Predicting price and outfit properties of multiple pieces of clothing using DeepProbLog

Jinfu Chen, Enrique Dehaerne

KU Leuven

Abstract

We present `budget` and `outfit`, two fashion applications powered by DeepProbLog [1]. The `budget` application predicts the price of a clothing item in an image and uses this to calculate the total cost of a collection of given images. `Outfit` uses domain knowledge about clothing items to reason about whether or not an outfit consisting of a given collection of clothing items, based on images provided of them, has certain properties. DeepProbLog combines neural networks and logical reasoning which allows both applications to perform better than baseline convolutional neural networks in terms of accuracy and F1 score. This can be attributed to the larger set of classes the baselines must be able to classify, the ability for DeepProbLog to handle uncertainty from neural network predictions and the lack of logical reasoning abilities of the baselines. Further evaluation into the distance between the predicted class value and the label, or how far off the prediction was, gave a slight, unexpected edge to the baselines. The Fashion-MNIST dataset [2] was used for the evaluation of both `budget` and `outfit`. To test the real-world practicality of text of `outfit`, the program was also evaluated using the more realistic and complex images from the DeepFashion dataset [3]. The results showed that there is a limit to DeepProbLogs abilities to handle uncertainty since the baseline was able to outperform `outfit`.

1. Introduction

In this report we present two fashion applications powered by DeepProbLog, `budget` and `outfit`. In the next section the general project design of both applications is explained. This includes the datasets used, the structure of the neural networks implemented, as well as the means of evaluation. The first application, `budget`, calculates the total price of a given collection of fashion items. `Budget` is detailed and evaluated in section 3. The second application, `outfit`, uses domain knowledge to predict if given pieces of clothing are or are not appropriate for rain, formal, and constitute a full-outfit. Section 4 explains and evaluates the `Outfit` program. Finally, in section 5 possible extensions to both programs are discussed.

2. Project design choices

2.1. Datasets

Two datasets were used for evaluation purposes in this project. The majority of evaluation used the Fashion-MNIST [2] dataset. Fashion- MNIST consists of $(28, 28)$ grayscale images of fashion items with ten different labels. As the name implies, this dataset is meant to be usable as a drop-in replacement of the original MNIST dataset [4]. A notable difference is that the images in Fashion- MNIST are more complex and thus more difficult to learn when compared to MNIST images.

The other dataset used is DeepFashion [3] which consists of full-colour images of clothing pieces on their own or being worn by models. Compared to fashion- MNIST, DeepFashion has 50 category labels. In the advanced version of the `outfit` program where DeepFashion images are used, these 50 category labels are consolidated to seven different labels for the purpose of simplicity. For example, the DeepFashion *anorak, bomber, blazer, peacoat, poncho* and *coat* labels all are considered *coats* in the advanced `budget` application. The DeepFashion dataset is used for evaluation purposes in 4.3.

2.2. Neural networks

Complementing the two datasets, there is also two different neural networks. A basic convolutional neural network was used for classifiying Fashion- MNIST images. This network has the same structure as the MNIST network used in the DeepProbLog MNIST addition examples described in [1].

Classifying the categories of DeepFashion dataset is much more difficult since the images are larger, have three channels for full-colour and are in general far more complex. For this task ResNet50 [5] (pretrained on ImageNet [6]) was used. The last layer of ResNet50 was replaced to output the seven labels used in `outfit`.

For both neural networks, hyperparameter tuning was only performed where it was absolutely necessary. The reason for this is that maximizing network performance was not a top priority in this project. Instead, the priority was to keep hyperparamters as consistent as possible between the different applications (that use the same datasets) to make them more comparable for evaluation purposes.

2.3. Evaluation baselines

A key advantage that DeepProbLog programs have over the baselines is that they will input two images separately and reason about the results. This suggests that DeepProbLog problems can be correct about one or more data examples, but a wrong classification of one of the images will cause the total classification to be wrong. Metrics like accuracy and F1, the main metrics used during evaluation in this report, only care about the final classification. Our hypothesis was that the distance between wrongly predicted classes and their true labels would be smaller for DeepProbLog programs compared to their baselines. This is because having a part of the problem correct reduces the set of possible classes that will be classified if another part of the problem is noisy. Distance metrics are proposed and further explained in the evaluation subsections below to measure how far off the predicted class is from the label.

