programming3

November 25, 2023

1 1. Association Rule Generation

Parts a, b, and c

[1]:
```python
import pandas as pd

from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import fpgrowth, association_rules

DATASET_FILE = 'Grocery_Items_53.csv'

df_csv= pd.read_csv(DATASET_FILE)
dataset = []

# drop NaN values
for _, x in df_csv.iterrows():
    dataset.append(list(x.dropna()))


def get_association_rules(dataset, min_support, min_confidence_threshold):
    # source: http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/
 ↪association_rules/
    te = TransactionEncoder()
    te_ary = te.fit_transform(dataset)
    df =  pd.DataFrame(te_ary, columns=te.columns_)
    frequent_itemsets = fpgrowth(df, min_support=min_support, use_colnames=True)
    rules = association_rules(frequent_itemsets,
                              min_threshold=min_confidence_threshold,
                              metric="confidence")
    return rules

get_association_rules(dataset, 0.01, 0.1)
```

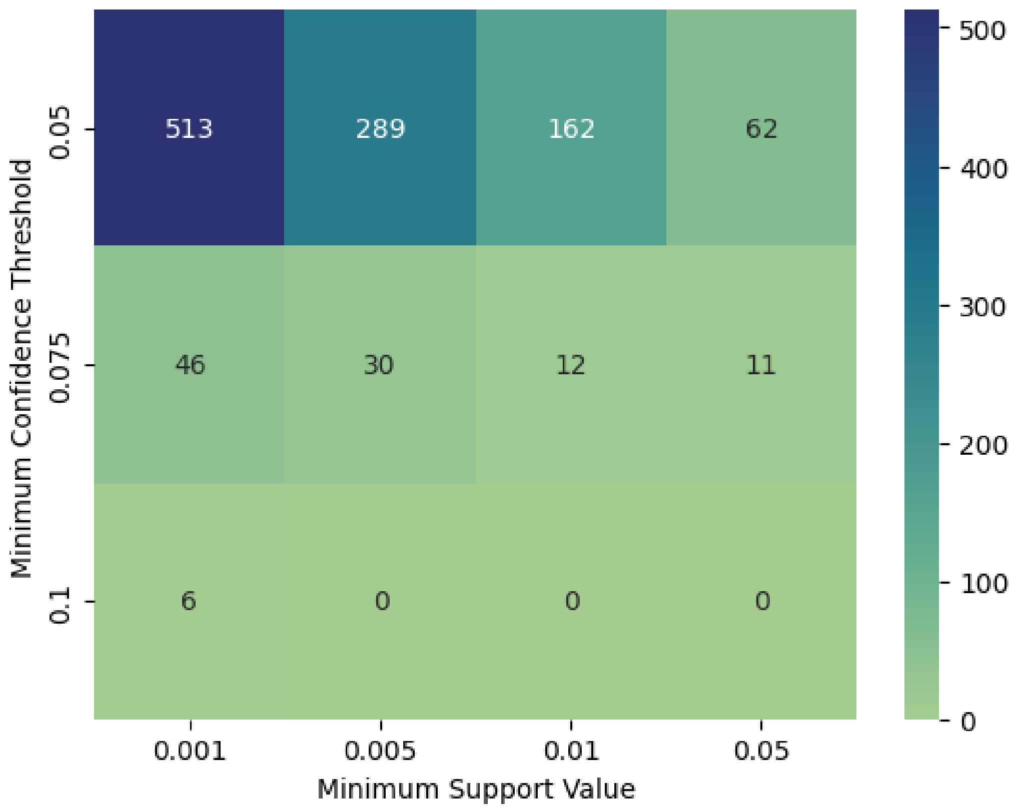
[1]:
```
     antecedents          consequents  antecedent support  \
0     (rolls/buns)        (whole milk)            0.109750
1         (yogurt)        (whole milk)            0.083750
2           (soda)  (other vegetables)            0.094500
3           (soda)        (whole milk)            0.094500
```

```
4        (whole milk)  (other vegetables)            0.158125
5  (other vegetables)        (whole milk)            0.123125

   consequent support   support  confidence      lift  leverage  conviction  \
0            0.158125  0.012875    0.117312  0.741895 -0.004479    0.953763
1            0.158125  0.011875    0.141791  0.896702 -0.001368    0.980967
2            0.123125  0.010000    0.105820  0.859453 -0.001635    0.980647
3            0.158125  0.010125    0.107143  0.677583 -0.004818    0.942900
4            0.123125  0.016875    0.106719  0.866756 -0.002594    0.981634
5            0.158125  0.016875    0.137056  0.866756 -0.002594    0.975585

   zhangs_metric
0      -0.280984
1      -0.111685
2      -0.152971
3      -0.344474
4      -0.154406
5      -0.149162
```

