CS5785: Homework #1

Due on October 1, 2020

Tianyou Xiao (tx43), Ziyu Song (zs363)

1 Digit Recognizer

For this programming exercise, we used the Digit recognizer dataset from Kaggle [1].

1.1 Display Digit

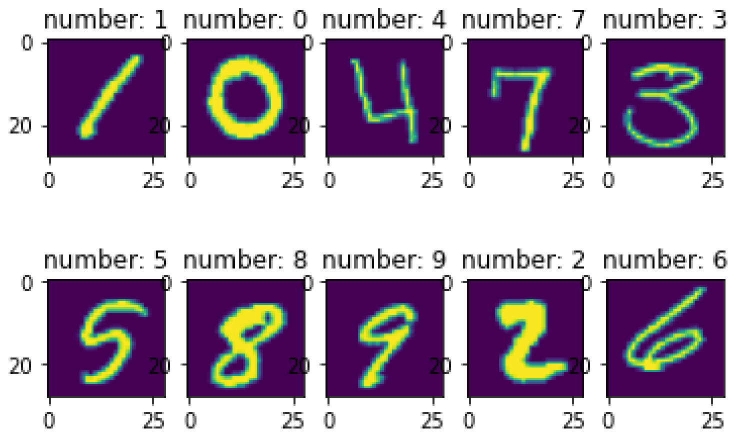


Figure 1: Display MNIST Digits

1.2 Prior Probability

The distribution of the prior probability of the classes in the training data is uniform across the digits.

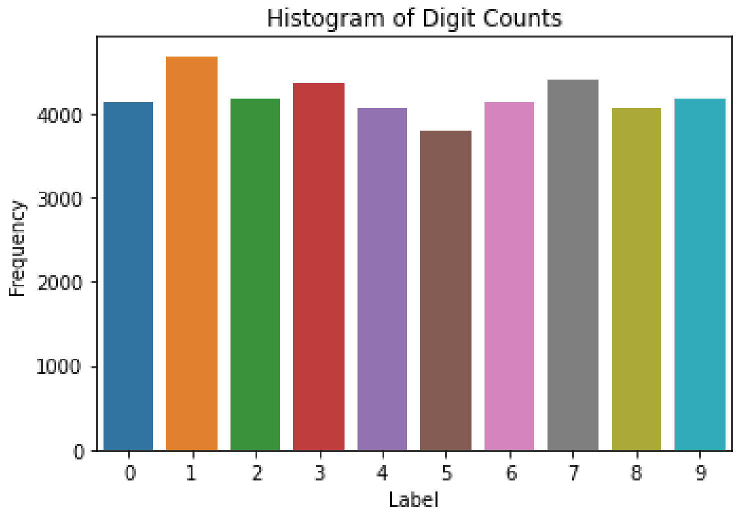


Figure 2: Histogram of Digit Counts

We also plotted the normalized histogram of digits counts. The distribution is roughly uniform, with several digits being slightly more frequent. Each class basically has the same probability. It's evenly distributed as the un-normalized version. The normalized histogram is shown in Figure 3:

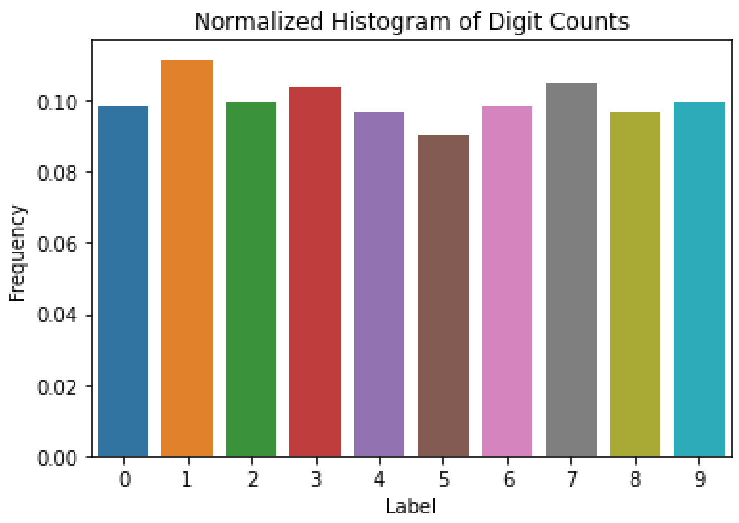


Figure 3: Normalized Histogram of Digit Counts

1.3 Best Match (Nearest Neighbor)

The strategy works for most of the digits. However, we can directly tell from the results that the prediction is incorrect for digit 3: a 5 is found as the best match for 3. We added an asterisk next to the example.

