Ryan Farr
rlf238
CS 5785 – Applied Machine Learning

Homework 1

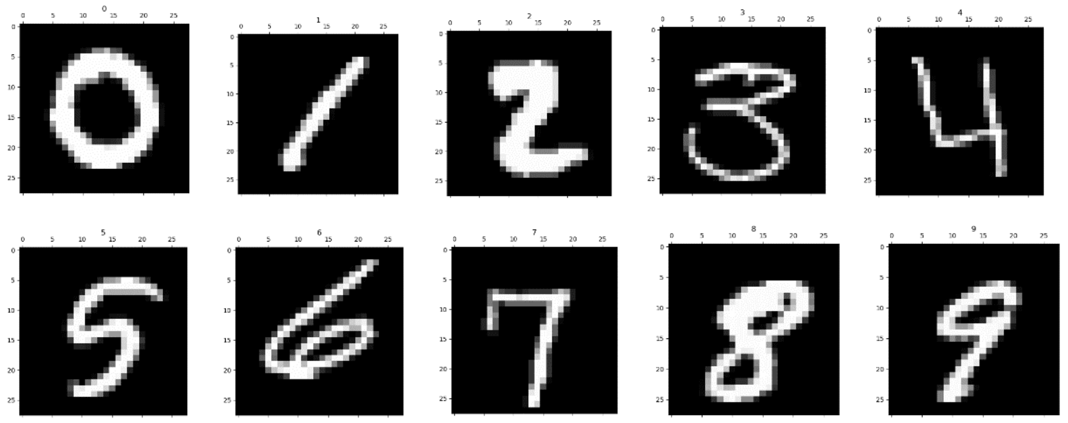
General Summary

In this assignment I developed a digit recognizer using a custom *K-NN* implementation, a Titanic survival classifier using an existing implementation of logistic regression and completed a set of written exercises. I'm impressed by the final accuracy of the digit recognizer, which had an accuracy of ~96% on the final testing set that was submitted to Kaggle. However, the challenge of using *K-NN* on a large dataset became very clear while doing the final testing, which had to be run overnight as it took several hours. Running logistic regression, however, took nearly no time at all for the given dataset but did not perform nearly as well on the final testing set.

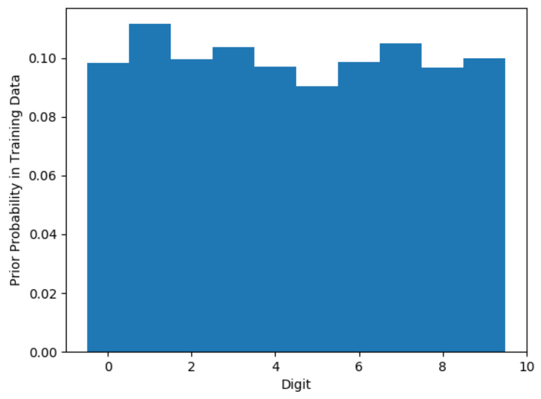
Digit Recognizer

a. Joined the Kaggle competition with username *rlf238*.

