Project: Face and Digit Classification

Dhruvil Patel, Brandon Yu

May 6, 2019

Introduction

We used the files from http://inst.eecs.berkeley.edu/cs188/sp11/projects/classification/ as base code. These files had the basic structure needed for the project only missing the classifier parts. We also added a couple of new argument options so we can display the information required by the project and implement our own choice of a classifier. As per instructions, we implemented classification, training, and tuning functions for Perceptron, Naive Bayes, and K-Nearest Neighbors. All of the options can be seen by the command `python dataClassifier.py -h`

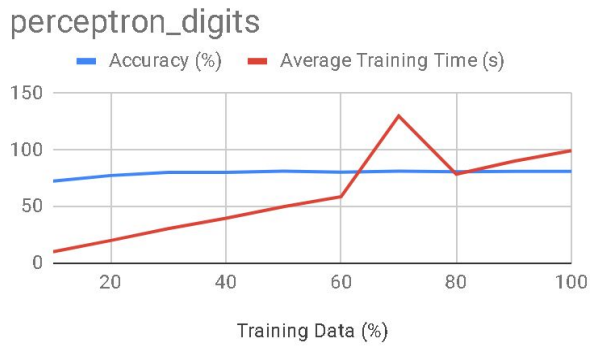
Comparison of Classification Algorithms

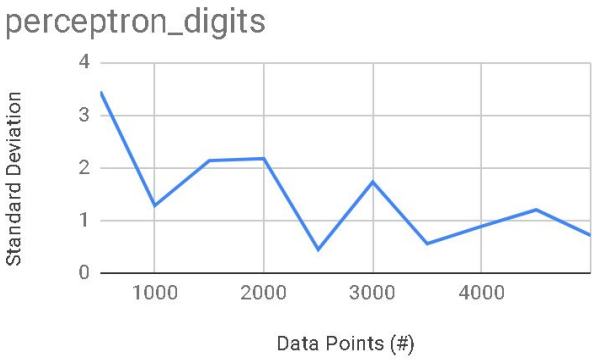
Perceptron

The Perceptron algorithm is used to develop a linear threshold function. Unlike the Naive Bayes, perceptron does not use probabilities for predictions and instead uses a linear prediction function.

Basic implementation of the perceptron was simpler since it uses a weight system. In other words, you would hold a global counter of weights for each possible label. There is one big loop for the number of iterations you would like to do. Within this loop, you would loop through the training data points. Lastly, inside this loop, you would keep a score for each possible label which is basically a counter of pixels values and update the weights if the current score is higher than the previous score (weight) of the label. It took quick long to compute since it involves many nested loops.

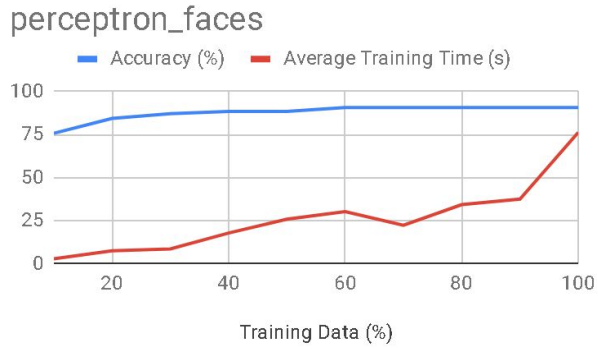
Perceptron Algorithm on Digits

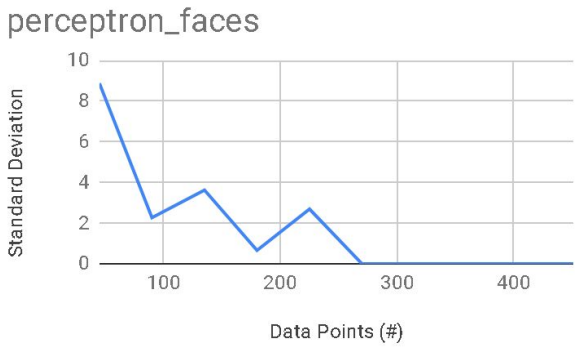




Training Data (%)	Accuracy (%)	Average Training Time (s)	Standard Deviation	Data Points (#)
10	72.45	10.281	3.446	500
20	77.425	20.196	1.284	1000
30	80.2	30.595	2.137	1500
40	80.25	39.815	2.176	2000
50	81.225	50.005	0.45	2500
60	80.45	58.74	1.731	3000
70	81.2	129.843	0.56	3500
80	80.8	78.609	0.891	4000
90	81.1	89.994	1.203	4500
100	81.15	99.373	0.719	5000