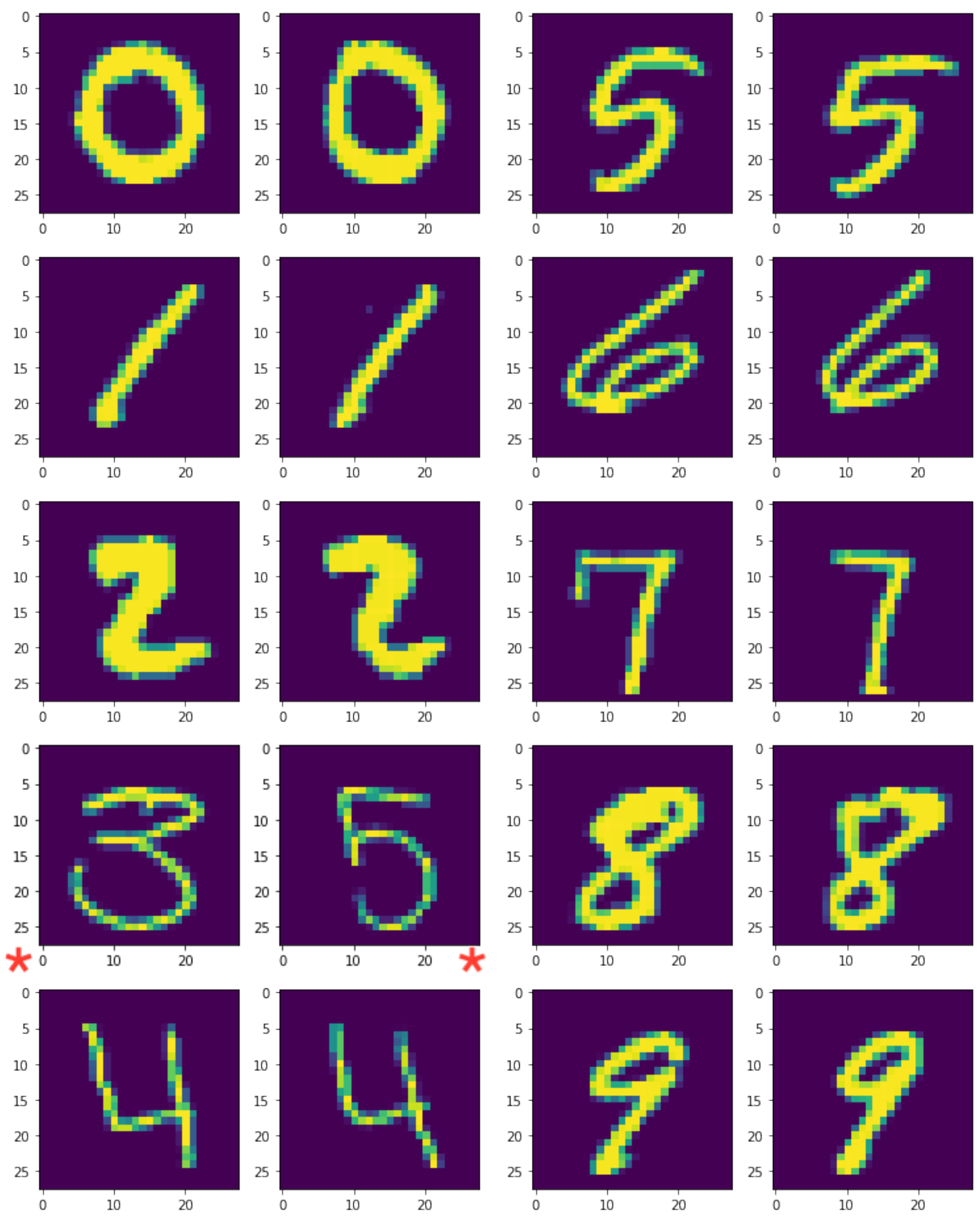


Figure 4: Best match between chosen sample and the rest of training data

1.4 Binary Comparison

In this section, we computed the pairwise distances of all genuine matches and all imposter matches. Genuine distances are the distances of zeros with themselves and ones with themselves. Imposter distances are the distances between zeros and ones. The histogram (normalized) is shown below.