3. Budget

The `budget` application predicts the total cost of a collection of clothing pieces represented by images. In the first subsection, the DeepProbLog program of this application is explained. In the subsequent two subsections, `budget` is evaluated when given two or three images respectively.

3.1. Budget program

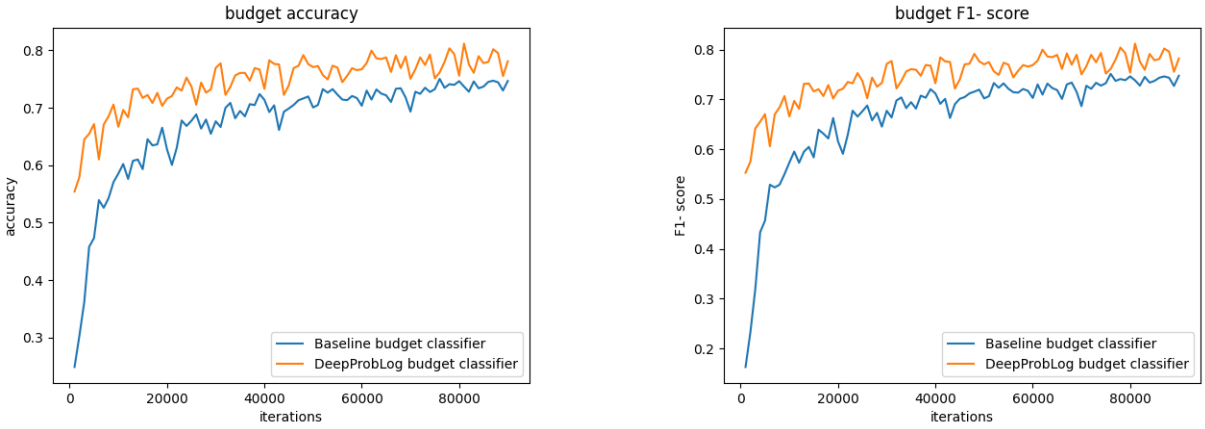
A part of the `budget` DeepProbLog program is shown in Appendix A. The program maps given clothing images to categories of clothing using the neural predicate *category*. These are then subsequently mapped to an average price for that category by deterministic predicates. The prices of each clothing item are then summed together to predict the *TotalPrice* of all given clothing.

This program is similar to the DeepProbLog MNIST addition examples described in the DeepProbLog paper [1]. A notable difference is the data used. Fashion- MNIST images are more complex than MNIST images and as a result the uncertainty from the neural predicate is greater for `budget`. Another notable difference is the mapping of values for labels. Value mapping is implicit in MNIST, each label denotes a digit of the same value. However, in Fashion-MNIST this is not the case. In a real-world application, a straight forward implementation of the price mapping would be the average price of clothing items of that category. Due to a lack of pricing data, the implementations of `budget` used in the evaluations of subsections 3.2 and 3.3 (available in Appendix B) were chosen without using any domain pricing knowledge. In the chosen price mapping the distances between subsequent labels are not all the same as they are in MNIST. A number of labels are mapped to the same price which is also different to the MNIST value mapping where all labels have unique values. This was done to further differentiate `budget` from the DeepProbLog MNIST addition examples.

3.2. Two clothing pieces budget evaluation

As seen in figure 1, the accuracy and F1- score of DeepProbLog is consistently higher than the baseline neural network. As was concluded in teh DeepProbLog paper [1] for the MNIST addition example, this improved performance is largely due to the fact that the baseline neural network must be able to classify far more possible classes than the neural network used in the DeepProbLog program. For 30,000 training examples, the MNIST addition program had a lower difference in accuracy with its baseline than `budget` did (see table 1). This is most likely due to the Fashion- MNIST data being harder to classify and DeepProbLog being able to handle this greater degree of

uncertainty better. `Budget` has fewer possible outcomes (17) when compared to MNIST addition (19) due to the price mapping. This supports the argument that the larger positive difference in accuracy between DeepProbLog and the baseline for `budget` comes from the more complex data used. Additionally, figure 1 shows that the differences between the baseline and DeepProbLog program gets smaller as the number of iterations increase. This suggests that the DeepProbLog program learns faster and needs less data than its baseline counterpart. This was also the case for the MNIST addition examples in [1].



