CS5785 Homework 1

Applied Machine Learning

Vijay Pillai
Thomas Matecki

September 13, 2017

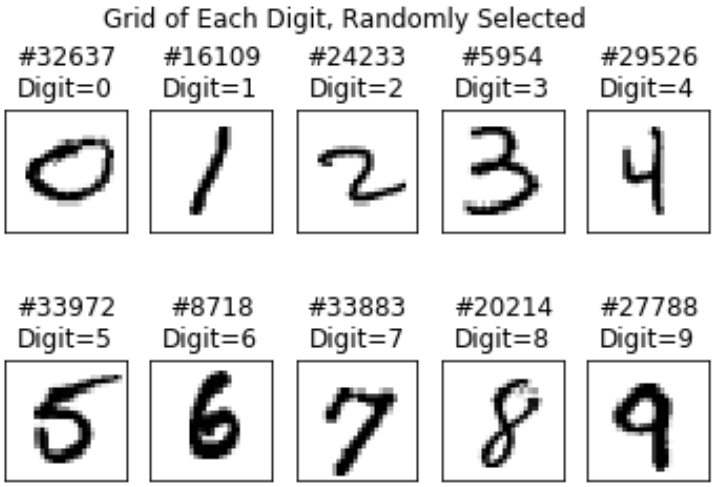
Programming Exercises

1. Digit Recognizer

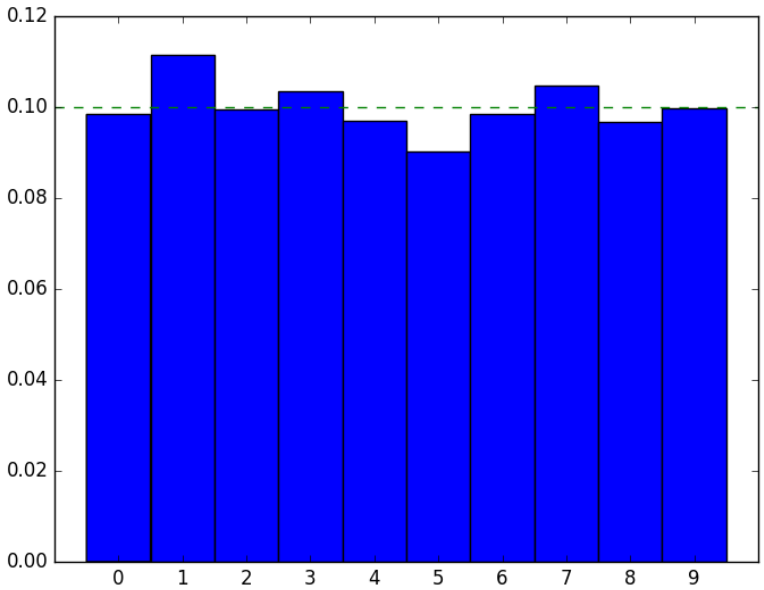
   (a) Training and test data are located in files *train.csv* and *test.csv* respectively. The training data comprises 42000 images and the test data comprises 28000 images.

   (b) Figure 1 is one of each digit(0-9) displayed as a 28 by 28 pixel grid.

