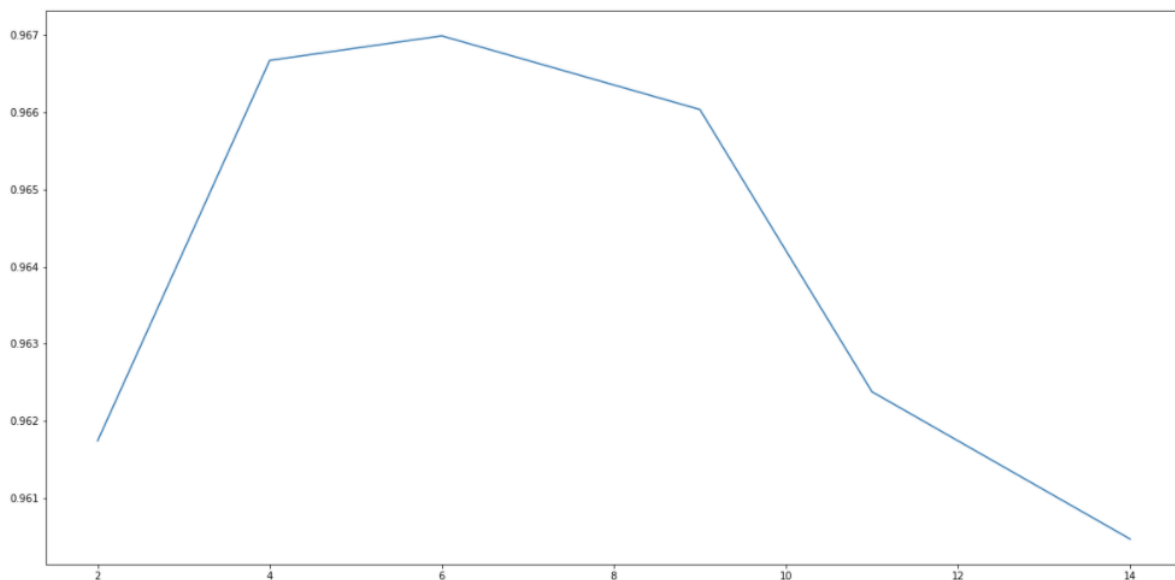
Next, the K nearest neighbor (K-NN) classifier was implemented. It takes four arguments, namely X\_train, Y\_train, X\_test, and a value k, which indicates the number of nearest neighbors.

Then we analyze the data set using the KNN classifier by taking 15% of the dataset to be the holdout set. We trained the classifier on the remaining data, using different values of K, which are 2, 4, 6, 8, 10, 12, 14 and used the holdout set to determine the best value of K. We found that when K=6 we got our best accuracy, which is 96.6

```
k_set = np.linspace(2, 14, 6, dtype = np.int16)
k_accuracy_set = k_accuracy(k_set)
plt.plot(k_set, k_accuracy_set, label="accuracy function")
```

```
100% |██████████████████| 6/6 [30:57<00:00, 309.56s/it]
```

```
] : [<matplotlib.lines.Line2D at 0x7f7dfb49e070>]
```



```
In [14]: pd.concat([pd.Series(k_set), pd.Series(k_accuracy_set)], axis=1)
```

```
Out[14]:
```

		K accuracy
	0	1
0	2	0.961746
1	4	0.966667
2	6	0.966984
3	9	0.966032
4	11	0.962381
5	14	0.960476

Then a confusion matrix was generated as follow:

```
In [16]: from sklearn.metrics import confusion_matrix
confusion_matrix(Y_true, Y_pred)
```

```
Out[16]: array([[643,  0,  0,  0,  0,  1,  1,  0,  0,  0],
 [  0, 713,  0,  0,  0,  0,  0,  0,  0,  0],
 [  5,  6, 570,  2,  0,  1,  0, 11,  1,  1],
 [  2,  2,  6, 629,  0,  5,  0,  1,  1,  2],
 [  0,  8,  0,  0, 616,  0,  1,  2,  0, 12],
 [  2,  2,  0, 10,  0, 522,  7,  0,  1,  3],
 [  3,  2,  0,  0,  2,  1, 627,  0,  0,  0],
 [  0,  8,  3,  0,  1,  0,  0, 617,  0,  2],
 [  2, 19,  1,  8,  4,  8,  3,  1, 574,  4],
 [  5,  1,  2,  6,  9,  1,  0, 16,  0, 581]])
```

We found out that digit 1, 7, 8 and 0 are the hard digits to predict.  
Finally, we output and uploaded our result to Kaggle.