Perceptron Algorithm on Faces





Training Data	Accuracy (%)	Average Training Time	Standard	Data Points

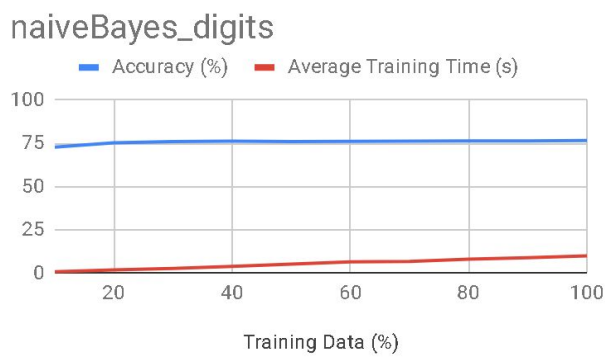
(%)		(s)	Deviation	(#)
10	75.667	3.04	8.886	45
20	84.333	7.63	2.277	90
30	87	8.68	3.631	135
40	88.333	17.901	0.667	180
50	88.333	25.949	2.694	225
60	90.667	30.353	0	270
70	90.667	22.428	0	315
80	90.667	34.366	0	360
90	90.667	37.565	0	405
100	90.667	76.153	0	451

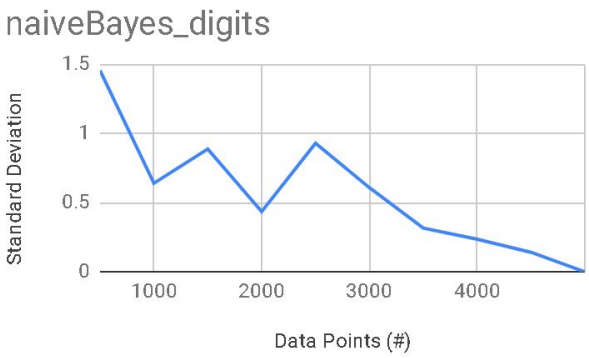
Naive Bayes Classifier

Naive Bayes is based on the Bayes' probability theorem. It's usually used for text classification, examples being spam filtration, sentimental analysis, and classifying news articles. It's also considered to be a simple, fast and effective algorithm. It's also "naive" in the way that it assumes features are independent of the occurrence of other features when predicting.

The conditional probabilities of our features given each label `y : P(Fi |Y = y)`
Further explanation of the implementation of Naive Bayes algorithm: You would loop through each of the training datasets and increment the counter for each feature we see. Also, increment priorDist for each label we see and increment the right feature in conditional probability [1] if it's white pixel and conditional probability [0] if black. When looped through the entire training data, we normalized the priorDist. Then, we can smooth our conditional probabilities by adding value such as 2 to every possible feature so we have no 0 values. Later, we normalized our conditional probabilities and that's all. We also computed log probabilities which had the same argmax as we would have by multiplying many probabilities together which often resulted in the underflow.