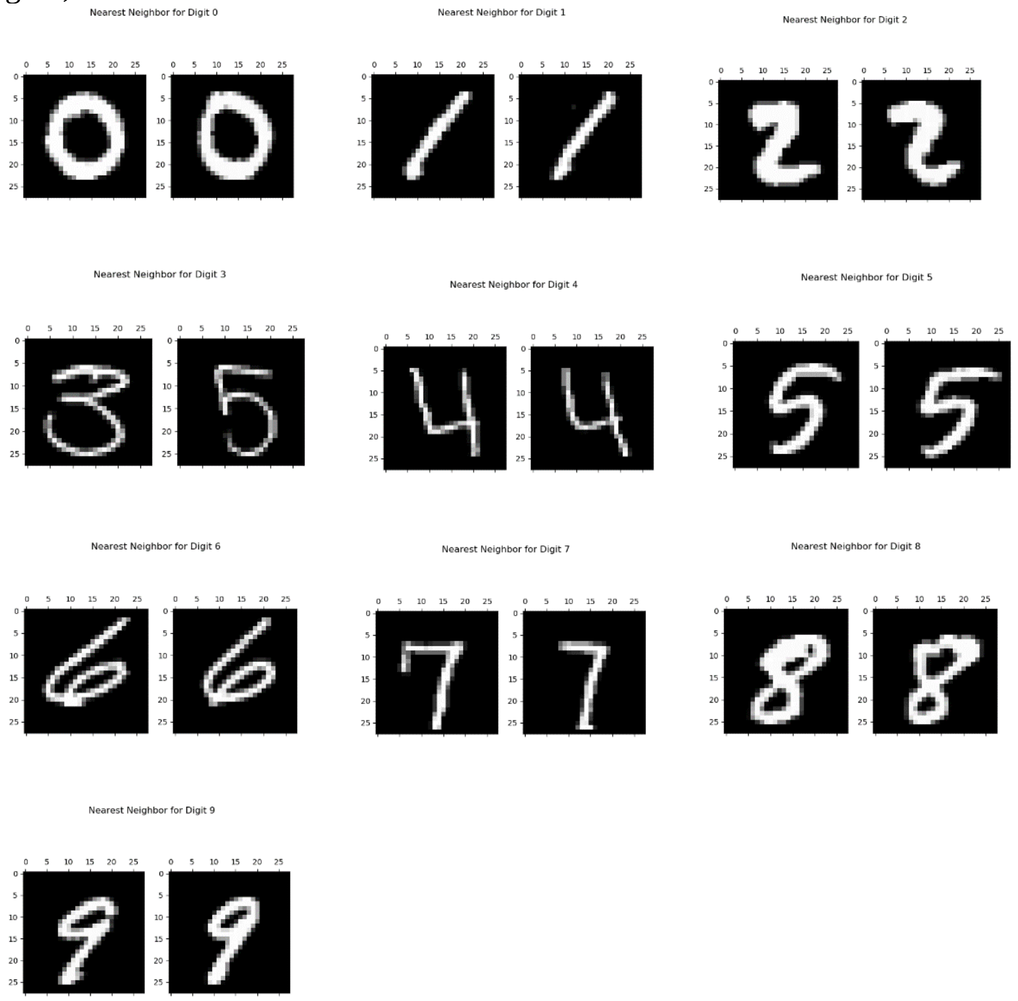
b. Below are 9 of the digits.



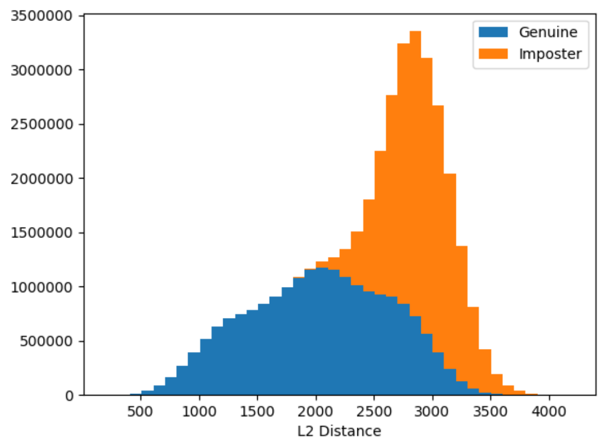
c. Below is a normalized histogram of the digit counts in the training data. As can be seen, it isn't entirely uniform across digits (2 is more likely than 5, for example); however, it's fairly uniform.



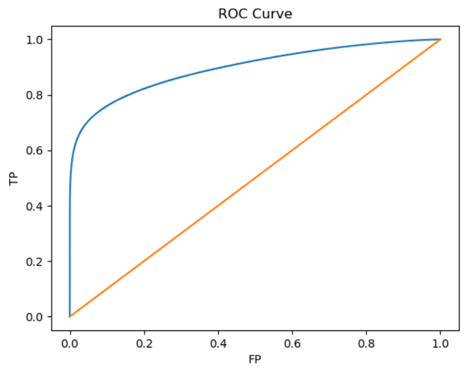
d. Below are the best match for digits selected from the training data. Note that data points were not compared to themselves. As can be seen, the only failure was with the selected digit 3, which chose a similar 5.



e. Below is a histogram of genuine and imposter match distances generated from the training dataset and using 40 bins.



f. Below is the ROC curve generated from the above histogram. The equal error rate occurs close to 1.0 (between 0.9 and 1.0). The worst case is shown in orange, which can be seen as randomly guessing. A random classifier should get as many true positives as false positives, true negatives as false negatives.



g. K-NN was implemented using a heap. The implementation can be found in *knn.py*.