(a) Budget accuracy plot

(b) Budget F1- score plot

Figure 1: Budget graphs

Accuracy difference for MNIST addition	3.57%
Accuracy difference for `budget`	5.08%

Table 1: Average accuracy differences between `budget` and MNIST addition DeepProbLog programs and their respective baselines. For `budget`, this average is calculated from the accuracy values of the last 40,000 iterations.

As mentioned in section 2.3, it was hypothesised that the value of the wrongly classified test examples would on average be closer to the value of the true label for `budget` when compared to its baseline. The reasoning is that if first data example is correctly classified and valued, a worst case classification of the second image will still result in a smaller distance to the correct overall label than a worst case classification of both data examples being wrongly classified. The worst case scenario for the baseline, which does not have the luxury of logical reasoning to be partially correct, is equal to the worst case scenario for both images being classified. This is why we hypothesised that the distance from the correct label for wrongly classified examples would be larger when compared to the DeepProbLog program. The absolute difference, |actual price-predicted price|, was the metric used to measure distance for `budget`. However, the results shown in table 2 refute this hypothesis as the baselines have slightly better average distances than their DeepProbLog counterparts. This will be discussed more in the next subsection.

3.3. Three clothing pieces budget evaluation

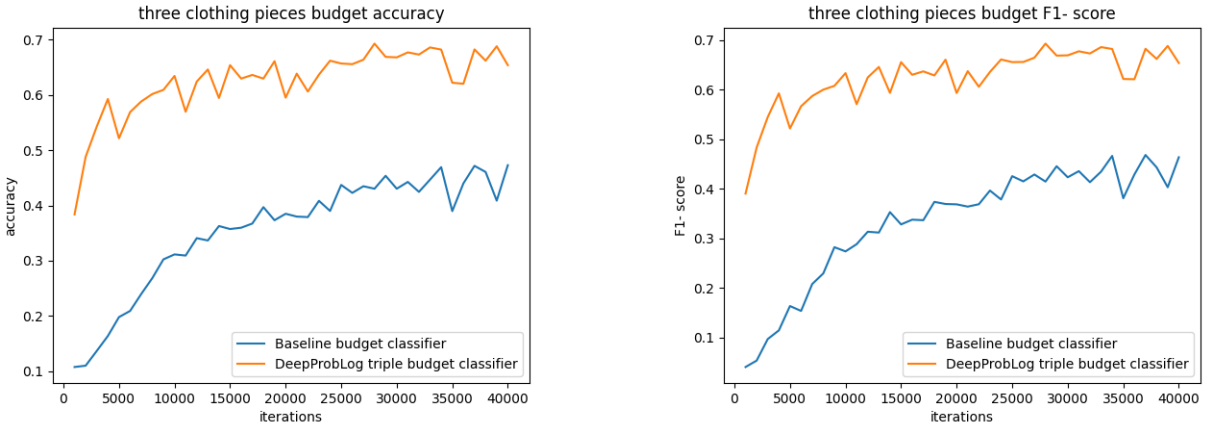
When given three clothing pieces instead of two, the differences in results between the baseline and the DeepProbLog program are amplified. This is shown in figure 2. The magnitude of the results are lower for both. For the DeepProbLog program, this comes from the added uncertainty from classifying an extra image. For the baseline this comes from a greatly enlarged number of possible classifications (from 17 in two images case to 27 for three images) as well as the added complexity of having three images as an input. Although it was not evaluated, it is

Average distance 2 images baseline	20.70
Average distance 2 images DPL	20.90
Average distance 3 images baseline	19.7
Average distance 3 images DPL	21.7

Table 2: Average distances between wrongly predicted class values and true label values for `budget` and its baseline

reasonable to assume that these trends of lower overall results with greater differences between the baseline and DeepProbLog will continue as more images are given.

