KI Project C

Martijn Dekker 6013368
Egor Dmitriev 6100120
Cody Bloemhard 6231888
LaTeX

January 24, 2019

1 Exercise 1

1.a ex1-a

Images are converted into 2 dimensional arrays with size 28x28 for digits and 60x74 for faces. In these arrays every element represents one pixel of the image.

The program reads the files and will come across 3 symbols: blank space, + and # For digits a blank space indicates a white pixel, a + indicates a gray pixel and a # indicates a black pixel. For faces a blank space indicates no edge and a # indicates an edge, and the + is not used with faces. These symbols are transformed and then put into the array. A blank space becomes a 0, a + becomes a 1 and a # becomes a 2. Then in dataClassifier all 1s and 2s are changed to 1, and all 0s stay the same.

For digits there are 10 possible labels. These are the numbers 0 to 9, and they represent the numbers 0 to 9. For faces there are 2 possible labels: 1 and 0, because it's either a face or it's not. Here a 1 represents a face, and a 0 represents it's not a face.

The frequencies of all the labels divides by set and datatype.

Frequency digits	Test labels	Train labels	Validation labels
1	10.8%	11.26%	12.6%
2	10.3%	9.76%	11.6%
3	10.0%	9.86%	10.7%
4	10.7%	10.70%	11.0%
5	9.2%	8.68%	8.7%
6	9.1%	10.02%	8.7%
7	10.6%	11.00%	9.9%
8	10.3%	9.24%	8.9%
9	10.0%	9.90%	9.4%
0	9.0%	9.58%	8.5%

1.b ex1-b

Most frequent counts for every possible label how often it appears and then uses the most common label to classify an input.

Frequency faces	Test labels	Train labels	Validation labels
0	51.33%	51.88%	51.83%
1	48.67%	48.12%	48.17%

Naive bayes uses the log-joint distribution, so it also looks at how often a label is given, but unlike most frequent, it does so for every feature and normalizes the results it found.

1.c ex1-c

Most frequent does use supervised learning because it directly looks at the labels given to the training data, and uses the most common label to classify all future inputs.
Naive bayes does also use supervised learning because it looks at the labels by counting how often a label A is given to a feature B.

2 Exercise 2

2.a ex2-a

Classifier	data	Validation	testing	k
Most frequent	digits	14%	14%	x
	faces	56%	53%	x
Naive Bayes	digits	69%	55%	2
	faces	77%	75%	2

What is important to know is that for digits there are ten possible labels and for faces there are only two possible labels. This means that if you randomly guess you will have a higher chance of guessing correctly with the faces data set. This is what happens for the most frequent classifier. The classifier found the most common label in the training set and this label also appears with an above average frequency in the validation and testing set.
Therefore the scores for digits is 14% instead of 10%, which would be the case if all labels were equally common in the test and validation set. And the same is true for faces, if both labels were equally common in the test and validation set, then the scores would be 50%, instead of 56% and 53%.
The naive bayes classifier scores higher because it really looks at the image. It tries to find good features that really help distinguish between the different labels. The score for faces is higher for naive bayes because with the digits the different images will overlap, even though they are not necessarily the same label. This is because multiple digits have, for example, a curve in the top. And with faces it is easier to find a feature that distinguishes well, because there are only two possible options.

2.b ex2-b

Imagine we have 5 possible labels, these labels do not occur with the same frequency, but rather one label is quite rare. Then we can have a big training set where this label does not occur. And so the classifier will assign a probability of 0 to it. Then when we come across an input in our test data with this label, the classifier can not possibly assign the correct label. In order to prevent this, and make sure that every label has a chance to be assigned, we use laplace smoothing. This entails that we will always add a standard value to the frequency of a label. The probability used to be: frequency of a label divided by the total number of elements in the training data. But with laplace smoothing it becomes: (frequency of a label + 1) / (number of elements + number of labels).

2.c ex2-c

Classifier	data	Validation	testing	k
Naive Bayes	digits	74%	65%	0.1
	faces	85%	82%	0.05

The autotune option finds the best value for k among these [0.001, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 20, 50]. And then the classifier uses the best k value that was found for the validation and test set. This option helps to improve the score for the naive bayes classifier and will in general always give better scores. It is better for both data sets. What does stay the same is the presence of a difference between accuracy on validation and testing set. Autotune will always choose a k that performs best on the validation set, but this is not necessarily the best k for the test set.

2.d ex2-d

For naive bayes: first it gets the classifier by looking at the training set, then the validation set can be used to get a good k value and make sure we do not get a classifier that does really well on the training set but poorly on other sets. And as last it is tested on the test set to get an indication of how the classifier will perform on real life data sets. For most frequent: it gets the classifier by looking at the training set. It looks which label is the most common and then uses that as classifier. It does nothing important with the validation set. It merely looks what score it gets on there. Then it uses the test set to get an indication of how the classifier will perform on real life data sets.