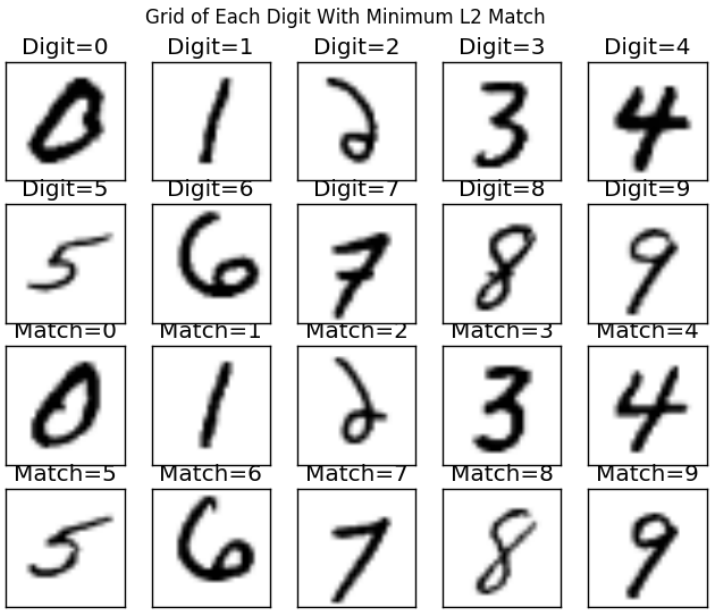
Figure 1



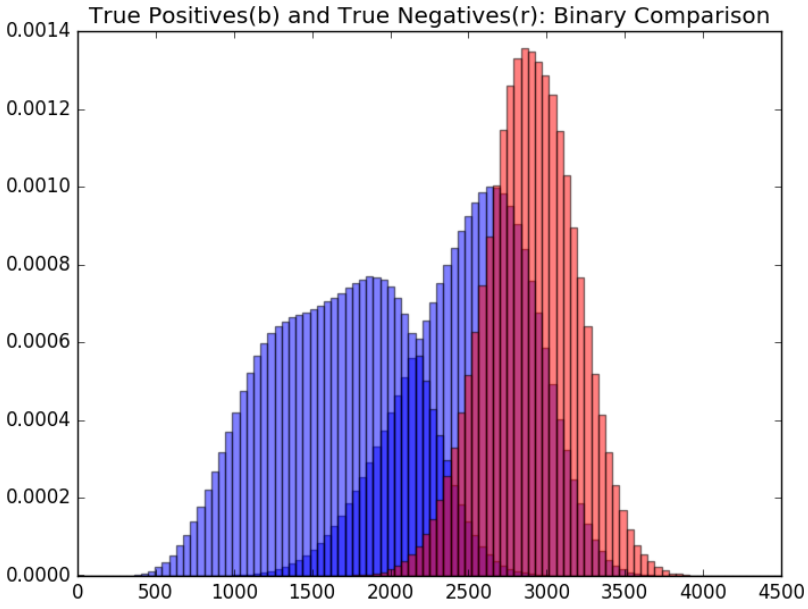
   (c) The histogram below illustrates the the distribution of digits within the training data. It is s not-uniform, but comparable.



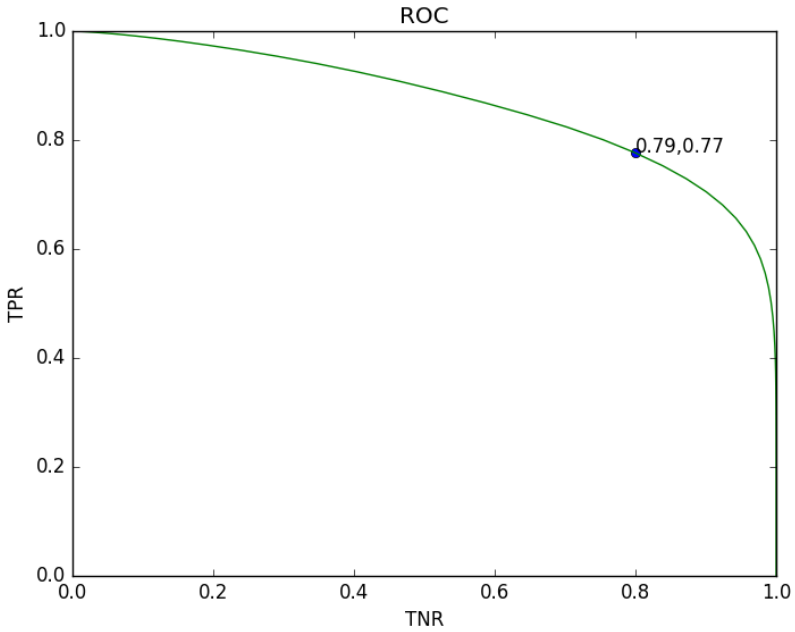
(d) 10 randomly picked digits and their closest matches are below.



(e) Below are the histograms for the genuine and impostor distances on the same set of axes.



(f) The image below exhibits the ROC curve for the genuine and imposter matches. The equal error rate is displayed at $\approx .78$. In the case of binary comparison, the error rate of a classifier that simply guesses is $\frac{1}{2}$.