1) Part D

[2]:
```python
from itertools import product

from seaborn import heatmap
import numpy as np
import matplotlib.pyplot as plt

msvs = [0.001, 0.005, 0.01, 0.05]
mcts = [0.05, 0.075, 0.1]

# cartesion product msvs X mcts
msvs_cross_mcts = product(msvs, mcts)

# Create empty array of shape(y_vals, x_vals)
heatmap_data = np.zeros((len(mcts), len(msvs)))

# fill heatmap_data array
for index, (msv, mct) in enumerate(msvs_cross_mcts):
    n_association_rules = len(get_association_rules(dataset, msv, mct))
    heatmap_data.flat[index] = n_association_rules

# display heatmap
heatmap = heatmap(heatmap_data,
                  xticklabels=msvs,
                  yticklabels=mcts,
                  annot=True,
                  fmt='g',
```

```
                    cmap='crest')
plt.xlabel('Minimum Support Value')
plt.ylabel('Minimum Confidence Threshold')
plt.show()
```



1) Part e

[3]:
```
a_rules = get_association_rules(dataset, min_support=0.005,␣
 ↪min_confidence_threshold=0)
max_confidence = a_rules['confidence'].max()
max_confidence_rules = a_rules.query(f'confidence == {max_confidence}')
for _, max_confidence_rule in max_confidence_rules.iterrows():
    print("Rule: {} -> {}".format(max_confidence_rule.antecedents,␣
 ↪max_confidence_rule.consequents))
    print("The confidence is ", max_confidence_rule.confidence)
```

```
Rule: frozenset({'bottled beer'}) -> frozenset({'whole milk'})
The confidence is  0.16111111111111112
```

2 2 Image Classification using CNN

[4]:
```python
import tensorflow as tf
from tensorflow.keras import models, layers

LEARNING_RATE = 0.01
N_EPOCHS = 30
BATCH_SIZE = 32
IMAGE_SIZE = [100,100]

## Load Dataset ##
normalize = lambda image, label : (tf.cast(image, tf.float32) / 255.0, label)

IMAGES_DIR = './Cropped'
train_dataset, val_dataset = tf.keras.utils.image_dataset_from_directory(
    IMAGES_DIR,
    labels='inferred',
    label_mode='categorical',
    color_mode='rgb',
    batch_size=BATCH_SIZE,
    image_size=IMAGE_SIZE,  # Adjust image size as needed
    subset='both',
    validation_split=0.2,
    seed=42
)
train_dataset.map(normalize)
val_dataset.map(normalize)

## Create model ##
INPUT_SHAPE = IMAGE_SIZE + [3]

model = models.Sequential()
model.add(
    layers.Conv2D(8, (3,3), activation='relu', input_shape=INPUT_SHAPE)
)
model.add(
    layers.MaxPooling2D((2, 2))
)
model.add(
    layers.Flatten()
)
model.add(
    layers.Dense(16, activation='relu')
)
model.add(
    layers.Dense(4, activation='softmax')
)
```

```python
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=LEARNING_RATE),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()

# Train model
history = model.fit(train_dataset, epochs=N_EPOCHS,␣
 ↪validation_data=val_dataset, batch_size=BATCH_SIZE)

plt.figure(figsize=(10,5))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.xticks(np.arange(N_EPOCHS))
plt.show()
```