In table 2 it is shown that the distance between wrongly predicted classes and their true labels increased for the DeepProbLog program while it decreased for the baseline. As discussed in the previous subsection, this refutes our hypothesis of the average distance being smaller for the DeepProbLog problem because of it being able to solve problems partially which reduces the worst-case distance. The evidence shows that the average-case distance of the neural network is slightly better than that of `budget`. The reason for this remains to be studied more closely.



(a) Three clothing pieces budget accuracy plot (b) Three clothing budget F1- score plot

Figure 2: Three clothing pieces budget graphs

4. Outfit

`Outfit` uses domain knowledge about clothing items to reason whether or not an outfit, consisting of a given collection of clothing items, has certain properties. In the first subsection, the `outfit` DeepProbLog program is explained. In the two subsections thereafter, `outfit` will be evaluated using two different datasets Fashion- MNIST and DeepFashion respectively. Note that classes of both datasets were grouped together to get 7 different categories of clothing for both. Both evaluations are performed using a collection of exactly two images.

4.1. Outfit program

Given a list of images, `outfit` uses the neural predicate to predict the category of clothing items in the list of images one at a time. Appendix A shows the partial DeepProbLog program not including the implementation of the domain knowledge rules (see Appendix B). Those categories are then checked on three rules in the `outfit` program each returning a true or false value (represented by 0 and 1 respectively). The rules check for the following three properties:

1. The given clothing pieces are suited for rain.
2. The given clothing pieces are considered formal.
3. The given clothing pieces form a complete outfit.

Whether the given clothing pieces belong to one or more characteristics is determined using domain knowledge. For example, a complete outfit must have a top and a bottom to be considered a complete outfit.
The baseline model outputs the same information, but it will be represented using labels $0-7$ (the decimal number of the binary representation of the three properties. For example, class 7 corresponds with 111).The baseline has none of the prior domain knowledge that is encoded in the DeepProbLog program.

4.2. Fashion-MNIST outfit evaluation

Figure 3 give the result of the accuracy and F1- score of the DeepProbLog program and the baseline. DeepProbLog outperforms the baseline consistently after the first 5000 iterations. As discussed in section 3.2, this is because DeepProbLog classifies one image at a time and is able to deal with uncertainty of the classifications of Fashion-MNIST images. Meanwhile, the baseline has to match two images with the corresponding label simultaneously.

Average distance baseline	1.278
Average distance DPL	1.261

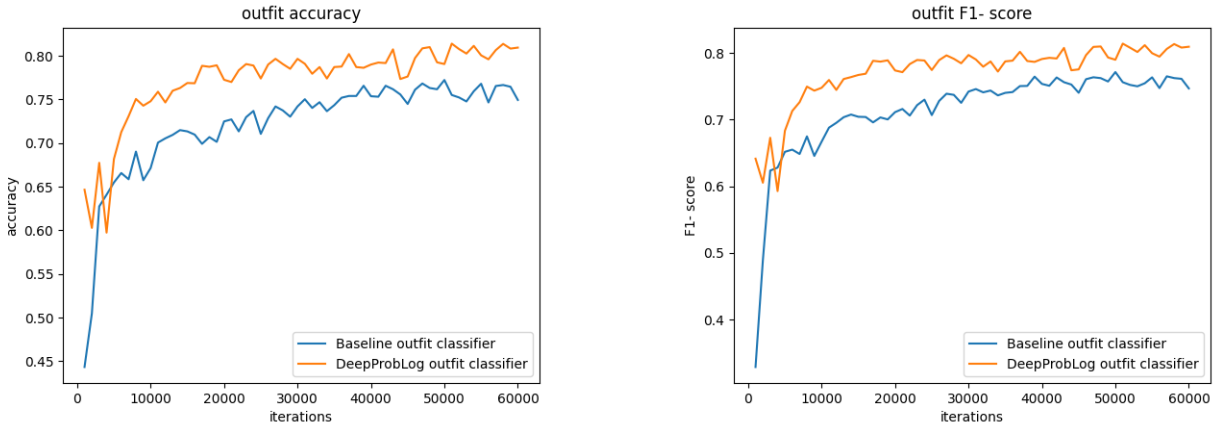
Table 3: Average distances between wrongly predicted class values and true label values for `outfit` and its baseline.