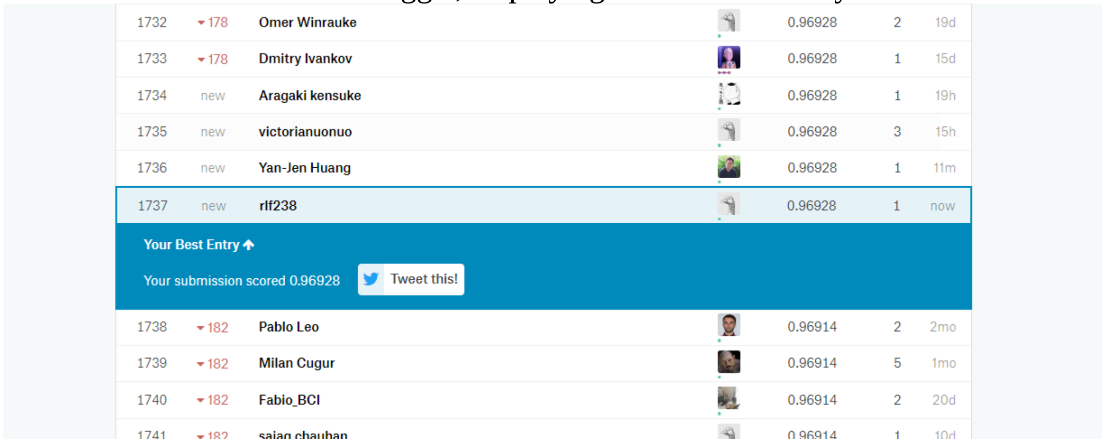
h. Due to time constraints, 3-fold cross validation was performed on a subset of the training data and contained roughly 2,000 data points. Values of 1 through 5 were tested for K using cross validation. As can be seen in the table below, $K = 3$ was the most successful value tested. 3-Fold cross validation was run again on the full dataset with only $K = 3$. The data from this run is what was used to generate the confusion matrix.

	K = 1	**K = 2**	**K = 3**	**K = 4**	**K = 5**
2,000 pts	0.8825	0.866	0.8875	0.875	0.8775
Full Training Set	-	-	0.9661	-	-

i. Below is a confusion matrix generated from 3-fold cross validation with $K = 3$. As can be seen, the numbers most similar when written tend to be confused: 9 and 4, 6 and 8, 2 and 7, 1 and 7, and so on.

		Guess									
		0	1	2	3	4	5	6	7	8	9
Actual	0	**9.7857%**	0.0000%	0.0738%	0.0143%	0.0071%	0.0333%	0.0619%	0.0048%	0.0476%	0.0333%
	1	0.0000%	**11.0833%**	0.1095%	0.0286%	0.1024%	0.0048%	0.0143%	0.1286%	0.1071%	0.0214%
	2	0.0095%	0.0190%	**9.4810%**	0.0714%	0.0000%	0.0024%	0.0000%	0.0214%	0.0238%	0.0048%
	3	0.0000%	0.0024%	0.0286%	**9.9571%**	0.0000%	0.1548%	0.0000%	0.0024%	0.1000%	0.0619%
	4	0.0000%	0.0048%	0.0071%	0.0000%	**9.2548%**	0.0048%	0.0119%	0.0190%	0.0333%	0.0905%
	5	0.0119%	0.0024%	0.0071%	0.0881%	0.0000%	**8.6048%**	0.0405%	0.0000%	0.1619%	0.0238%
	6	0.0238%	0.0095%	0.0095%	0.0048%	0.0357%	0.1024%	**9.7143%**	0.0000%	0.0405%	0.0048%
	7	0.0000%	0.0167%	0.1738%	0.0500%	0.0095%	0.0048%	0.0000%	**10.1619%**	0.0238%	0.1381%
	8	0.0048%	0.0071%	0.0357%	0.0881%	0.0071%	0.0333%	0.0071%	0.0000%	**9.0286%**	0.0476%
	9	0.0024%	0.0071%	0.0190%	0.0571%	0.2786%	0.0905%	0.0000%	0.1405%	0.1071%	**9.5452%**

j. The full training set was used to train a 3-NN classifier, then tested using the testing set. The results were submitted to Kaggle, displaying 96.298% accuracy.