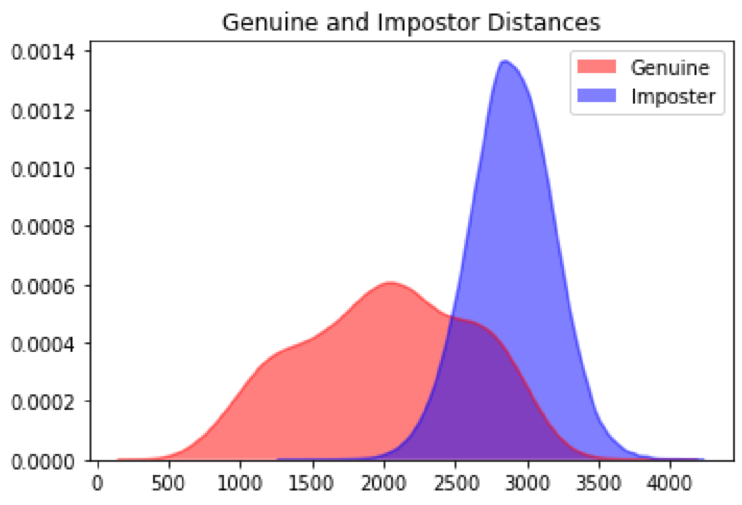


Figure 5: Histograms of the Genuine and Imposter Distances

1.5 ROC Curve

Equal error rate (EER) is around 0.1855, where $TPR + FPR = 1$ (the intersection of EER and ROC curve in the plot). The error rate of random guess is 0.5.

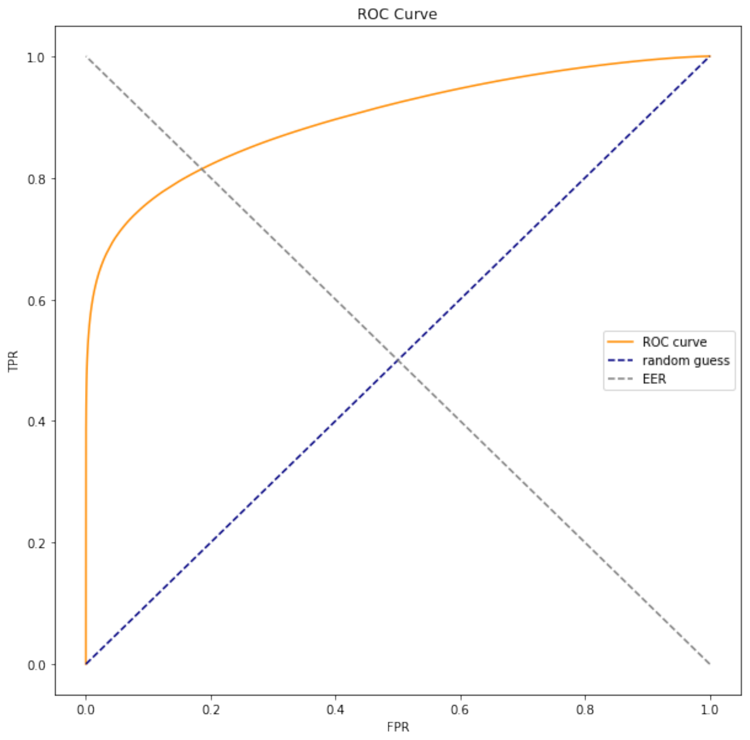
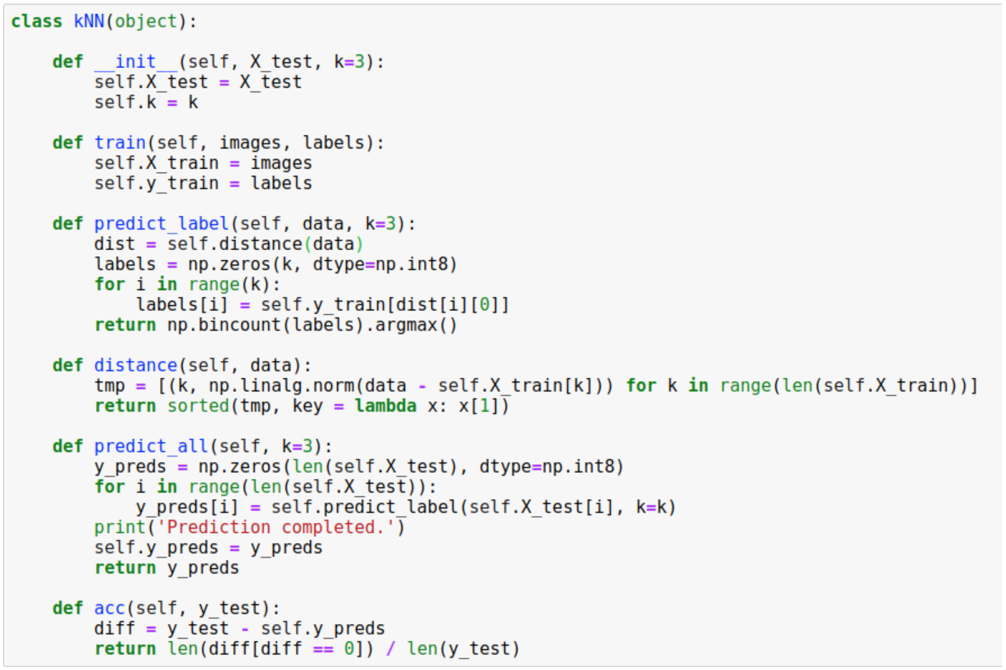