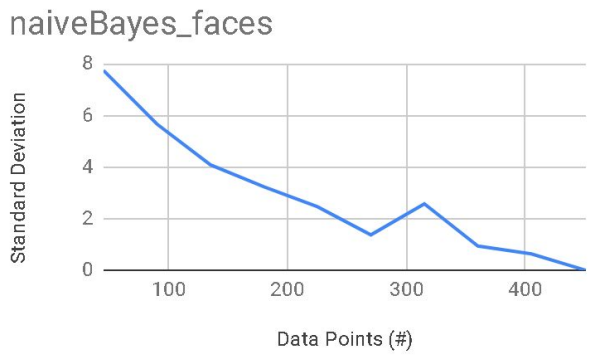
Naive Bayes Algorithm on Digits





Training Data (%)	Accuracy (%)	Average Training Time (s)	Standard Deviation	Data Points (#)
10	72.725	0.966	1.455	500
20	75.275	2.076	0.64	1000
30	76	2.829	0.887	1500
40	76.225	4.115	0.435	2000
50	76.025	5.324	0.929	2500
60	76.075	6.717	0.608	3000
70	76.2	6.896	0.316	3500
80	76.375	8.263	0.236	4000
90	76.3	9.039	0.141	4500
100	76.6	10.113	0	5000

Naive Bayes Algorithm on Face



Training Data (%)	Accuracy (%)	Average Training Time (s)	Standard Deviation	Data Points (#)
10	75	0.854	7.746	45
20	81.5	1.503	5.667	90
30	83.167	2.266	4.087	135
40	87.167	2.811	3.238	180
50	87.667	3.627	2.465	225
60	86.833	5.179	1.374	270
70	89.167	5.555	2.575	315
80	88.667	5.724	0.943	360
90	89.5	6.572	0.638	405
100	90	7.118	0	451

K-Nearest Neighbor (KNN)

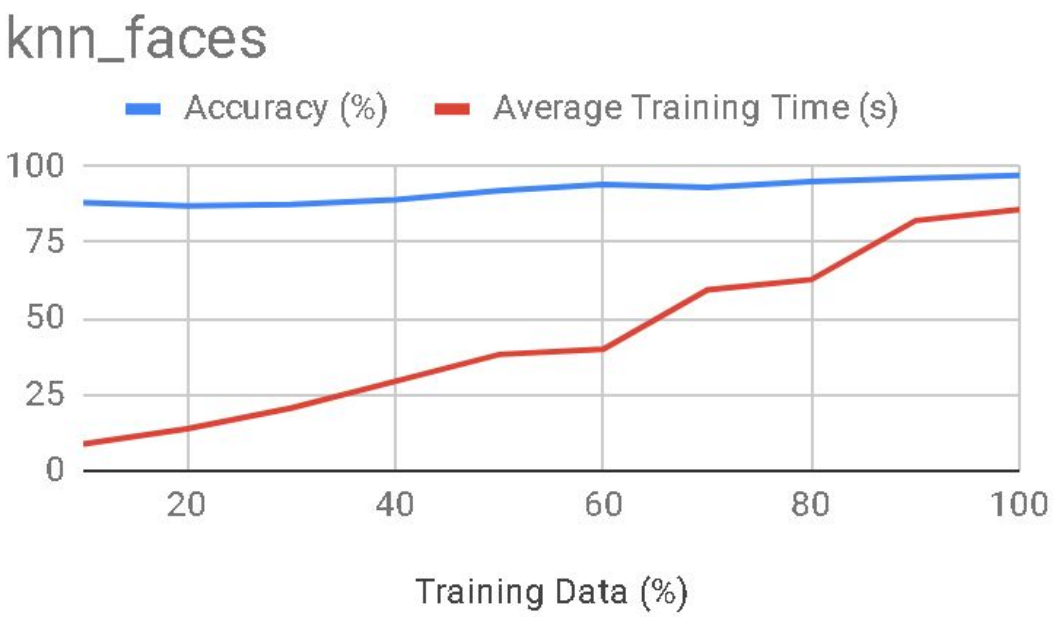
The KNN algorithm is one of the simplest classifier algorithms, and uses “lazy learning”. KNN predicts based on feature similarity. KNN should be used when data is labeled, noise-free, and data-set is small. KNN is a non-parametric learning algorithm, which means that it doesn't assume anything about the underlying data. This is an extremely useful feature since most of the real world data doesn't really follow any theoretical assumption.