Your most recent submission



Name	Submitted	Wait time	Execution time	Score
SubmissionHW1.1.1.csv	just now	2 seconds	0 seconds	0.96582

Complete

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

>\_

kaggle competitions submit -c digit-recognizer -f submission.csv -m "Message"

Make a submission

## Coding2: Write-up

For this part, we aim to practice linear regression and regularized linear regression on the training dataset and then predict the test dataset. Instead of choosing all features, we take out “PoolArea”, “MiscVal”, “MoSold”, “YrSold” to conduct our analysis.

```
#Take out useful values I want
new_columns = ["PoolArea", "MiscVal", "MoSold", "YrSold"]
x_train = train[new_columns]
y_train = train["SalePrice"]
x_test = test[new_columns]
```

We aim to analyze the relationship between each of the four features and SalePrice. Based on the conducting linear regression on the training dataset, we apply the same four features from the test dataset to predict the SalePrice. After creating the linear regression, we fit the regression on the training dataset and predict on the testing dataset.

```
regr = linear_model.LinearRegression()
#fit on the train data
regr.fit(x_train_pool, y_train)
#predict on test data
y_test_pred = regr.predict(x_test_pool)
```

The relationship between each one of four features and the SalePrice is indicated by slope and intercept.

```
print('Slope (theta1) on poolArea of test dataset: \t', regr.coef_[0]) # 182.70934030078107
print('Slope(theta0) on test dataset: \t', regr.intercept_) # 180417.1183405948
print("\n")
```

For example, from this prediction, we can tell that the bigger the pool area is, the higher the sale price of the house is.

Then, we repeat the same process for the left three features: MiscVal, MoSold, YrSold.

```
# train on Misc value and the make prediction
x_train_Misc = x_train.loc[:,['MiscVal']]
x_test_Misc = x_test.loc[:,['MiscVal']]

regr2 = linear_model.LinearRegression()
regr2.fit(x_train_Misc, y_train)
y_test_pred2 = regr2.predict(x_test_Misc)

print('Slope (theta1) on miscvlaue of test dataset: \t', regr2.coef_[0]) # -3.3930157615667067
print('Slope (theta0) on test dataset : \t', regr2.intercept_) #181068.7548923047
print("\n")
```

```

x_train_mo = x_train.loc[:,['MoSold']]
x_test_mo = x_test.loc[:,['MoSold']]

regr3 = linear_model.LinearRegression()
regr3.fit(x_train_mo, y_train)
y_test_pred3 = regr3.predict(x_test_mo)

print ('Slope (theta1) on MoSold of test dataset: \t', regr3.coef_[0]) # 1364.3505021688502
print('Slope (theta0) on test dataset : \t', regr3.intercept_) #172295.88415409692

#train on Yrsold and make prediction
x_train_yr = x_train.loc[:,['YrSold']]
x_test_yr = x_test.loc[:,['YrSold']]

regr4 = linear_model.LinearRegression()
regr4.fit(x_train_yr, y_train)
y_test_pred4 = regr4.predict(x_test_yr)

print ('Slope (theta1) on YrSold of test dataset: \t', regr4.coef_[0]) # -1730.0587285436372
print('Slope (theta0) on test dataset : \t', regr4.intercept_)

```

As we can tell, the bigger the MiscVal is, the lower the sale price is. The bigger the MoSold is, the higher the sale price is. The bigger is YrSold is, the higher the sale price is.

Next part, we apply Lasso regression and Ridge regression to the same training dataset and test dataset. Lasso regression is a type of linear regression that uses shrinkage. Thus, all the data points are shrunk towards the central point. If the distribution of data points is sparse, lasso regression could be more refined. The process for applying Lasso regression is similar to the above process.

```

model_lasso = LassoCV(alphas = [1,0.1,0.001,0.0005])
lasso1 = model_lasso.fit(x_train_pool, y_train)
y_pred_lasso1 = lasso1.predict(x_test_pool)#182.70933999
score1 = lasso1.score(x_train_pool, y_train)

print('lasso coefficient for the 1st prediction is: ',lasso1.coef_)
print('score is: ', score1) #0.008538415958697199
print('\n')

lasso2 = model_lasso.fit(x_train_Misc, y_train)
y_pred_lasso2 = lasso2.predict(x_test_Misc)#-3.3930117
score2 = lasso2.score(x_train_Misc, y_train)
print('lasso coefficient for the 2nd prediction is: ', lasso2.coef_)
print('score is: ', score2) #0.00044899828533218056
print('\n')

lasso3 = model_lasso.fit(x_train_mo, y_train)
y_pred_lasso3 = lasso3.predict(x_test_mo) #1364.33681214
score3 = lasso3.score(x_train_mo, y_train)
print('lasso coefficient for the 3rd prediction is: ', lasso3.coef_)
print('score is: ', score3) #0.0021559533965250788
print('\n')

lasso4 = model_lasso.fit(x_train_yr, y_train)
y_pred_lasso4 = lasso4.predict(x_test_yr) #-1730.00199509
score4 = lasso4.score(x_train_yr, y_train)
print('lasso coefficient for the 4th prediction is: ',lasso4.coef_)
print('score is: ', score4) #0.000836515842886465
print("\n")