Figure 6: ROC Curve

1.6 K-NN Classifier

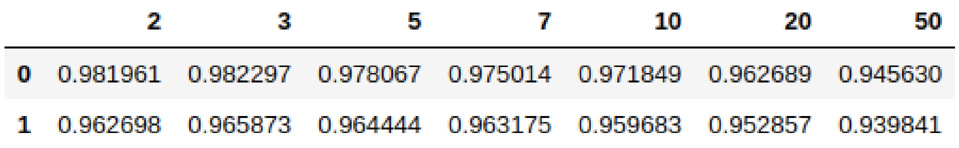
We implement the K-NN classifier as a class, as shown below:



```python
class kNN(object):

    def __init__(self, X_test, k=3):
        self.X_test = X_test
        self.k = k

    def train(self, images, labels):
        self.X_train = images
        self.y_train = labels

    def predict_label(self, data, k=3):
        dist = self.distance(data)
        labels = np.zeros(k, dtype=np.int8)
        for i in range(k):
            labels[i] = self.y_train[dist[i][0]]
        return np.bincount(labels).argmax()

    def distance(self, data):
        tmp = [(k, np.linalg.norm(data - self.X_train[k])) for k in range(len(self.X_train))]
        return sorted(tmp, key = lambda x: x[1])

    def predict_all(self, k=3):
        y_preds = np.zeros(len(self.X_test), dtype=np.int8)
        for i in range(len(self.X_test)):
            y_preds[i] = self.predict_label(self.X_test[i], k=k)
        print('Prediction completed.')
        self.y_preds = y_preds
        return y_preds

    def acc(self, y_test):
        diff = y_test - self.y_preds
        return len(diff[diff == 0]) / len(y_test)
```

Figure 7: Our implementation of K-NN

We tried a bunch of K values. As shown in the x-axis, the K value we choose are [2,3,5,7,10,20,50]. Due to limited time, we use the K-NN function provided by sklearn to find the optimal K. From the plot, we can tell that the optimal K for this dataset is $K = \mathbf{3}$, and we ignore the trivial case of $K = 1$ (only considering the label of the data point itself). The table and plot showing the values of K as well as their training and holdout accuracy is provided below:



	2	3	5	7	10	20	50
0	0.981961	0.982297	0.978067	0.975014	0.971849	0.962689	0.945630
1	0.962698	0.965873	0.964444	0.963175	0.959683	0.952857	0.939841

Figure 8: Table of k vs accuracy. Column names represent K values. First row is the accuracy for training set and second row is the accuracy for holdout set.