2.e ex2-e

The test set is used to get an indication of the expected performance in the real world. This is because the data in the test set is not used for training. This means it is not overfit on the test set and you can see how well the classifier generalizes. This is important, for it will receive data it has not been trained on either in the real world. The validation set can be use to fine tune the classifier and rule out any form of overfitting. This happens with the naive bayes classifier when the autotune option enable. The validation test is used to get the best k value from the training data.

3 Exercise 3

3.a ex3-a

Because there are a lot of features which vary between zero and one multiplying them together can yield very small numbers. These numbers in some cases cannot be represented using float or double types.
A workaround for this is representing probabilities as log probabilities. This way their values are between -∞ and 0.
Also because of this we no longer use product, but use summation instead. This works since sum of logarithms is equivalent to the log of the product of all the probabilities.

3.b ex3-b

Conditional probability can also be written as P(A$\wedge$B) = P(A|B)P(B) . With this we can see that joint probability yields the same result as posterior probability multiplied with probability of P(B).
Using log in joint probabilities achieves same result as described earlier because sum of logarithms of the probabilities is equivalent to the log of the product of all the probabilities.

3.c ex3-c

	0.55	0.60	0.65	0.70	0.75	0.80
accuracy test	85.0%	85.0%	85.0%	85.0%	85.0%	85.0%
accuracy validation	81.0%	81.0%	81.0%	81.0%	81.0%	81.0%

Thresholding does not seem to to improve performance. It rather worsens the performance by introducing more false negatives. The posterior probabilities are all around 50% for which thresholding by the given amounts is too much. This is because features are of a low quality.//

Reproduce dataClassifier.py -c naiveBayes -d faces -k 0.1 –threshold=0.45,0.55

3.d ex3-d

The naive bayes classifier would be more biased towards classifying numbers as even numbers. Normally one would need to balance the dataset first, but this would require retraining. Since naive bayes works using frequency counting we can also correct the amount of even number occurences in the frequency table by reducing them by 50%. This would balance the dataset out.

4 Exercise 4

4.a ex4-a

The perceptron goes through the examples same way as they are ordered in trainingData. The element that is on position 0 in trainingData will be processed first. Whether or not the perceptron is better than Naive Bayes depends

Training iterations →	1	2	3	4	5	6	10	50
Validation accuracy	63%	60%	55%	55%	56%	56%	56%	56%
Testing accuracy	57%	57%	48%	54%	54%	54%	54%	54%

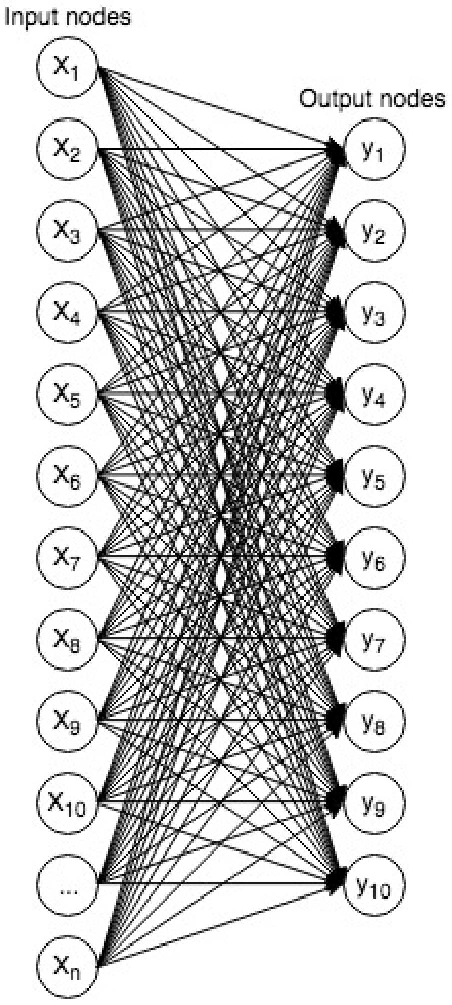
heavily on the options one might change when training on the data. As follows from the data given in section 2.1 Naive Bayes gets a 69% score for validation and a 55% score for testing with a K of 2.
If you compare this to the score of the perceptron with 5 iterations then the Naive Bayes classifier is obviously better because both the validation and testing score is higher. However, when you compare it with the perceptron with 1 or 2 training iterations, the perceptron scores are better.
The Naive Bayes validation accuracy might be higher but the testing accuracy is the important score because this will give the best indication for future scores. The testing accuracy is higher with 57% versus 55%. But once you use autotune to get a better score for Naive Bayes, it is significantly better than the best score of the perceptron with a validation accuracy of 74% and a testing accuracy of 65% for a K of 0.1. So in most cases the Naive Bayes classifier is better than the perceptron.