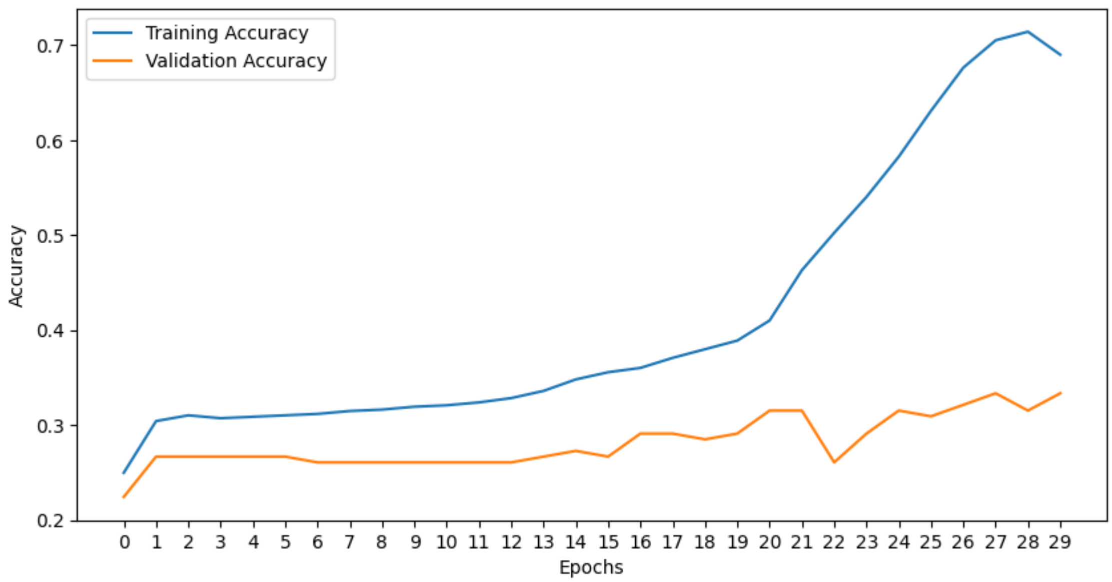
```
Found 826 files belonging to 4 classes.
Using 661 files for training.
Using 165 files for validation.
```

```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs
slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 98, 98, 8)         224

 max_pooling2d (MaxPooling2  (None, 49, 49, 8)         0
 D)

 flatten (Flatten)           (None, 19208)             0

 dense (Dense)               (None, 16)                307344

 dense_1 (Dense)             (None, 4)                 68

=================================================================
Total params: 307636 (1.17 MB)
Trainable params: 307636 (1.17 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/30
21/21 [==============================] - 1s 21ms/step - loss: 676.4770 -
accuracy: 0.2496 - val_loss: 1.3854 - val_accuracy: 0.2242
```

```
Epoch 2/30
21/21 [==============================] - 0s 16ms/step - loss: 1.3765 - accuracy:
0.3041 - val_loss: 1.3833 - val_accuracy: 0.2667
Epoch 3/30
21/21 [==============================] - 0s 14ms/step - loss: 1.3748 - accuracy:
0.3101 - val_loss: 1.3854 - val_accuracy: 0.2667
Epoch 4/30
21/21 [==============================] - 0s 15ms/step - loss: 1.3717 - accuracy:
0.3071 - val_loss: 1.3868 - val_accuracy: 0.2667
Epoch 5/30
21/21 [==============================] - 0s 14ms/step - loss: 1.3691 - accuracy:
0.3086 - val_loss: 1.3876 - val_accuracy: 0.2667
Epoch 6/30
21/21 [==============================] - 0s 15ms/step - loss: 1.3670 - accuracy:
0.3101 - val_loss: 1.3891 - val_accuracy: 0.2667
Epoch 7/30
21/21 [==============================] - 0s 14ms/step - loss: 1.3642 - accuracy:
0.3116 - val_loss: 1.3892 - val_accuracy: 0.2606
Epoch 8/30
21/21 [==============================] - 0s 15ms/step - loss: 1.3614 - accuracy:
0.3147 - val_loss: 1.3901 - val_accuracy: 0.2606
Epoch 9/30
21/21 [==============================] - 0s 14ms/step - loss: 1.3595 - accuracy:
0.3162 - val_loss: 1.3898 - val_accuracy: 0.2606
Epoch 10/30
21/21 [==============================] - 0s 15ms/step - loss: 1.3561 - accuracy:
0.3192 - val_loss: 1.3882 - val_accuracy: 0.2606
Epoch 11/30
21/21 [==============================] - 0s 16ms/step - loss: 1.3522 - accuracy:
0.3207 - val_loss: 1.3899 - val_accuracy: 0.2606
Epoch 12/30
21/21 [==============================] - 0s 16ms/step - loss: 1.3467 - accuracy:
0.3238 - val_loss: 1.3873 - val_accuracy: 0.2606
Epoch 13/30
21/21 [==============================] - 0s 15ms/step - loss: 1.3418 - accuracy:
0.3283 - val_loss: 1.3844 - val_accuracy: 0.2606
Epoch 14/30
21/21 [==============================] - 0s 15ms/step - loss: 1.3323 - accuracy:
0.3359 - val_loss: 1.3834 - val_accuracy: 0.2667
Epoch 15/30
21/21 [==============================] - 0s 15ms/step - loss: 1.3163 - accuracy:
0.3480 - val_loss: 1.3768 - val_accuracy: 0.2727
Epoch 16/30
21/21 [==============================] - 0s 15ms/step - loss: 1.3093 - accuracy:
0.3555 - val_loss: 1.3745 - val_accuracy: 0.2667
Epoch 17/30
21/21 [==============================] - 0s 15ms/step - loss: 1.2990 - accuracy:
0.3601 - val_loss: 1.3680 - val_accuracy: 0.2909
```

```
Epoch 18/30
21/21 [==============================] - 0s 15ms/step - loss: 1.2713 - accuracy:
0.3707 - val_loss: 1.3647 - val_accuracy: 0.2909
Epoch 19/30
21/21 [==============================] - 0s 15ms/step - loss: 1.2729 - accuracy:
0.3797 - val_loss: 1.3743 - val_accuracy: 0.2848
Epoch 20/30
21/21 [==============================] - 0s 18ms/step - loss: 1.2422 - accuracy:
0.3888 - val_loss: 1.3742 - val_accuracy: 0.2909
Epoch 21/30
21/21 [==============================] - 0s 17ms/step - loss: 1.2053 - accuracy:
0.4100 - val_loss: 1.3844 - val_accuracy: 0.3152
Epoch 22/30
21/21 [==============================] - 0s 16ms/step - loss: 1.1236 - accuracy:
0.4629 - val_loss: 1.4306 - val_accuracy: 0.3152
Epoch 23/30
21/21 [==============================] - 0s 16ms/step - loss: 1.0363 - accuracy:
0.5023 - val_loss: 1.5683 - val_accuracy: 0.2606
Epoch 24/30
21/21 [==============================] - 0s 16ms/step - loss: 0.9714 - accuracy:
0.5401 - val_loss: 1.7341 - val_accuracy: 0.2909
Epoch 25/30
21/21 [==============================] - 0s 16ms/step - loss: 0.8942 - accuracy:
0.5825 - val_loss: 1.7965 - val_accuracy: 0.3152
Epoch 26/30
21/21 [==============================] - 0s 15ms/step - loss: 0.8532 - accuracy:
0.6309 - val_loss: 2.1945 - val_accuracy: 0.3091
Epoch 27/30
21/21 [==============================] - 0s 15ms/step - loss: 0.7335 - accuracy:
0.6762 - val_loss: 2.9857 - val_accuracy: 0.3212
Epoch 28/30
21/21 [==============================] - 0s 15ms/step - loss: 0.6751 - accuracy:
0.7050 - val_loss: 3.1495 - val_accuracy: 0.3333
Epoch 29/30
21/21 [==============================] - 0s 16ms/step - loss: 0.6456 - accuracy:
0.7141 - val_loss: 3.4778 - val_accuracy: 0.3152
Epoch 30/30
21/21 [==============================] - 0s 15ms/step - loss: 0.7193 - accuracy:
0.6899 - val_loss: 3.9653 - val_accuracy: 0.3333
```