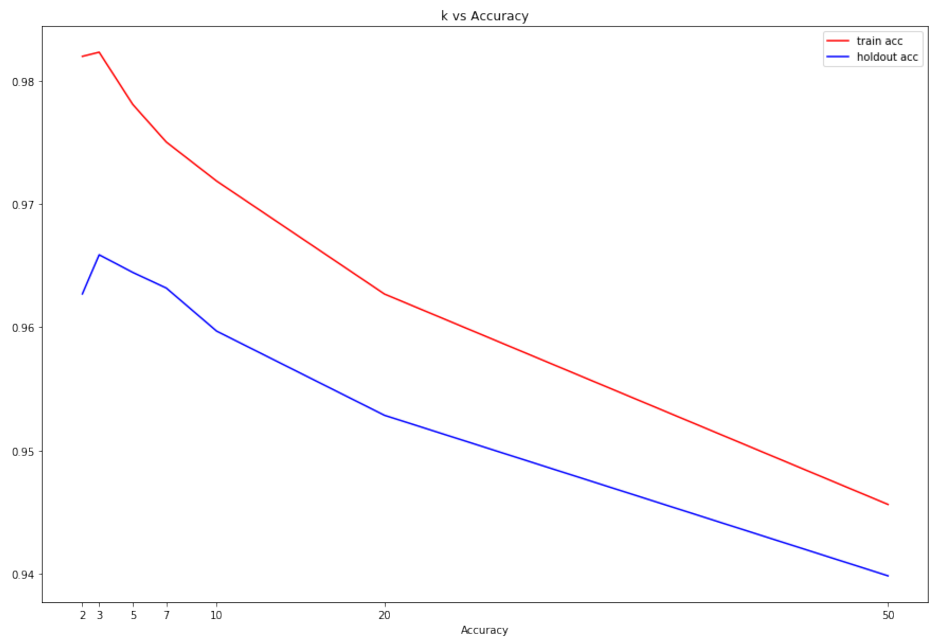
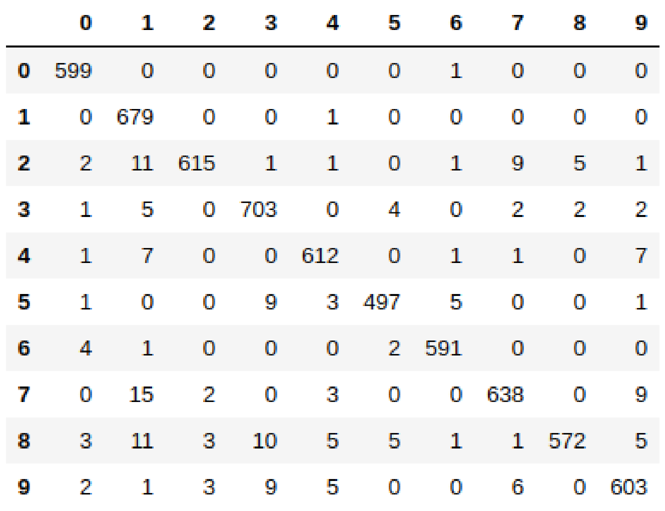


Figure 9: Plot of k vs accuracy. The peak happens at K=3.

1.7 Confusion Matrix

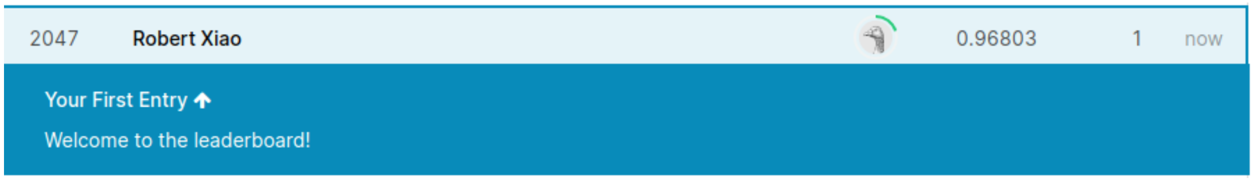


	0	1	2	3	4	5	6	7	8	9
0	599	0	0	0	0	0	1	0	0	0
1	0	679	0	0	1	0	0	0	0	0
2	2	11	615	1	1	0	1	9	5	1
3	1	5	0	703	0	4	0	2	2	2
4	1	7	0	0	612	0	1	1	0	7
5	1	0	0	9	3	497	5	0	0	1
6	4	1	0	0	0	2	591	0	0	0
7	0	15	2	0	3	0	0	638	0	9
8	3	11	3	10	5	5	1	1	572	5
9	2	1	3	9	5	0	0	6	0	603

Figure 10: Confusion Matrix

Figure 10 shows the confusion matrix generated on holdout dataset by the K-NN model. The column names represent the actual labels and the row names represent the predictions provided by the classifier. Here we can see that there is a relatively large amount of misclassification for number 1 and 9 (from the column #1 and #9). Also regarding misclassification between pairs, it's tricky to distinguish between (1,7), (1,2) and (1,8).

1.8 Kaggle Submission


2047 Robert Xiao 0.96803 1 now

Your First Entry

Welcome to the leaderboard!

Figure 11: Kaggle Submission and leaderboard

Based on the evaluation from Kaggle, the model implemented by us achieved 96.803% of accuracy on the test set.

2 House Prices

For this programming exercise, we used the Digit recognizer dataset from Kaggle [2].

2.1 Least Squares

We apply several basic criteria for preprocessing:

- drop features where most values are missing (in this case we drop *PoolQC, MiscFeature, Alley, Fence, FireplaceQu*)
- fill other missing values with mode (for categorical data) and mean (for numerical data)
- deal with skewness within numerical feature by taking log transformation
- create dummy variables for categorical features

Our model achieves a performance of $R^2 = 0.949$ and $RMSE = 0.00803523$. The scatter plot below shows a nice linear trend between the actual prices and our predictions.