4.b ex4-b

Based on the scores collected in part 4.1 it is safe to stop training after 5 iterations. This is because the scores you get when training even more are not better than after 5 iterations, in fact, they are exactly the same. However, when faced with a different training set it is possible that the perceptron's score keep improving after 5 iterations.

4.c ex4-c

Figure 1:



The activation function for the perceptron is $y_j = \sum_{i=1}^{n} X_i * w_i^j$. Here every connection has a weight w_i^j where i indicates the input node and j indicates the output node.

5 Exercise 5

5.a ex5-a

Sequence weights (a) are more representable of the perceptron because these represent features that are weighted the most in classifying the given label. Because these features are mostly used to determine whether and input corresponds to the label it is fair to conclude that perceptron sees weights 'a' as the ones that represent the labels the most.

5.b ex5-b

Perceptron is capable of correctly classifying separable data. In this case digit writings can not be considered separable data. This is because a lot of digits look very similar. They also depend on handwriting and because of this a lot of different digits can look alike when written by different people.
Some white pixel position can belong to different digits. For example the top-most pixels of a digit are present in every digit.

6 Exercise 6

6..1 ex6-a

By limiting weight change so extreme outliers can not influence the weights as much. By not limiting the weight change large errors may cause weights to overshoot the minima which causes the weights converge slower. Because there is a large number of features, they may vary a lot in values which may cause such a slowdown.

6..2 ex6-b

Autotune trains the model with different values for constant C and picks the best value which achieves the best accuracy.
Autotune has used three different values for C [0.002, 0.004, 0.008]. Between those, value of 0.004 for C has given the highest accuracy on the validation set.
MIRA has given 68.0% on the validation and 62.0% on training datasets with C=0.004 whereas perceptron gives 55.0% for validation and 48.0% for training dataset. Naive bayes gives 74.0% on the validation and 65.0% on training datasets with k=0.1.
MIRA has better performance than perceptron but performs worse than best naive bayes.
MIRA performs better than perceptron because it variably limits the learning rate trying to put a margin between the boundaries which gives it an advantage at classifying unseen data points.
MIRA itself is still perceptron based and does not handle inseparable data well. Classify digits with pixels as features is an example of inseparable data and therefore causes MIRA perform worse than naive bayes.

6..3 ex6-c

The first thing to notice when comparing most relevant weights from perceptron and mira is that more dense regions on the perceptron are even denser in mira and less dense regions are less dense at MIRA. This is mostly noticable with digits like 7 and 5.
The only difference between the two algorithms is the learning rate. MIRA adds a variable learning rate which tries to add a margin between between the boundaries. This is responsible for making regions more or less dense than they are on the perceptron.

7 Exercise 7

7.a ex7-a

No, naive bayes classifier is not a suitable instrument for classifying digits. This is because it assumes the values of the features are independent of the values of any other features given a class variable. This is not true because digits aren't made of independent pixels. They are made of lines / curve segements which together form a digit. Therefore it is possible that values of features are dependent on other features. Like the pixels around it.

7.b ex7-b

Given command shows all the features which are more likely to belong to label 4 in comparison to label 2. -1 and -2 take as argument the labels that need to be compared. Since features belong to pixels, these features are drawn again as pixels/characters. The likelyness is represented by the weights. The label that has a higher weight for a feature in comparison with another, then it is more likely to represent that label.

7.c ex7-c

In the following list we describe a few features which would have a positive impact at classifying digits. Some of these are non binary or take up multiple binary inputs.

Hole count Some digits have holes. Some have multiple. For example if a digit has 2 holes, then is is quite certain that it is an 8 since only 8 has 2 holes. We can count holes by partitioning image into multiple groups of empty/black pixels which are in direct contact. If we reduce this number by once, since background doesnt count as a hole. Then we can map it into 3 binary features (has no holes, 1 hole, 2 or more). This feature is implemented in our code and has increased the performance.

Curvature/Convexity defects Almost all digits are concave except zero. One can fit a polygon around the borders of white pixels and reduce it to have fewer polygons. This way one can determine the count of convexity defects in the shape using gift wrapping algorithm. Then this can be stored in multiple binary features (Ex. 0, 1, 2, 3 convex defects). This way a number can be determined quite accurately. 0 has 0 defects, 1 has 1 defect 2 has 2 defects, 3 has 3 defects, 4 has 3/1 defects and 9 has 1 defect.

Color cross rate One can count how many times pixel color changes (white -> black) in a row or column and add 3 binary features for each row/column representing 0, 1 or 2 color changes. This is another way to determine if the input has holes. Since 9 would have highest values in top rows meanwhile 6 would have those in lower rows.