```



As we can tell, all the four lasso coefficients produced by lasso regression are similar to the four slopes produced by the above linear regression. What's more, the performance of lasso regression can be indicated by its score. From all the four scores, even though they are small, this could happen due to our choices of features. Also, the score can only be produced by data from the training dataset, since SalePrice in the test dataset has to be predicted by us.

Ridge regression is also a kind of linear regression. Ridge regression works when the number of predictor variables exceeds the number of observations. The process for applying Lasso regression is similar to the above process.

```
alpha = 10
ridge = Ridge(max_iter = 100, alpha = alpha)
ridgel = ridge.fit(x_train_pool, y_train)
y_pred_ridgel = ridgel.predict(x_test_pool)
r_score1 = ridgel.score(x_train_pool, y_train)
print('Ridge coefficient for the 1st prediction is: ',ridgel.coef_) #182.70856451
print('score is: ', r_score1) #0.00853841595854321
print('\n')

ridge2 = ridge.fit(x_train_Misc, y_train)
y_pred_ridge2 = ridge2.predict(x_test_Misc)
r_score2 = ridge2.score(x_train_Misc, y_train)
print('Ridge coefficient for the 2nd prediction is: ',ridge2.coef_) # -3.39301567
print('score is: ', r_score2) #0.00044899828533262465
print('\n')

ridge3 = ridge.fit(x_train_mo, y_train)
y_pred_ridge3 = ridge3.predict(x_test_mo)
r_score3 = ridge3.score(x_train_mo, y_train)
print('Ridge coefficient for the 3rd prediction is: ',ridge3.coef_) #1363.07238599]
print('score is: ', r_score3) #0.0021559515044946576
print('\n')

ridge4 = ridge.fit(x_train_yr, y_train)
y_pred_ridge4 = ridge4.predict(x_test_yr)
r_score4 = ridge4.score(x_train_yr, y_train)
print('Ridge coefficient for the 4th prediction is: ',ridge4.coef_) #-1723.36199749
print('score is: ', r_score4) #0.0008365033991728499
```

As we can tell, all the four ridge coefficients produced by ridge regression are similar to the four slopes produced by the above two linear regressions. What's more, the performance of ridge regression can be indicated by its score. From all the four scores, even though they are small, this could happen due to our choices of features. Also, the score can only be produced by data from the training dataset, since SalePrice in the test dataset has to be predicted by us.

Both Ridge and Lasso regression can reduce model complexity and prevent over-fitting which may result from simple linear regression.

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
Submission.csv	just now	0 seconds	0 seconds	0.42592

Complete

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

$$\begin{aligned}
 Q_1: \quad & \arg \min_{\theta \in \Theta} E_{\hat{P}(x)} [KL(\hat{P}(y|x) \parallel P_{\theta}(y|x))] \\
 &= \arg \min_{\theta \in \Theta} E_{\hat{P}(x)} \left[ \sum_{y_i|x_i} \hat{P}(y_i|x_i) \log \frac{\hat{P}(y_i|x_i)}{P_{\theta}(y_i|x_i)} \right] \\
 &= \min_{\hat{P}(x_i)} \left( \sum_{x_i} \hat{P}(x_i) \sum_{y_i|x_i} \hat{P}(y_i|x_i) \log \frac{\hat{P}(y_i|x_i)}{P_{\theta}(y_i|x_i)} \right)
 \end{aligned}$$

According to Bayes' Theorem

$$\begin{aligned}
 &= \min_{\hat{P}(x_i)} \left( \sum_{x_i, y_i} \hat{P}(x_i, y_i) \log \frac{\hat{P}(y_i|x_i)}{P_{\theta}(y_i|x_i)} \right) \\
 &= \min_{\hat{P}(x_i)} \left\{ \sum_{x_i, y_i} \hat{P}(x_i, y_i) [\log \hat{P}(y_i|x_i) - \log P_{\theta}(y_i|x_i)] \right\} \\
 &= \arg \min_{\theta \in \Theta} E_{\hat{P}(x, y)} [\log \hat{P}(y|x) - \log P_{\theta}(y|x)] \\
 &= \arg \min_{\theta \in \Theta} [E_{\hat{P}(x, y)} \log \hat{P}(y|x) - E_{\hat{P}(x, y)} \log P_{\theta}(y|x)]
 \end{aligned}$$