Compared to the `budget` program, `outfit`'s accuracy difference with its baseline is lower but still strong, as shown in table 4. The reason is that the baseline for `outfit` only has 8 possible classes to predict. Meanwhile, the baseline in `budget` had a choice of 19 possible classes. However, the DeepProbLog neural predicate of `outift` must classify between 7 different classes, only one less than the number the baseline must be able to classify. Therefore it is most likely that the logical reasoning with domain knowledge plays an important role in the superiority of `outfit` over its baseline to justify the relatively large positive difference even in the face of similar number of possible classifications.

To measure the distance between the predicted classes and the true labels, a hamming distance of the binary representation of the three properties listed in the previous subsection was used. The interpretation of this metric is that a hamming distance of one means that the predicted class predicts two out of the three properties correctly. The results shown in table 3 seem to verify our hypothesis discussed in section 2.3 but not certainly not significantly so. Again, this phenomenon must be study more closely to understand why this is.

Average accuracy difference for MNIST addition	3.57%
Average accuracy difference for `outfit`	4.00%
Average accuracy difference for `budget`	5.08%

Table 4: Average accuracy differences between the baseline and DeepProbLog programs



(a) Outfit accuracy plot

(b) Outfit F1- score plot

Figure 3: Outfit graphs

4.3. DeepFashion outfit evaluation

The `outfit` program given DeepFashion dataset images produced poor results. As can be seen in figure 5, the baseline performs considerably better than the DeepProbLog program. The reasoning for this is not obvious but it is believed to be due to the difficulty of classifying the complex images of the DeepFashion dataset. While in the previous evaluations, the DeepProbLog programs were able to classify images more accurately and handle the uncertainty of these predictions well, it seems that the level of uncertainty in the task of classifying realistic fashion images is too great. Even MMFashion[7], creators of DeepFashion, had relatively poor results based on recall when classifying DeepFashion image categories on neural networks they built also using ResNet50 as feature extractor. A possible explanation for why the baseline performed better than `outfit` could be that it detected spurious correlations between the images it receives as inputs. This ultimately results in `outfit` having less desirable predictions. Another possible explanation could be that the knowledge rules of `outfit` worked against itself under extreme uncertainty and the lack of these rules allowed for more flexibility for the baseline. It is expected that if the computer vision classification problem can be solved sufficiently well (the exact specifications on how good is

good enough here is not known), `outfit` will outperform the baseline. To accomplish this, sophisticated neural network training techniques and more resources to train the models on more data are needed.

Average accuracy of baseline	41.92%
Average accuracy of `outfit`	30.65%

Table 5: Average accuracy of ResNet50 and `outfit` for DeepFashion dataset

5. Extension possibilities

A possible upgrade to `budget` is using learnable probabilistic parameters ($t(p)$) DeepProbLog offers to reduce the number of classes the neural predicate in the program has to learn. From section 3.2 it was shown that having to learn more classes makes it harder for a neural net to predict any class exactly. Grouping similar classes together, such as *pullovers* and *coats* in the case of Fashion- MNIST or *coats* and *anoraks* in DeepFashion, should increase accuracy (see the last listing in Appendix B to see an example). Using the frequency of how often the sub-elements appear in the dataset, the DeepProbLog program can then pick the different prices for elements within these groups with an appropriate probability when the group is predicted by the neural predicate. This could be especially useful in minimising distance if the prices of items in these groupings are similar. With extremely adept neural predicates, the distribution of sub-categories, such as whether or not a t-shirt is of a luxury brand or not, can be modelled using these probabilistic parameters instead of having to learn to recognize this new class.