1732	▾178	Omer Winrauke	0.96928	2	19d
1733	▾178	Dmitry Ivankov	0.96928	1	15d
1734	new	Aragaki kensuke	0.96928	1	19h
1735	new	victorianuonuo	0.96928	3	15h
1736	new	Yan-Jen Huang	0.96928	1	11m
1737	new	rlf238	0.96928	1	now

Your Best Entry ↑

Your submission scored 0.96928

1738	▾182	Pablo Leo	0.96914	2	2mo
1739	▾182	Milan Cugur	0.96914	5	1mo
1740	▾182	Fabio_BCI	0.96914	2	20d
1741	▾182	sajag chauhan	0.96914	1	10d

Titanic Disaster

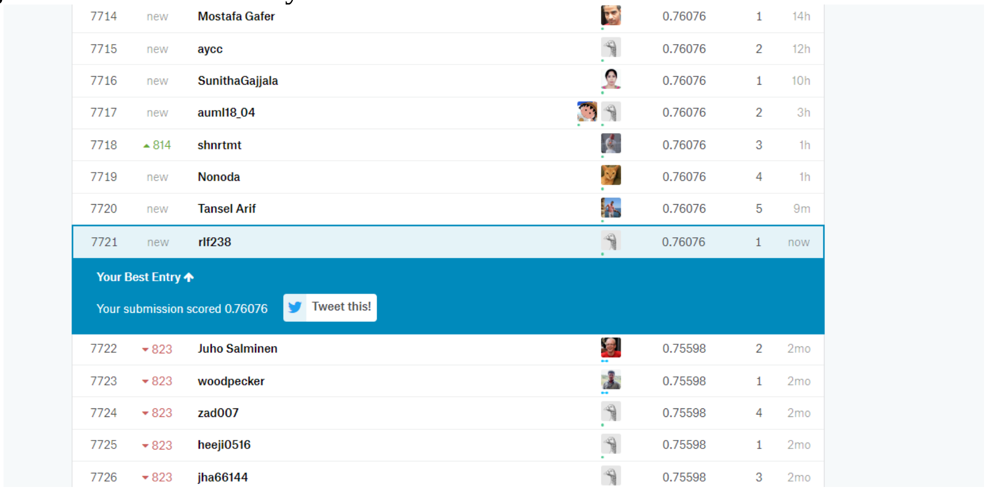
a. Joined the Kaggle competition with username *rlf238*.

b. I used the logistic regression classifier found in the *sklearn* python library. Below is a table of the features I chose to use in the training set and why.

Variable	Included?	Reasoning
pclass	Yes	In general, it's safe to assume that rich people get saved and pclass acts as a proxy for wealth.
passengerId	No	Random value, not included
sex	Yes	Women and children are often saved first.
name	No	May as well be random for deciding survival
age	Yes	Children (along with women) are often saved first.
sibsp	Yes	Family works to save family

parch	Yes	Family works to save family
ticket	No	Ticket number is random, should not play into survival
fare	No	Fare isn't a good proxy for any relevant information
cabin	No	Cabin number would be useful it could be mapped to specific areas within the Titanic (areas that were hit first would likely have more casualties.) However, no such mapping is feasible here and it is thus excluded.
embarked	No	The port of embarkment shouldn't reasonably play a role in survival.

c. All training data was used to train the classifier, then the classifier was tested using the testing set. The final accuracy was 76.076%.



7714	new	Mostafa Gafer	0.76076	1	14h
7715	new	aycc	0.76076	2	12h
7716	new	SunithaGajjala	0.76076	1	10h
7717	new	auml18_04	0.76076	2	3h
7718	▲814	shnrtmt	0.76076	3	1h
7719	new	Nonoda	0.76076	4	1h
7720	new	Tansel Arif	0.76076	5	9m
7721	new	rlf238	0.76076	1	now

Your Best Entry ↑

Your submission scored 0.76076

Tweet this!

7722	▼823	Juho Salminen	0.75598	2	2mo
7723	▼823	woodpecker	0.75598	1	2mo
7724	▼823	zad007	0.75598	4	2mo
7725	▼823	heeji0516	0.75598	1	2mo
7726	▼823	jha66144	0.75598	3	2mo

Written Exercises

1. $var(x-y) = E((x-y)^2) - (\mu_x - \mu_y)^2$
   $var(x-y) = E(x^2 - 2xy + y^2) - \mu_x^2 + 2\mu_x\mu_y - \mu_y^2$
   $var(x-y) = (E(x^2) - \mu_x^2) + (E(y^2) - \mu_y^2) - 2E(xy) + 2\mu_x\mu_y$
   $var(x-y) = var(x) + var(y) - 2(E(xy) - \mu_x\mu_y)$
   $var(x-y) = var(x) + var(y) - 2cov(x,y)$