$\therefore$  The first term doesn't depend on  $P_{\theta}$ :

$$= \arg \max_{\theta \in \Theta} E_{\hat{P}(x, y)} \log P_{\theta}(y|x)$$

Thus, the equivalence holds

Q2

(A)

A: The widget is actually defective

B: The test shows that a widget is defective

According to Bayes' rule,

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \quad (*)$$
$$= \frac{P(B|A) \cdot P(A)}{P(B|A)P(A) + P(B|A^c)P(A^c)}$$

$$P(B|A) = 0.95, \quad P(B|A^c) = 1 - P(B^c|A^c) = 1 - 0.95 = 0.05$$

$$P(A) = \frac{10,000,000}{100,000,000} = 0.00001$$

$$P(A^c) = 1 - 0.00001 = 0.99999$$

$$\therefore (*) = \frac{0.95 \times 0.00001}{0.95 \times 0.00001 + 0.05 \times 0.99999} \approx 0.019\%$$

(B)

$$| \text{Total non-defective widgets} = 10,000,000 \times 0.99999 = 9999900$$

$$P(B|A^c) = 1 - P(B^c|A^c) = 0.05 \quad // \text{ The test shows that the}$$

widget is defective, given to that

the widget isn't defective

$$\therefore \# \text{ Non-defective but thrown away widget} = 9999900 \times 0.05 = 499995$$

$$P(B^c|A) = 1 - 0.95 = 0.05 \quad // \text{ the test shows that a widget isn't defective, given to it's actually defective}$$

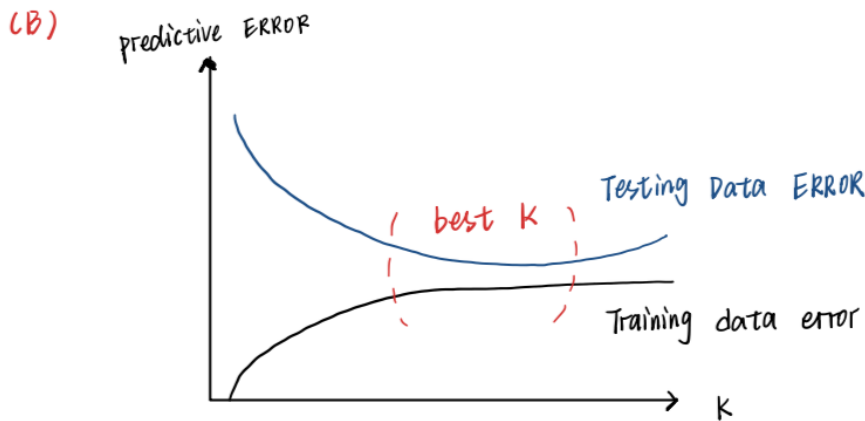
$$\therefore \# \text{ Defective but not thrown away} = (10,000,000 - 9999900) \times 0.05 = 5$$

Q3

(A) When  $k=1$ , there will be no training error. In other words, the training data can be perfectly predicted. Since we just pick 1 nearest neighbour, the probability of choosing from each category is the same. Thus, the bias is 0. However, since the model predicts the training data too well, it's too good to be possible. Thus, bad overfitting happens.

As  $k$  increases (increases to  $N$ ), the bias increases. As we include more data points, it's inevitable that the chance of choosing data from each category starts to be unequal. On the contrary to  $k=1$ , underfitting happens under this situation. ( $\uparrow k$  = underfit = high bias + low variance)

$\downarrow k$  = overfit = low bias + high variance



When  $k=1$ , there are no training errors due to low bias. However, based on (A), variance under such situation can be high which makes the algorithm pay more attention to training data rather than testing data. Thus, the testing data error is big under such situation. As  $k$  increases, the bias produced by training data increases, the training data error increases as the variance decreases, however, the testing data error decreases. Therefore, when testing data error and training data error reach the similar level, we get the best  $k$ .

(c) we can improve the performance of KNN algorithm by weighed voting.

suppose  $x$  is the data point, whose label  $y$  has to be predicted. Then select the set of  $K$  nearest training data

points to the query points. Then apply distance-weighted voting

$$y' = \underset{v}{\operatorname{argmax}} \sum_{(x_i, y_i)} W_i \times I(v = y_i) \quad // v \text{ represents the class labels}$$

↓ Each  $K$  neighbor is multiplied by the Inverse (distance between selected point and neighbor)