(g) The KNN classifier is implmented in file *mnist/digit_recognizer.py* as function *knn*

(h) The KNN classifier is implmented in file *mnist/digit_recognizer.py* as function *cross_validate*

(i) The confusion matrix for all digits is displayed below, with the vertical axis representing *actual* values and the horizontal axis representing *predicted*.

	0	1	2	3	4	5	6	7	8	9
0	99.911	0.004	0.010	0.002	0.001	0.015	0.046	0.004	0.004	0.004
1	0.000	99.919	0.024	0.006	0.012	0.004	0.006	0.019	0.002	0.008
2	0.098	0.180	99.326	0.056	0.014	0.015	0.023	0.230	0.038	0.020
3	0.013	0.052	0.062	99.464	0.007	0.158	0.008	0.075	0.095	0.064
4	0.007	0.130	0.000	0.000	99.450	0.002	0.045	0.025	0.004	0.337
5	0.025	0.037	0.005	0.177	0.023	99.468	0.136	0.013	0.036	0.081
6	0.076	0.020	0.001	0.000	0.014	0.051	99.827	0.001	0.007	0.001
7	0.006	0.177	0.029	0.002	0.033	0.000	0.000	99.587	0.002	0.163
8	0.038	0.157	0.037	0.199	0.044	0.219	0.051	0.037	99.056	0.162
9	0.042	0.038	0.008	0.088	0.121	0.024	0.005	0.206	0.027	99.440

(j) See function *kaggle_knn* in *mnist/digit_recognizer.py*. The output csv was submitted to kaggle with the following result:

2. The Titanic Disaster

   (a) The test datasets can be found in the *./titanic/titanic_data* directory.

   (b) Implementation is completed using scipy's *LogisticRegression* class and can be found in file *./titanic/titanic.py*. Two things to note about the implementation:

   - The categoricaly features are translated to *One Hot* encoding before the classifier is parsed.
   - Features *PassengerId, Name, Ticket Fare, Cabin*, and *Embarked* are ommitted from the classifer. Through some trial and error, one can draw the conclusion that omitting these features yields a more accurate classifier, most likely because they are scarcely populated(E.g. *Cabin* has values in the test set do not exist in the training data) or contain a a high number of unique, categorical values(E.g. *Fare*). The values thus increase the dimensionality of the model while offering nothing but idiosyncracies that likely exist only in the sample training data.

   (c) the result of the classifier have been submitted to Kaggle and can be found in the *./titanic/titanic_data*.

Written Exercises

1. Based on the defintions of variance and covariance,:

$$var(X) = E[(X - E[X])^2]$$
$$cov(X) = E[(Y - E[Y])(X - E[X])$$

we have:

$$\begin{aligned}
var[X - Y] &= E[(X - Y - E[X - Y])^2] \\
&= E[(X - Y - E[X] - E[Y])^2] \\
&= E[(X - E[X]) - (Y - E[Y]))^2] \\
&= E[(X - E[X])^2 - 2E[(Y - E[Y])(X - E[X])] + E[(Y - E[Y])^2]] \\
&= var(X) + var(Y) - 2E[(Y - E[Y])(X - E[X])] \\
&= var(X) + var(Y) - 2cov(X, Y)
\end{aligned}$$

2. Letting event R be a positive test result, and event D be the occurence of a defective widget, we have...

$$\begin{aligned}
P(D) &= \frac{1}{100,000} \\
P(R|D) &= .95 \\
P(\neg R|\neg D) &= .95
\end{aligned}$$

...and therefore...

$$\begin{aligned}
P(\neg D) &= 1 - \frac{1}{100,000} = \frac{99,999}{100,000} \\
P(\neg R|D) &= 1 - P(R|D) = .05 \qquad \text{(because } R \cap \neg R = \emptyset\text{)} \\
P(R|\neg D) &= .05
\end{aligned}$$

Additionally, the probability of a positive test result is the sum of the probabilities of a defective widget testing positive and a non-defective widget testing positive; i.e.,

$$\begin{aligned} P(R) &= P(R \cap D) + P(R \cap \neg D) \\ &= P(R|D)P(D) + P(R|\neg D)P(\neg D) \\ &= (.95)\frac{1}{100,000} + (.05)\frac{99,999}{100,000} = \frac{.95}{100,000} + \frac{4999.95}{100,000} = \frac{\mathbf{5000.9}}{\mathbf{100,000}} \end{aligned}$$