3 2) Part 2

Banner ID 916322804

[6]:
```
N_EPOCHS = 30
BATCH_SIZE = 32
LEARNING_RATE = 0.01

plt.figure(figsize=(10,5))
for index, n_filters in enumerate([4, 16]):
    model = models.Sequential()
    model.add(
        layers.Conv2D(n_filters, (3,3), activation='relu',␣
 ↪input_shape=INPUT_SHAPE)
    )
    model.add(
        layers.MaxPooling2D((2, 2))
    )
    model.add(
        layers.Flatten()
    )
    model.add(
        layers.Dense(16, activation='relu')
    )
    model.add(
        layers.Dense(4, activation='softmax')
    )
```

```
    model.compile(optimizer=tf.keras.optimizers.
↪Adam(learning_rate=LEARNING_RATE),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    model.summary()
    # Train model
    history = model.fit(train_dataset, epochs=N_EPOCHS,␣
↪validation_data=val_dataset, batch_size=BATCH_SIZE)
    plt.subplot(2, 1, index + 1)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.xticks(np.arange(N_EPOCHS))
    plt.title(f'{n_filters} filters')
plt.tight_layout()
plt.show()
```

```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs
slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 98, 98, 4)         112

 max_pooling2d_3 (MaxPoolin  (None, 49, 49, 4)         0
 g2D)

 flatten_3 (Flatten)         (None, 9604)              0

 dense_6 (Dense)             (None, 16)                153680

 dense_7 (Dense)             (None, 4)                 68

=================================================================
Total params: 153860 (601.02 KB)
Trainable params: 153860 (601.02 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/30
21/21 [==============================] - 1s 19ms/step - loss: 348.6417 -
accuracy: 0.2784 - val_loss: 1.4206 - val_accuracy: 0.3091
Epoch 2/30
21/21 [==============================] - 0s 13ms/step - loss: 1.3617 - accuracy:
```

```
0.3343 - val_loss: 1.3916 - val_accuracy: 0.2909
Epoch 3/30
21/21 [==============================] - 0s 14ms/step - loss: 1.3426 - accuracy:
0.3374 - val_loss: 1.3931 - val_accuracy: 0.2788
Epoch 4/30
21/21 [==============================] - 0s 14ms/step - loss: 1.3296 - accuracy:
0.3419 - val_loss: 1.3925 - val_accuracy: 0.2848
Epoch 5/30
21/21 [==============================] - 0s 16ms/step - loss: 1.3104 - accuracy:
0.3585 - val_loss: 1.3876 - val_accuracy: 0.2909
Epoch 6/30
21/21 [==============================] - 0s 15ms/step - loss: 1.2782 - accuracy:
0.3722 - val_loss: 1.3820 - val_accuracy: 0.3030
Epoch 7/30
21/21 [==============================] - 0s 14ms/step - loss: 1.2144 - accuracy:
0.4115 - val_loss: 1.4687 - val_accuracy: 0.3273
Epoch 8/30
21/21 [==============================] - 0s 14ms/step - loss: 1.0884 - accuracy:
0.4856 - val_loss: 1.7022 - val_accuracy: 0.3091
Epoch 9/30
21/21 [==============================] - 0s 14ms/step - loss: 0.9726 - accuracy:
0.5719 - val_loss: 1.8657 - val_accuracy: 0.3030
Epoch 10/30
21/21 [==============================] - 0s 18ms/step - loss: 0.8968 - accuracy:
0.6142 - val_loss: 3.0555 - val_accuracy: 0.2970
Epoch 11/30
21/21 [==============================] - 0s 14ms/step - loss: 0.8236 - accuracy:
0.6445 - val_loss: 3.4837 - val_accuracy: 0.3091
Epoch 12/30
21/21 [==============================] - 0s 15ms/step - loss: 0.7121 - accuracy:
0.6944 - val_loss: 3.5034 - val_accuracy: 0.3152
Epoch 13/30
21/21 [==============================] - 0s 15ms/step - loss: 0.6253 - accuracy:
0.7322 - val_loss: 3.8416 - val_accuracy: 0.3091
Epoch 14/30
21/21 [==============================] - 1s 23ms/step - loss: 0.5329 - accuracy:
0.7700 - val_loss: 4.2074 - val_accuracy: 0.3152
Epoch 15/30
21/21 [==============================] - 0s 16ms/step - loss: 0.4408 - accuracy:
0.8124 - val_loss: 4.1727 - val_accuracy: 0.3091
Epoch 16/30
21/21 [==============================] - 0s 20ms/step - loss: 0.4590 - accuracy:
0.8200 - val_loss: 4.0695 - val_accuracy: 0.3091
Epoch 17/30
21/21 [==============================] - 0s 14ms/step - loss: 0.4299 - accuracy:
0.8109 - val_loss: 5.3895 - val_accuracy: 0.2970
Epoch 18/30
21/21 [==============================] - 0s 15ms/step - loss: 0.3521 - accuracy:
```

```
0.8411 - val_loss: 5.6919 - val_accuracy: 0.2970
Epoch 19/30
21/21 [==============================] - 0s 15ms/step - loss: 0.3265 - accuracy:
0.8729 - val_loss: 5.8187 - val_accuracy: 0.3091
Epoch 20/30
21/21 [==============================] - 0s 16ms/step - loss: 0.3817 - accuracy:
0.8457 - val_loss: 6.3731 - val_accuracy: 0.2970
Epoch 21/30
21/21 [==============================] - 0s 14ms/step - loss: 0.9131 - accuracy:
0.6914 - val_loss: 3.2801 - val_accuracy: 0.2545
Epoch 22/30
21/21 [==============================] - 0s 14ms/step - loss: 0.8594 - accuracy:
0.6884 - val_loss: 8.9771 - val_accuracy: 0.3091
Epoch 23/30
21/21 [==============================] - 0s 14ms/step - loss: 0.5582 - accuracy:
0.7776 - val_loss: 9.8382 - val_accuracy: 0.2970
Epoch 24/30
21/21 [==============================] - 0s 16ms/step - loss: 0.3861 - accuracy:
0.8366 - val_loss: 10.3183 - val_accuracy: 0.3212
Epoch 25/30
21/21 [==============================] - 0s 14ms/step - loss: 0.3987 - accuracy:
0.8457 - val_loss: 8.2636 - val_accuracy: 0.3091
Epoch 26/30
21/21 [==============================] - 0s 15ms/step - loss: 0.3963 - accuracy:
0.8608 - val_loss: 11.4684 - val_accuracy: 0.2909
Epoch 27/30
21/21 [==============================] - 0s 15ms/step - loss: 0.3372 - accuracy:
0.8638 - val_loss: 15.4791 - val_accuracy: 0.3030
Epoch 28/30
21/21 [==============================] - 0s 14ms/step - loss: 0.3140 - accuracy:
0.8669 - val_loss: 14.6776 - val_accuracy: 0.3030
Epoch 29/30
21/21 [==============================] - 0s 16ms/step - loss: 0.5451 - accuracy:
0.8442 - val_loss: 9.2552 - val_accuracy: 0.3030
Epoch 30/30
21/21 [==============================] - 0s 14ms/step - loss: 0.4682 - accuracy:
0.8502 - val_loss: 12.9274 - val_accuracy: 0.3091
```