`Outfit` could also be further expanded, by giving more information output. Adding an extra output like 'good for warm weather' will likely further expand the gap between DeepProbLog and the baseline. The baseline will have to learn to classify more outputs while `outfit` keeps the same neural predicate and simply adds more domain knowledge rules to its vocabulary.

6. Conclusion

As seen in sections 3 and 4, DeepProbLog fashion applications `budget` and `outfit` outperform baselines when using simplified data. However, section 4.3 shows that, using conventional resources and techniques, these applications are not applicable for use with more realistic and complex data. Still, the integration of both neural and symbolic learning that DeepProbLog offers allows for simpler, better neural networks and easier extensibility. Because of these reasons, DeepProbLog is sure to be useful in many domains for various applications. The time we spent on different activities for this project in man-hours are as follows: 10h - readings, 60h - design, 140h - programming, 20 - report + evaluation results.

References

[1] R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester, L. D. Raedt, Deepproblog: Neural probabilistic logic programming, CoRR abs/1907.08194 (2019).

[2] H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[3] Z. Liu, P. Luo, S. Qiu, X. Wang, X. Tang, Deepfashion: Powering robust clothes recognition and retrieval with rich annotations, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[4] Y. LeCun, C. Cortes, MNIST handwritten digit database (2010).

[5] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, CoRR abs/1512.03385 (2015).

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: 2009 IEEE conference on computer vision and pattern recognition, Ieee, pp. 248–255.

[7] X. Liu, J. Li, J. Wang, Z. Liu, Mmfashion: An open-source toolbox for visual fashion analysis, 2020.

Appendix A. General programs

```
nn(fashion_mnist_net,[X], O, [0,1,2,3,4,5,6,7,8,9]) :: category(X,O).

totalPrice([],0).
totalPrice([I|Is], TotalPrice) :-
    category(I,C),
    price(C,P),
    totalPrice(Is,PartialPrice),
    TotalPrice is P + PartialPrice.
```

Listing 1: General `Budget` DeepProbLog program (not complete)

```
nn(fashion_mnist_net,[X], O, [0,1,2,3,4,5,6]) :: category(X,O).

getCategories([I],[C]) :- category(I,C).
getCategories([I|Is],[C|Cs]) :-
    category(I,C),
    getCategories(Is,Cs).

appropriateWardrobe(Images, Rain, Formal, Full) :-
    getCategories(Images,Categories)
    goodForRain(Categories,Rain),
    formal(Categories,Formal),
    fullOutfit(Categories,Full).
```

Listing 2: General `Outfit` DeepProbLog program (not complete)

Appendix B. DeepProbLog evaluation implementations

```
nn(fashion_mnist_net,[X], O, [0,1,2,3,4,5,6,7,8,9]) :: category(X,O).
% Tshirt
price(0,10).
%Trouser
price(1,25).
%Pullover
price(2,20).
%Dress
price(3,25).
%Coat
price(4,50).
%Sandal
price(5,40).
%Shirt
price(6,20).
%Sneaker
price(7,30).
%Bag
price(8,60).
%Ankleboot
price(9,20).

totalPrice(I1, I2, TP) :-
    category(I1,C1),
    category(I2,C2),
    price(C1,P1),
    price(C2,P2),
    TP is P1 + P2.
```

```
nn(fashion_mnist_net,[X], O, [0,1,2,3,4,5,6,7,8,9]) :: category(X,O).
%uses same price predicates as two piece budget program
totalPrice(I1, I2, I3, TP) :-
    category(I1,C1),
    category(I2,C2),
    category(I3,C3),
    price(C1,P1),
    price(C2,P2),
    price(C3,P3),
    PartialPrice is P1 + P2,
    TP is PartialPrice + P3.
```