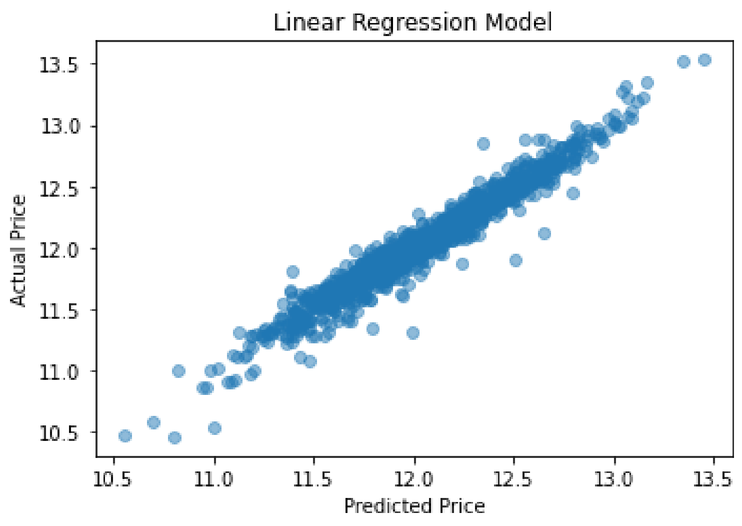


Figure 12: Scatter plot of pred vs actual for Least Squares

2.2 Regularized Least Squares

In this section, we tried to predict housing prices using regularized least squares: L1: Lasso Regulation and L2: Ridge Regulation. We want to see if using regulation least squares can help to decrease the influence of less important features. We've also tried using different values of alpha (regularization strength) to see the effect of regularization term on the performance. The results are shown in the scatterplots in Figure 13.

From the plots we discover that:

- Model with L1 regularization (Lasso) performs worse than model with L2 regularization (Ridge).
- Both models perform worse when the value of alpha increases.
- Overall the regularized least squares model perform worse than normal least squares model.

2.3 Kaggle Submission

Among over 5000 submissions, our submission ranked at 2447, which is almost middle of the leaderboard. Our score is 0.13795, as shown in Figure 14.