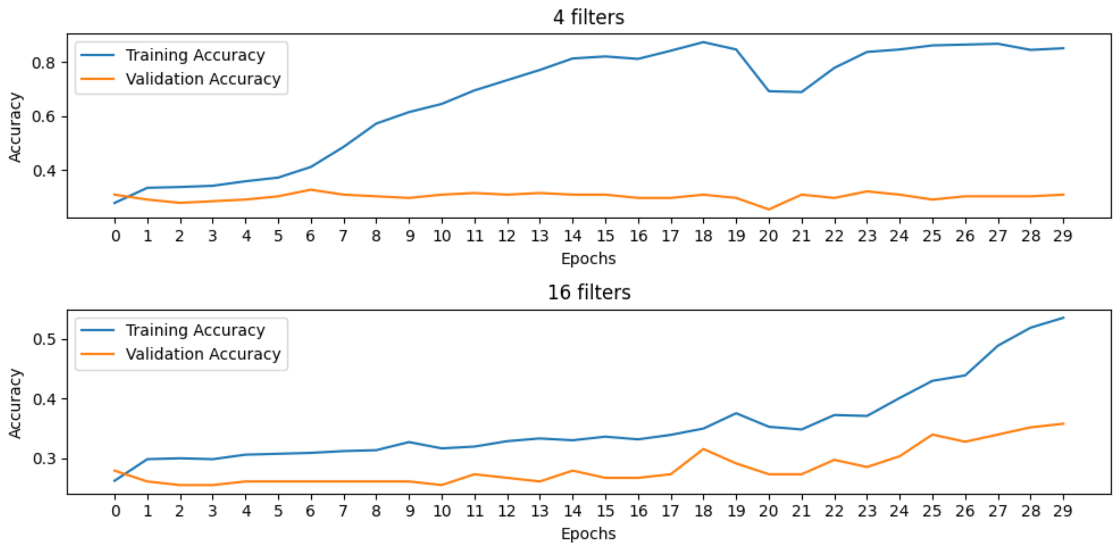
```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs
slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 98, 98, 16)        448


 max_pooling2d_4 (MaxPoolin  (None, 49, 49, 16)        0
```

```
 g2D)

 flatten_4 (Flatten)         (None, 38416)             0

 dense_8 (Dense)             (None, 16)                614672

 dense_9 (Dense)             (None, 4)                 68

=================================================================
Total params: 615188 (2.35 MB)
Trainable params: 615188 (2.35 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/30
21/21 [==============================] - 1s 22ms/step - loss: 1142.8787 -
accuracy: 0.2617 - val_loss: 1.3874 - val_accuracy: 0.2788
Epoch 2/30
21/21 [==============================] - 0s 19ms/step - loss: 1.3808 - accuracy:
0.2980 - val_loss: 1.3896 - val_accuracy: 0.2606
Epoch 3/30
21/21 [==============================] - 0s 18ms/step - loss: 1.3800 - accuracy:
0.2995 - val_loss: 1.3923 - val_accuracy: 0.2545
Epoch 4/30
21/21 [==============================] - 0s 18ms/step - loss: 1.3799 - accuracy:
0.2980 - val_loss: 1.3928 - val_accuracy: 0.2545
Epoch 5/30
21/21 [==============================] - 0s 20ms/step - loss: 1.3779 - accuracy:
0.3056 - val_loss: 1.3914 - val_accuracy: 0.2606
Epoch 6/30
21/21 [==============================] - 0s 18ms/step - loss: 1.3730 - accuracy:
0.3071 - val_loss: 1.3941 - val_accuracy: 0.2606
Epoch 7/30
21/21 [==============================] - 0s 21ms/step - loss: 1.3676 - accuracy:
0.3086 - val_loss: 1.3944 - val_accuracy: 0.2606
Epoch 8/30
21/21 [==============================] - 1s 22ms/step - loss: 1.3635 - accuracy:
0.3116 - val_loss: 1.3950 - val_accuracy: 0.2606
Epoch 9/30
21/21 [==============================] - 0s 19ms/step - loss: 1.3590 - accuracy:
0.3132 - val_loss: 1.3995 - val_accuracy: 0.2606
Epoch 10/30
21/21 [==============================] - 0s 19ms/step - loss: 1.3568 - accuracy:
0.3268 - val_loss: 1.3918 - val_accuracy: 0.2606
Epoch 11/30
21/21 [==============================] - 0s 18ms/step - loss: 1.3535 - accuracy:
0.3162 - val_loss: 1.3910 - val_accuracy: 0.2545
Epoch 12/30
21/21 [==============================] - 0s 19ms/step - loss: 1.3485 - accuracy:
```

```
0.3192 - val_loss: 1.4113 - val_accuracy: 0.2727
Epoch 13/30
21/21 [==============================] - 0s 18ms/step - loss: 1.3392 - accuracy:
0.3283 - val_loss: 1.4110 - val_accuracy: 0.2667
Epoch 14/30
21/21 [==============================] - 0s 19ms/step - loss: 1.3378 - accuracy:
0.3328 - val_loss: 1.4047 - val_accuracy: 0.2606
Epoch 15/30
21/21 [==============================] - 0s 20ms/step - loss: 1.3350 - accuracy:
0.3298 - val_loss: 1.4144 - val_accuracy: 0.2788
Epoch 16/30
21/21 [==============================] - 0s 21ms/step - loss: 1.3265 - accuracy:
0.3359 - val_loss: 1.4130 - val_accuracy: 0.2667
Epoch 17/30
21/21 [==============================] - 0s 18ms/step - loss: 1.3324 - accuracy:
0.3313 - val_loss: 1.4240 - val_accuracy: 0.2667
Epoch 18/30
21/21 [==============================] - 0s 19ms/step - loss: 1.3234 - accuracy:
0.3389 - val_loss: 1.4558 - val_accuracy: 0.2727
Epoch 19/30
21/21 [==============================] - 0s 19ms/step - loss: 1.3075 - accuracy:
0.3495 - val_loss: 1.5542 - val_accuracy: 0.3152
Epoch 20/30
21/21 [==============================] - 0s 20ms/step - loss: 1.2858 - accuracy:
0.3752 - val_loss: 1.5512 - val_accuracy: 0.2909
Epoch 21/30
21/21 [==============================] - 0s 18ms/step - loss: 1.3943 - accuracy:
0.3525 - val_loss: 1.4053 - val_accuracy: 0.2727
Epoch 22/30
21/21 [==============================] - 0s 18ms/step - loss: 1.3304 - accuracy:
0.3480 - val_loss: 1.4203 - val_accuracy: 0.2727
Epoch 23/30
21/21 [==============================] - 0s 19ms/step - loss: 1.2763 - accuracy:
0.3722 - val_loss: 1.5726 - val_accuracy: 0.2970
Epoch 24/30
21/21 [==============================] - 0s 19ms/step - loss: 1.2768 - accuracy:
0.3707 - val_loss: 1.4456 - val_accuracy: 0.2848
Epoch 25/30
21/21 [==============================] - 0s 19ms/step - loss: 1.2317 - accuracy:
0.4009 - val_loss: 1.5588 - val_accuracy: 0.3030
Epoch 26/30
21/21 [==============================] - 0s 18ms/step - loss: 1.2158 - accuracy:
0.4297 - val_loss: 2.0493 - val_accuracy: 0.3394
Epoch 27/30
21/21 [==============================] - 0s 18ms/step - loss: 1.1699 - accuracy:
0.4387 - val_loss: 1.8856 - val_accuracy: 0.3273
Epoch 28/30
21/21 [==============================] - 0s 20ms/step - loss: 1.0851 - accuracy:
```

```
0.4887 - val_loss: 1.7559 - val_accuracy: 0.3394
Epoch 29/30
21/21 [==============================] - 0s 18ms/step - loss: 1.0104 - accuracy:
0.5189 - val_loss: 2.5837 - val_accuracy: 0.3515
Epoch 30/30
21/21 [==============================] - 0s 19ms/step - loss: 0.9908 - accuracy:
0.5356 - val_loss: 3.1492 - val_accuracy: 0.3576
```



Describe and discuss what you observe by comparing the performance of the first model and the other two models you constructed in (a), (b) or (c) (depending on which one you did). Are there model overfit or underfit or just right? (1 point)

The model is performing extermely inconsistenly due to the relatively small size of the dense layer compared to the output of the convolutional layer. Also, we do not have many datapoints per class so validation accuracy will always be poor. The model gets trapped into local minima most of the time, and when it doesn't it simply memorizes the training data hence the low validation score.

During the experiment I tried raising and decreasing the number of kernels. Decreasing this number to 4 led to a smaller output size therefore leading to better training accuracy. Increasing it to 16 made the output of the convolutional layer bigger which in this case decreased training accuracy. Other runs yielded different results, with either one or both models getting stuck in a local minima or the 16 filter model winning. It all depends on the gradiant descent path and initialization, which is random.

Overall I will say all three models are underfitting due to the dense layer being too small, and the relatively small amount of data.