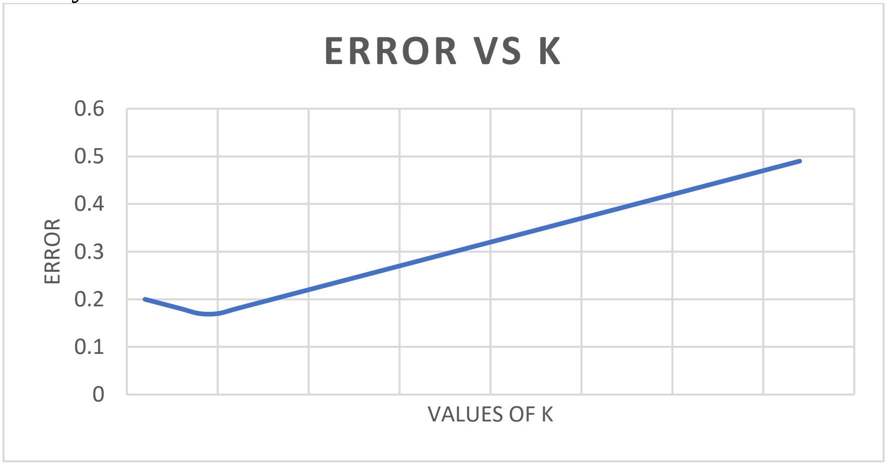
2. Let $+ = test\ positive$, $- = test\ negative$, $d = defective$, and $\sim d = not\ defective$
   a. From the description we know $pr(+|d) = 0.95$, $pr(-|\sim d) = 0.95$, and $pr(d) = 0.00001$. Here we're going to use Bayes theorem to get $pr(d|+) = \frac{pr(d)pr(+|d)}{pr(+)}$.
   We need to solve for $pr(+) = pr(+|d)pr(d) + pr(+|\sim d)pr(\sim d) = (0.95)(0.00001) + (1-0.95)(1-0.00001) = 0.050009$
   Next, we simply plug in:
   $$pr(d|+) = \frac{(0.00001)(0.95)}{0.050009} = 0.000189$$

b. We know that 10,000,000 units are produced each year with $\frac{1}{100,000}$ of the units being defective. In total, that's 9,999,900 good units and 100 bad units produced each year. Testing the good units, 5% will test as defective, meaning 499,995 good units will be thrown out and 5 bad widgets will be shipped.

3.

a. At $k = n$ we should see the accuracy be as good as guessing (50% accuracy), because kNN will return an equal number of votes for each label. On the other end of the spectrum, at $k = 1$,we should see 100% accuracy because we're testing on our training set, so kNN will only count the vote with 0 distance. kNN will include the "self" vote for all values of $k \geq 1$, but this vote becomes diluted with higher k. Thus, we'd expect to see accuracy peak at 100% with $k = 1$, then gradually decline to 50% at $k = n$.

b. Now that the testing set is removed from the training set we'll see that $1NN$ is less accurate, but $n - NN$ should still be 50% accurate for the same reason as described in part a. $1NN$ would likely test more poorly in the overlap between the labels than $2 - NN$ or $3 - NN$, but would test equally well to them in the areas that don't overlap. Thus, I'd expect to see a slight bump in accuracy with $k > 1$ which slowly declines to 50% at $k = n$.



c. First, we'll look at how computation scales with the number of folds: as k increases we see computation increase linearly. This is because, independent of the number of folds, there will be the same number of calculations/comparisons on each pass (we're going over the entire dataset no matter what); however, with higher-fold cross validation we're doing more passes. We can also look at accuracy with cross validation: at one end of the spectrum we have 2-fold cross validation. With this, we do equal testing and training. However, when you train on the entire dataset you'll have twice as many training points and thus your testing results may be a low-ball estimate of what your actual accuracy would be. If we look at high-fold cross validation, though, we'll see highly trained models that may over-estimate accuracy because the testing set is so small with each pass. Thus, it's all about getting the right ratio of training to testing data. I personally recommend 3-fold cross validation for most problems so that you use the majority of your dataset for training but still maintain 33% of data for testing on each pass and you keep computation relatively low. However, your choice of cross validation will depend on your dataset and situation.

d. Simply, add weight to the votes that is inversely proportional to distance. That is, closer neighbors have more vote than neighbors that are far away.

e. The first and most obvious reason is that kNN is more computationally expensive in higher dimensions. The second (and less obvious) reason is that, in higher dimensions, the difference between "close" and "far" points becomes less obvious, and thus the classifier will perform poorly. For example, if we compare two points to a query and one of them is very far away in one dimension but relatively close in the others and the other is moderately close in all dimensions then they may have the same distance. If the problem had lower dimensions then the first point would have a larger distance from the query than the second point.