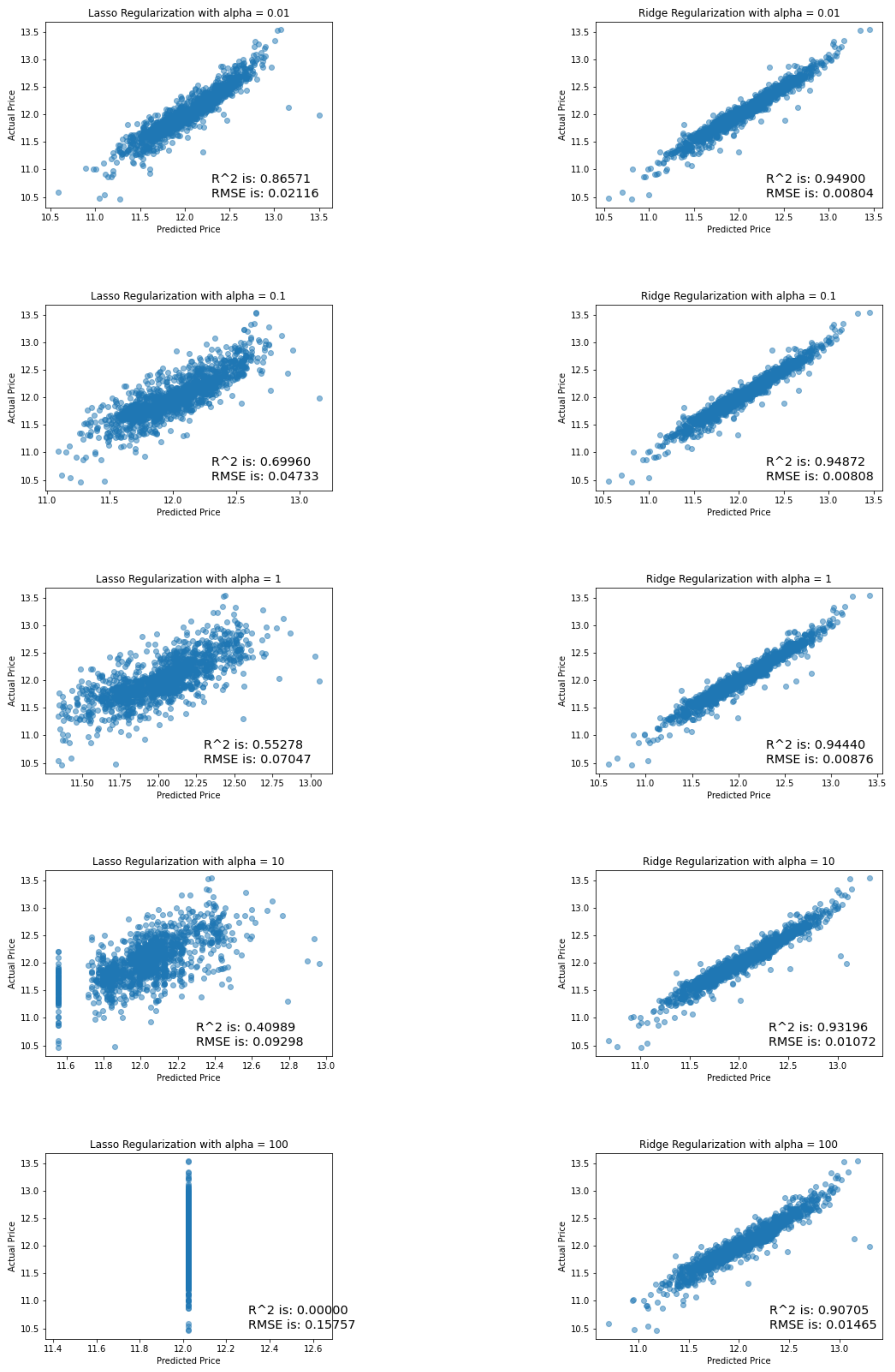
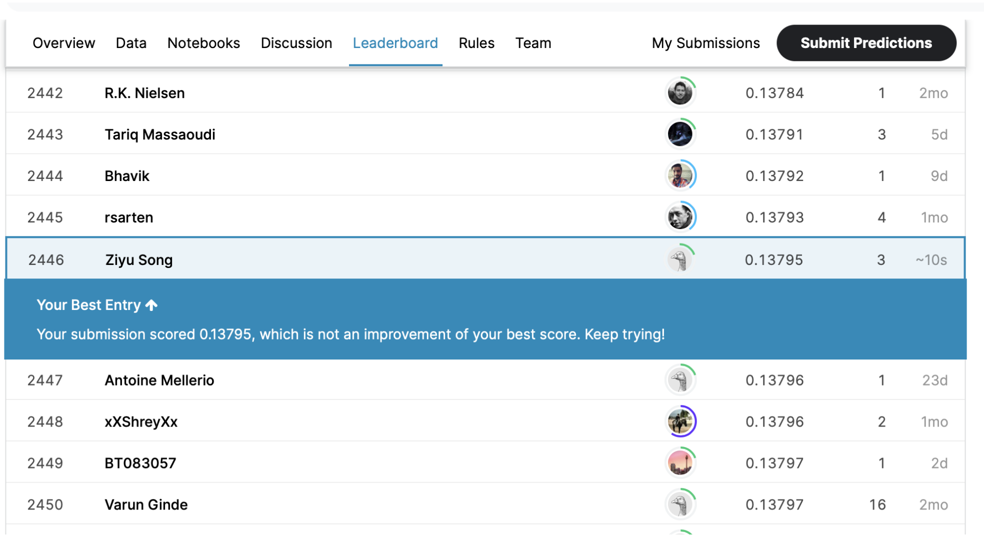


Figure 13: L1 and L2 Regularization Comparison



2442	R.K. Nielsen	0.13784	1	2mo
2443	Tariq Massaoudi	0.13791	3	5d
2444	Bhavik	0.13792	1	9d
2445	rsarten	0.13793	4	1mo
2446	Ziyu Song	0.13795	3	~10s

Your Best Entry
Your submission scored 0.13795, which is not an improvement of your best score. Keep trying!

2447	Antoine Mellerio	0.13796	1	23d
2448	xXShreyXx	0.13796	2	1mo
2449	BT083057	0.13797	1	2d
2450	Varun Ginde	0.13797	16	2mo

Figure 14: Kaggle Submission

3 Written Exercises

3.1 Maximum Likelihood and KL Divergence

$$\begin{aligned} RHS &= \arg\min_{\theta\in\Theta} E_{\hat{p}(x)}[KL(\hat{p}(y|x) \parallel p_\theta(y|x)] \\ &= \arg\min_{\theta\in\Theta} E_{\hat{p}(x)}[E_{\hat{p}(y|x)}[\log \hat{p}(y|x) - \log P_\theta(y|x)]] \end{aligned}$$