As done above, we would iterate over a loop for the number of iterations. Within this loop, we would loop through the training data points. Later, we would get neighbors as given through the command prompt. To get neighbors, we calculate the Euclidean Distance of all data instances and return a subset with the smallest distance values, which would be our set of neighbors. Using this set, we can determine a prediction by allowing each neighbor to vote for their class attribute. The class with the most votes will be taken as the prediction.

KNN Algorithm on Digits

Training Data (%)	Accuracy (%)	Average Training Time (s)	Data Points (#)
10	78	7.879	500
20	87	15.84	1000
30	87	23.703	1500
40	88	32.45	2000
50	91	41.269	2500
60	92	49.006	3000
70	92	60.421	3500
80	92	67.808	4000
90	93	86.161	4500
100	94	91.768	5000

KNN Algorithm on Faces

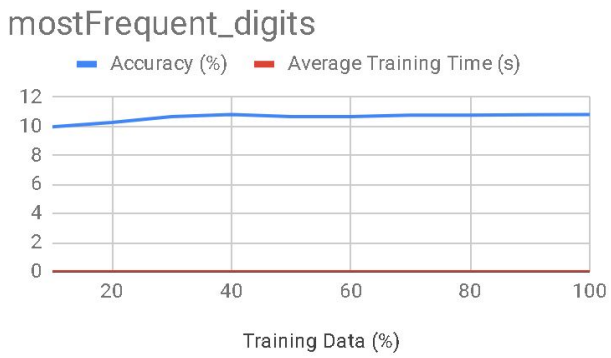


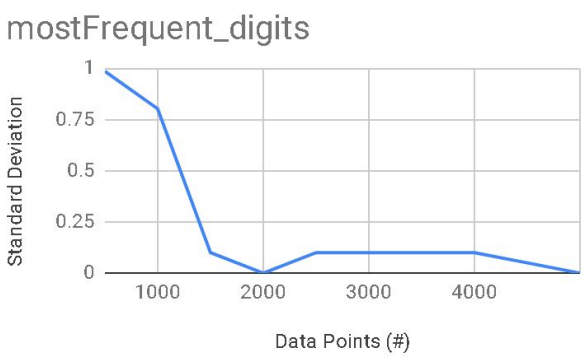
Training Data (%)	Accuracy (%)	Average Training Time (s)	Data Points (#)
10	88	8.879	45
20	87	13.84	90
30	87.4	20.703	135
40	89	29.45	180
50	92	38.269	225
60	94	40.006	270
70	93	59.421	315
80	95	62.808	360
90	96	82.161	405
100	97	85.768	451

Most Frequent Classifier

The Most Frequent Classifier is a very simple classifier. For every test instance, it looks at, it returns the label that was most frequently seen in the training data.

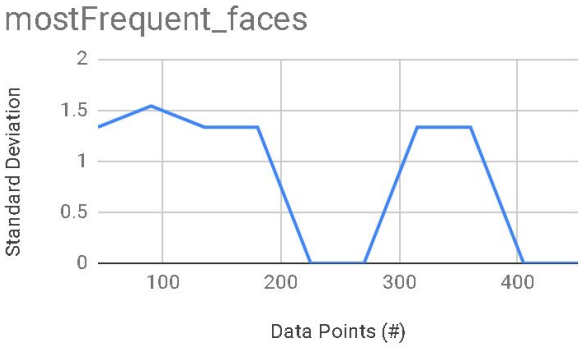
Most Frequent Classifier on Digits





Training Data (%)	Accuracy (%)	Average Training Time (s)	Standard Deviation	Data Points (#)
10	9.95	0.005	0.985	500
20	10.25	0.004	0.802	1000
30	10.65	0.004	0.1	1500
40	10.8	0.003	0	2000
50	10.65	0.006	0.1	2500
60	10.65	0.004	0.1	3000
70	10.75	0.004	0.1	3500
80	10.75	0.005	0.1	4000
90	10.775	0.008	0.05	4500
100	10.8	0.005	0	5000