Density One could split the shape into multiple smaller squares and calculate the density/amount of white pixels in the given region. Then represent each square in as a binary feature. One could use 0 als "not dense" or "less than 50while 1 gives the opposite. Some digits have high density in their corners. These corners have different positions for every digit.

7.d ex7-d

We have implemented the first described feature. We have done this by taking each pixel not in closed list and expanding it in all directions using depth first search. Neighbours that are out of bounds or are a white pixel are not added. This has increased the accuracy on the test set from 78% to 84% which is quite an improvement and it makes sense since this is a string indicator which type of digit it is. Digit 8 can be easily classified since its the only that has 2 holes. Also numbers 1, 2, 3 cant be mistaken for other numbers since they dont have any holes.

8 Exercise 8

8.a ex8-a

The perceptron classifier for pacman has a score of 88/100 in on the validation set. Is has a score of 84/100 on the test set. This means it might be a little overfitted, but it can generalize fairly well.

8.b ex8-b

StopAgent: The stopagent just wants to stand still. This is very easy for the perceptron, it will increase the weights for stopping, and lower the weights of all the other things. The perceptron does not need any aditional features, for it can achieve a 100% score without any features.
FoodAgent: The foodagent just wants to eat food. The distance to the closest food of the succesor state looks like a good feature. A high value on this feature means that the agent is far away from food, and a low value means it is close to food. The agents can now try to optimise for a low value, meaning it will go towards the food.
SuicideAgent: The suicideagent just wants to crash into a ghost to die. The distance to the closest ghost can be a helpful feature. It can again optimise for a low value and learn to move into the ghost. There is a better chance of getting killed if the agent is in a area with multiple ghosts. By adding the feature of the sum of the distances to the ghosts could help the agent to have a bias towards more crowded places.
ContestAgent: The contestagent wants to play the game to the best of it's ability. The distance to the closest enemy is again important, but this time to avoid the ghosts. The distance to the closest food is also important again, because it really needs to eat the food to win. The distance to the closest power pallet could be a good feature here. We want to stimulate pacman to eat those, as it makes him invincible for a short time. That can help with winning the game.

8.c ex8-c

StopAgent: It does not matter what feature you give it or, if you give it a feature at all. I gave it the score of the game as feature.
FoodAgent: The distance to the nearest food did not work out well. Normalizing the distance worked out. Our function for doing so is norm(x) = 1.0 / (x + 1). This is very simple and guards against division through zero where x > 0. I rare ocations, when a piece of food is alone in an area, eating it willincrease the distance to the next food significantly. An attempt to fix this was by setting the feature's value to 1 if the succesor state has less food than the current state. By doing so the feature always has a higher or equal value for an action where pacman eats food verus an action where pacman does not eat. However, this change had a small negative effect on the performance, and it was removed.
SuicideAgent: Here we took the distance to the nearest ghost from pacman. The raw distance did not work again, so it is normalized again. This was enough to get a 90% test score. The other feature, the sum of the distances to all the ghosts was also normalized. It did help a little bit, the test score went up to 93%. Interestingly, the validation score was lower than the test score with 87% at first and only gained 0.6%.
ContestAgent: The feature of the normalized distance to the nearest food had a very positive effect on the score of the agent, as expected. If the agent does not eat, he can not win. The normalized distance to the nearest ghost had a negative influence on the score. This is because it might get in the way of some other features that were added. The same goes with the normalized distance to the closest power pallet, it decreased the score. A feature that did have a big positive effect was the normalized absolute difference of the succesor's state score and the current state's score. This can be because it indicates many things, although indirectly. For example, eating food yields a higher value than not eating food. Getting hit by a ghost yields lower value than not getting hit. Eating a power pallet yields a higher score than not eating one. All the good things here lead to a higher value. More importantly than if it leads to a high or low value, all the good things lead the value into the same direction. This feature is a strong indicator if the succesor state is a good one. The perceptron had trouble with all this information in seperate features, but combining seems the way to go. There is another small feature, it's value is 1 if the succesor state is winning, 0 if it is losing, and else it is 0.5. This has a very small but positive effect on the score. The idea is to advice the agent against really stupid moves like walking into a ghost or walking past the last piece of food.

9 Exercise 9

Note: Ex8 was made with all the agents having it's own features. However, to check what agent is chosen, i made a simple hack in the code. I works with the command
python dataClassifier.py -c perceptron -d pacman -f -g ContestAgent -t 1000 -s 1000
and variants(for different agents). However it does not work with the auto-grader. I have commented out the real method and placed a all-in-one feature function. It adds features and every agent uses those. It works and gives 4/4 in

Q1	Q2	Q3	Q4	Q5	Q6	Total
4/4	1/1	6/6	6/6	4/4	4/4	25/25

the autograder. If you want to test it the way i made Ex8, got to enhancedPa-manFeatures and follow instructions in the comments.