Here, we can see that the first term in the expectation does not involve anything related to θ, therefore it can be ignored. The RHS then becomes:

$$\arg\min_{\theta\in\Theta} E_{\hat{p}(x)}[E_{\hat{p}(y|x)}[-\log P_\theta(y|x)]]$$

The dual expectation with condition of $\hat{p}(x)$ and $\hat{p}(y|x)$ can be transformed into expectation given $\hat{p}(x,y)$, according to the Bayes Rule, where $\hat{p}(x,y) = \hat{p}(y|x) \times \hat{p}(x)$:

$$\begin{aligned} RHS &= \arg\min_{\theta\in\Theta} E_{\hat{p}(x,y)}[-\log p_\theta(y|x)] \\ &= \arg\max_{\theta\in\Theta} E_{\hat{p}(x,y)}[\log p_\theta(y|x)] \\ &= LHS \quad \square \end{aligned}$$

3.2 Bayes Rule for Quality Control

Let P demotes for Test Positive, and D denotes for Defective.

$$Pr(P|D) = 0.95$$

$$Pr(D) = 0.00001$$

$$Pr(P) = Pr(D) * Pr(P|D) + Pr(\neg D) * (1 - Pr(P|D))$$

$$Pr(P) = 0.00001 * 0.95 + (1 - 0.00001) * (1 - 0.95) = 0.050009$$

$$Pr(D|P) = \frac{\Pr(P|D)\Pr(D)}{\Pr(P|D)\Pr(D) + \Pr(P|\neg D)\Pr(\neg D)} = 1.899 \times 10^{-4}$$

The chances that it's actually defective given the test result positive is 1.899×10^{-4}.

The amount of good widgets are thrown away is:

$$Pr(P \cap \neg D) * widgets = 0.05 * (1 - 0.00001) * 10^7 = 499995$$

The amount of bad widgets are still shipped to customers is:

$$Pr(\neg P \cap D) * widgets = (1 - 0.95) * 0.00001 * 10^7 = 5$$

3.3 K-Nearest Neighbors Classification

3.3.1 (a)

The average 0-1 prediction error should be high when k is too large (close to n). The error rate decreases as the value of k decreases. The error rate may rise again after the optimal k value (excluding k=1). In the setting of this question, when k=1, the model will perfectly predict every data point. However, although choose k=1 might result in low prediction error, the model will tend to overfit and not generalize well on new data.

3.3.2 (b)

The prediction error should be high when k is too large (close to n) or too small (close to 1), and relatively low when k is reasonably large (e.g. $sqrt(n)$). Compared with the previous question, the problem of overfitting will lead to poor performance on the held-out dataset simply because the poor generalization of the model.

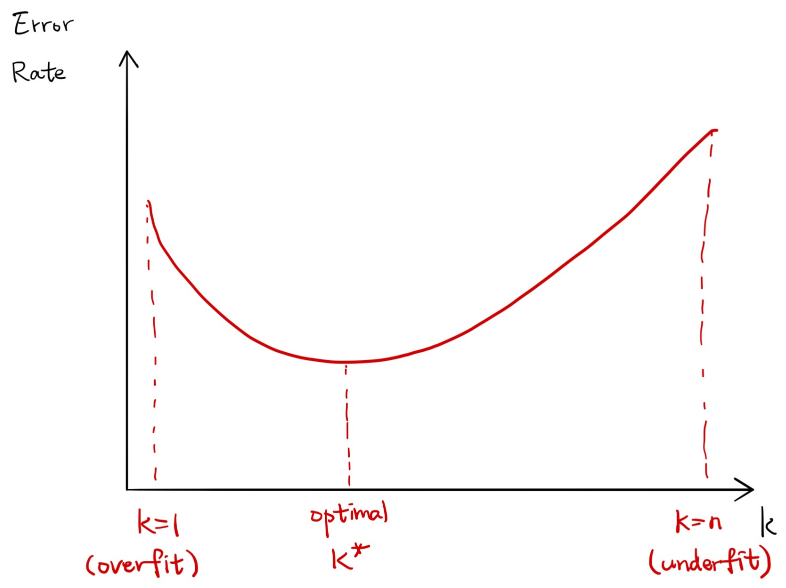


Figure 15: Sketch

3.3.3 (c)

We can use weighted average instead of uniform vote. The weight can come from some kernel function, where the weight value decreases as the distance increases. For example, we can simply use the inverse distance as weight.

References

[1] "Digit recognizer." [Online]. Available: https://www.kaggle.com/c/digit-recognizer

[2] "House prices: Advanced regression techniques." [Online]. Available: https://www.kaggle.com/c/house-prices-advanced-regression-techniques