Most Frequent Classifier on Faces

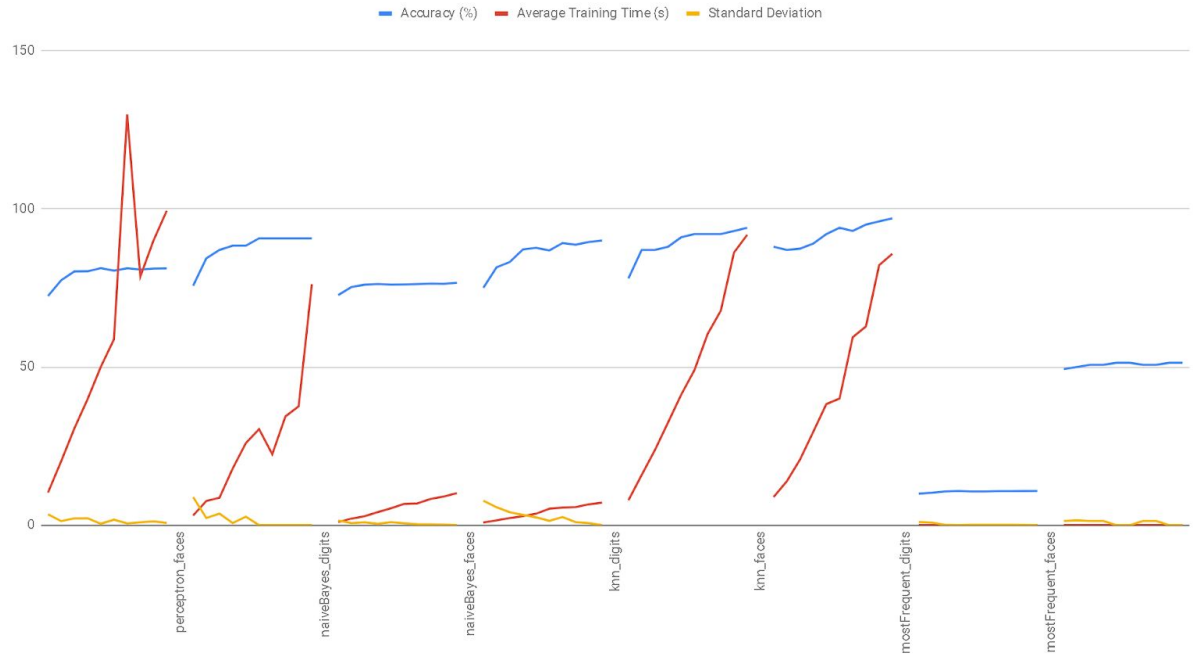


Training Data (%)	Accuracy (%)	Average Training Time (s)	Standard Deviation	Data Points (#)
10	49.333	0.004	1.333	45
20	50	0.003	1.54	90
30	50.667	0.005	1.333	135
40	50.667	0.004	1.333	180
50	51.333	0.004	0	225
60	51.333	0.004	0	270
70	50.667	0.004	1.333	315
80	50.667	0.004	1.333	360
90	51.333	0.002	0	405
100	51.333	0.004	0	451

Conclusion

Observations:

- Naive Bayes is the, overall, fastest algorithm, taking into account general accuracy. The Most Frequent Classifier is faster but very inaccurate.
- Perceptron and Naive Bayes appear to be less accurate when dealing with a large data set with Digits.
- Perceptron digits average time is mostly linear compared to Perceptron faces which seems to be somewhat exponential. Even though Perceptron takes longer its standard deviation is usually closer to 0 and provides high accuracy in the 80-90% even with a smaller subset of the training data.
- KNN is a lazy learning algorithm and therefore requires pretty much no training prior to making real-time predictions. This makes the KNN algorithm much faster than other algorithms that require heavy training.
- KNN is also faster when given a smaller training set. i.e. KNN Faces was faster than KNN Digits.


Accuracy (%)
Average Training Time (s)
Standard Deviation
150
100
50
0
perceptron_faces
naiveBayes_digits
naiveBayes_faces
knn_digits
knn_faces
mostFrequent_digits
mostFrequent_faces