(a) Given a positive test result, the probability the widget is actually defective is:

$$\begin{aligned} P(D|R) &= \frac{P(R|D)P(D)}{P(R)} \qquad \text{(bayes rule)} \\ &= \frac{\frac{.95}{100,000}}{\frac{5000.9}{100,000}} = \mathbf{0.000189966} \end{aligned}$$

(b) The probability a widget is not defective and tests positive is:

$$\begin{aligned} P(R \cap \neg D) = P(R|\neg D)P(\neg D) &= (.05)\frac{99,999}{100,000} \\ &= 0.0499995 \end{aligned}$$

The probability a widget is defective and does not test positive is:

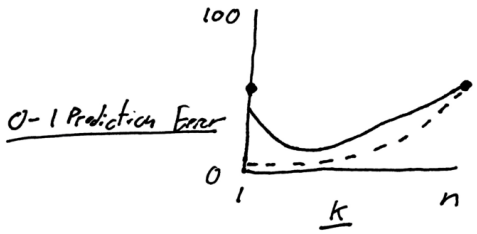
$$\begin{aligned} P(\neg R \cap D) = P(\neg R|D)P(D) &= \frac{.05}{100,000} \\ &= 0.0000005 \end{aligned}$$

Therefore 499995 non-defective widgets are thrown out and 5 defective widgets are shipped per year.

3. For KNN classification with n data points...

(a) When $k = n$, all data points are considered for any prediction, and therefore any prediction will simply yield the prior probability of the training data. As k approaches 1, fewer training points are taken in to consideration for a prediction, therfore, as $k \to n$ the mean squared prediction error decreases. When $k = 1$, the classifier considers only x_i for prediction, and thus the prediction error becomes 0.

(b) Previously, when training and testing data were the same set of points, the 0-1 prediction error was 0 for for $k = 1$, and approaches $\frac{1}{2}$ as k approaches n. When testing on the *held-out half*, the error is low to when $k = 1$ (relative to how much the 0-1 classes overlap) and gradually increases to $\frac{1}{2}$ as k approaches n. In the image below, the dotted line is training kNN with all of the training data as in (a) and the solid line is when the training data is split.



(c) If we have a data set containing n datapoints, the relative computational cost C_i, of performing cross validation, as a function of i folds can be expressed as:

$C =$ (number of folds)(data points in testing set)(distance calculation for each point in the training set)

$$C_i = i\left(\frac{n}{i}\right)\left(\frac{n(i-1)}{i}\right) = \frac{n^2(i-1)}{i}$$

Extending our conclusion from (b), candidates for k should be tested with an adjustment proportional to $\frac{i-1}{i}$ (specifically - adjust k according to the change in density that occurs when removing $\frac{1}{i}$th of the samples from the training data - see (e); one may also consider the dimensionality, of the feature vector, x).

Choosing i to be large (say 10), will mitigate the impact of density reduction, however the larger i is more computationally expensive($i \to \infty, C_i \to n^2$). As such, if performance is a concern, then $k = 3$ is a reasonable choice for at this point, $C = \frac{2}{3}n^2$. Once $k = 5$, however, $C = \frac{4}{5}n^2$, and at $k = 10$, $C = \frac{9}{10}n^2$. At this point, any sadvings in cost are negligible.

(d) If k is large, for some x it may be beneficial to weigh votes from a neighbor $N_k(x)$ using some function w that decreases as the distance d, from x increases. E.g.:

$$w_k(x, N_k(x)) = \frac{1}{1 + d(x, N_k(x))}$$

(e) As the dimension of a model increases, the *closeness* of a point to it's neighbors decreases. That is, the the size of the neighborhood about a point does not scale linearly with the part of each each input dimension needed to cover that neighborhood. If we consider the unit space in 1, 2, and 3 dimensions and create subspaces that are half of the input unit spaces for each dimension, then the respective volume is $\frac{1}{2}$, $\frac{1}{4}$, and $\frac{1}{8}$. The inverse is also true; to maintain a fixed fraction of the total space, a large interval of each dimension must be considered. Additionally, the stability of the KNN algorithm relies upon a dense training sample. For a fixed number of points in dimension N, increasing the dimension by p' requires a an increase in the number of data points on the order of to $N^{\frac{1}{p'}}$.