Face and Digit Classification

Final Project

Michael Li - 192008938
Daniel Liu - 184007283
Robert Kulesa - 185009892

CS440 Fall 2021
Professor Boularias
Rutgers University - New Brunswick
December 14, 2021

Classifier 1 - Perceptron

The Perceptron Classifier works by using a binary classifier, in which the training data is read and its features compared to determine if the input matches the attributes of its corresponding class. For every training image with feature vector f, the classifier determines the score for every class y utilizing the following:

$$score(f, y) = \sum_k f_k w_k^y$$

The max score y' for f is then determined to identify the closest matching class for f. If it is found that the best guess y' does not match the actual score y, then we adjust our weights as such:

$$w^y + = f$$

$$w^{y\prime} - = f$$

The perceptron classifier in essence classifies each image in the class with the maximum score calculated by multiplying the feature vector f by classes' weight vectors w.

Perceptron Results - 5 iterations

The accuracy of the classifier was tested by calculating the standard deviation and mean of accuracy through 5 random samples of training data per percentage level. As seen from the table following, the accuracy of the classifier generally increased the more training data that were introduced. Likewise, the standard deviation of the accuracy decreased the more data presented.

% Training Data	Mean Accuracy	Standard Deviation
10	0.607	4.573
20	0.740	4.000
30	0.753	1.007
40	0.827	1.412
50	0.800	0.000
60	0.867	0.000
70	0.813	0.000
80	0.813	0.000
90	0.840	0.000
100	0.847	0.000

Classifier 2 - Naive Bayes

The Naive Bayes classifier is based off of the Bayes Theoreum, which is defined as:

$$P(y|X) = \frac{P(X|y) * P(y)}{P(X)}$$

It functions by calculating the posterior probability $P(y|X)$ of each possible class based off of frequency distributions observed in the training set and choosing the class with the highest posterior probability as its prediction. This specific implementation of Naive Bayes used each pixel in the training images as an input, and calculated the likelihood of a each pixel being "on" or "off" given a class label (i.e. face, not face). Typically the posterior probability is calculated by multiplying the prior probability of a class with the likelihood of each feature, but since the number of features was so high the log-sum of likelihoods and prior probabilities was used instead to avoid arithmetic underflow when calculating posterior probabilities. Additionally, calculating the denominator $P(X)$ is a redundant calculation since it does not change the relative difference between likelihood scores. As a result, the posterior probability equation used by this implementation of Naive Bayes is the following:

$$P(y|X) = log(P(y)) + log(P(X|y))$$

A smoothing factor of +1 was also applied to each likelihood calculation to avoid a zero probability prediction if a given feature distribution is not represented well in the training data.

NB Results - 5 iterations

The Naive Bayes classifier was able to achieve fast classification speeds by precalculating all possible likelihoods for each feature (pixel) in a given sample and class. Thus, when the predict() method is called, the model only needs to look up the likelihoods of each feature in a sample from a pregenerated table of likelihoods. The Naive Bayes model was able to classify the face test set in 0.576 seconds (150 samples) with an accuracy of 0.893 and the digit test set in 3.699 seconds (1000 samples) with an accuracy of 0.769.

Testing accuracy of the classifier when trained on different percentages of training data was evaluated. The model randomly sampled a percentage of the training data 5 times at each stage and the means and standard deviations are displayed below. Testing accuracy increased with the amount of training data provided, with the greatest increase being observed when increasing from 10% of the training data to 20% of the training data.

% Training Data	Mean Accuracy	Standard Deviation
10	0.487	0.001
20	0.813	0.053
30	0.8	0.067
40	0.84	0.027
50	0.863	0.01
60	0.847	0.007
70	0.88	0.007
80	0.887	0.007
90	0.88	0.007
100	0.893	0.0

Classifier 3 - K Nearest Neighbor

The K-Nearest Neighbor (KNN) Classifier works by using distance functions to compute the distance of a test sample to all training samples, and classifying the test sample as whichever class the majority of the K nearest samples. In this instance, each $m \times n$ image sample is flattened into a vector with $m * n$ dimensions.

Many distance functions can be used, and each has their own advantages and disadvantages for different datasets, such as speed and memory requirements.

For the face dataset, this implementation of KNN uses the cosine distance as the distance function. Mathematically, the cosine distance is the cosine of the angle between two vectors in n-dimensional space. The cosine distance of vectors q and p can be calculated as:

$$dist\,(q,p) = 1 - \cos(\theta) = 1 - \frac{q \cdot p}{\|q\| \|p\|}$$

For the digits dataset, this implementation of KNN uses the euclidean distance as the distance function. Mathematically, the euclidean distance is the length of the line segment that connects two points in n-dimensional space. The euclidean distance of vectors q and p can be calculated as:

$$dist\,(q,p) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

The training phase of KNN only involves loading the data into the model, and all the distance computation is done during the testing phase. Therefore, the training phase has a runtime of $\mathcal{O}(1)$. The number of computations done in for each test sample scales linearly with the number of training samples n, giving the testing phase of m test samples a runtime of $\mathcal{O}(mn)$.

KNN Results - 5 iterations

Overall, KNN achieved good accuracy (~80%) for both datasets when using even just 10% of the training data. As stated before, the runtime of the test phase of KNN appears to scale linearly with the number of training samples used. However, the runtime for the digits dataset is extremely high in comparison to the runtime for the faces dataset. Future versions of this model may improve upon dimensionality reduction to lower runtime while maintaining a high level of accuracy.

% Training Data	mean(acc) - Faces	std(acc) - Faces	mean(acc) - Digits	std(acc) - Digits
10	0.8080	0.00275	0.7904	0.00147
20	0.8213	0.00098	0.8380	0.00032
30	0.8840	0.00116	0.8536	0.00059
40	0.8813	0.00244	0.8732	0.00044
50	0.9173	0.00137	0.8820	0.00025
60	0.9107	0.00053	0.8922	0.00066
70	0.9333	0.00119	0.8950	0.00040
80	0.9320	0.00078	0.9014	0.00050
90	0.9413	0.00078	0.9030	0.00024
100	0.9467	0.0	0.9060	0.0

% Training Data	mean(runtime) - Faces (seconds)	mean(runtime) - Digits (seconds)
10	0.43	71.95
20	0.84	143.94
30	1.27	216.87
40	1.69	290.61
50	2.13	363.72
60	2.56	433.94
70	2.99	510.67
80	3.47	579.12
90	3.91	651.01
100	4.45	721.92