Listing 3: Three pieces `budget` DeepProbLog program

```
nn(fashion_mnist_net,[X], O, [0,1,2,3,4,5,6]) :: category(X,O).

%Tshirt = 0
%Trouser = 1
%Pullover = 2
%Dress = 3
%Coat = 4
%Shirt = 5
%Bag = 6

% Wardrobe must include a coat or pullover.
goodForRain(Pieces,1) :- memberchk(2,Pieces).
goodForRain(Pieces,1) :- memberchk(4,Pieces).
goodForRain(Pieces,0) :- \+goodForRain(Pieces,1).

% Wardrobe must not include a tshirt or a dress
formal(Pieces,0) :- memberchk(0,Pieces).
formal(Pieces,0) :- memberchk(3,Pieces).
formal(Pieces,1) :- \+formal(Pieces,0).

% Whether the pieces in the given wardrobe make a full outfit.
fullOutfit(Pieces,1) :- hasTop(Pieces), hasBottoms(Pieces).
fullOutfit(Pieces,0) :- \+fullOutfit(Pieces,1).

hasTop(Pieces) :- memberchk(0,Pieces).
hasTop(Pieces) :- memberchk(2,Pieces).
hasTop(Pieces) :- memberchk(5,Pieces).
hasTop(Pieces) :-  memberchk(4,Pieces).
hasBottoms(Pieces) :- memberchk(1,Pieces).
hasBottoms(Pieces) :- memberchk(3,Pieces).

appropriateWardrobe(I1, I2, Rain, Formal, Full) :-
    category(I1,C1),
    category(I2,C2),
    goodForRain([C1,C2],Rain),
    formal([C1,C2],Formal),
    fullOutfit([C1,C2],Full).
```

Listing 4: `Outfit` DeepProbLog program

```
nn(fashion_df_net,[X], O, [0,1,2,3,4,5,6]) :: category(X,O).

%shirt alike 0
%pullover alike 1
%short 2
%trousers casual 3
%trousers normal 4
%skirt + dress 5
%coat alike 6

% Wardrobe must include a coat or pullover alike.
goodForRain(Pieces,1) :- memberchk(6,Pieces).
goodForRain(Pieces,1) :- memberchk(1,Pieces).
goodForRain(Pieces,0) :- \+goodForRain(Pieces,1).

% Wardrobe must not include casual trousers or a pullover alike.
formal(Pieces,0) :- memberchk(3,Pieces).
formal(Pieces,0) :- memberchk(1,Pieces).
formal(Pieces,1) :- \+formal(Pieces,0).

% Whether the pieces in the given wardrobe make a full outfit.
fullOutfit(Pieces,1) :- hasTop(Pieces), hasBottoms(Pieces).
fullOutfit(Pieces,0) :- \+fullOutfit(Pieces,1).

hasTop(Pieces) :- memberchk(0,Pieces).
hasTop(Pieces) :- memberchk(1,Pieces).
hasTop(Pieces) :- memberchk(6,Pieces).
hasBottoms(Pieces) :- memberchk(2,Pieces).
hasBottoms(Pieces) :- memberchk(3,Pieces).
hasBottoms(Pieces) :- memberchk(4,Pieces).
hasBottoms(Pieces) :- memberchk(5,Pieces).

appropriateWardrobe(I1, I2, Rain, Formal, Full) :-
    category(I1,C1),
    category(I2,C2),
    goodForRain([C1,C2],Rain),
    formal([C1,C2],Formal),
    fullOutfit([C1,C2],Full).
```

Listing 5: `Advanced outfit` DeepProbLog program

```
nn(advanced_budget_net,[X], O, [0,1,2,3,4]) :: category(X,O).
%%%%%%%%%%%%%%% FashionMNIST
% Shoes : [5,7,9]
% Tops : [0,6]
% Overtop : [2,4]
% Bag : [8]
% Pants : [1]
t(0.333) :: price(0,40); t(0.333) :: price(0,30); t(0.333) :: price(0,20).
t(0.5) :: price(1,10); t(0.5) :: price(1,2).
t(0.5) :: price(2,20); t(0.5) :: price(2,50).
price(3,60).
price(4,25).

totalPrice(I1, I2, TP) :-
    category(I1,C1),
    category(I2,C2),
    price(C1,P1),
    price(C2,P2),
    TP is P1 + P2.
```

Listing 6: `Advanced Budget` DeepProbLog program