# 6.047 Problem Set 4 Writeup

Matthew Feng

November 4, 2018

## 1 Simulated GWAS

## 2 Finding eQTLs

### (a) Principle Components Analysis

### (b) Finding eQTLs via Linear Regression

For every SNP $x_i$, we find the mean and variance $(\mu_i, \sigma_i^2)$ of the correlation coefficients $r_{ij}$ that $x_i$ has with expression of gene $y_j$. In this way, we are determining the "typical" contribution of SNP $x_i$ to any gene. We then select the SNP and gene pairs $(x_i, e_j)$ for which $r_{ij}$ is satistically significant under the null hypothesis $H_0 : \rho_{ij} = \mu_i$ (i.e. contribution of $x_i$ to $y_j$ is typical).

To implement the Bonferroni correction, we test each hypothesis that SNP $x_i$ contributes to the expression of gene $y_j$ with significance of $\alpha/5000$, where $\alpha = 0.05$ (since we are testing 5000 different genes corresponding to 5000 different hypotheses).

## 3 Convolutional Neural Networks

### (a) Implementation

```python
#!/usr/bin/env python

from keras.models import *
from keras.layers import *
import keras

import numpy as np

from datetime import datetime

import argparse
```

```python
BATCH_SIZE = 10
NUM_EPOCHS = 20
KERNEL_SIZE = (4, 4)
POOL_SIZE = (4, 6)
HIDDEN_UNITS = 32
CONV_FILTERS = 32

def get_x_y_data():
    negative_data = []
    with open('negativedata.txt') as f:
        for line in f:
            final_mat = np.zeros((4,len(line)-1,1))
            for i in range(len(line)):
                char = line[i]
                if char == 'a':
                    final_mat[:,i,:] = np.array([[1],[0],[0],[0]])
                if char == 'c':
                    final_mat[:,i,:] = np.array([[0],[1],[0],[0]])
                if char == 'g':
                    final_mat[:,i,:] = np.array([[0],[0],[1],[0]])
                if char == 't':
                    final_mat[:,i,:] = np.array([[0],[0],[0],[1]])
            negative_data.append(final_mat)

    positive_data = []
    with open('positivedata.txt') as f:

        for line in f:
            final_mat = np.zeros((4,len(line)-1,1))
            for i in range(len(line)):
                char = line[i]
                if char == 'a':
                    final_mat[:,i,:] = np.array([[1],[0],[0],[0]])
                if char == 'c':
                    final_mat[:,i,:] = np.array([[0],[1],[0],[0]])
                if char == 'g':
                    final_mat[:,i,:] = np.array([[0],[0],[1],[0]])
                if char == 't':
                    final_mat[:,i,:] = np.array([[0],[0],[0],[1]])
            positive_data.append(final_mat)

    X = np.array(negative_data + positive_data)
    y = np.array([0] * len(negative_data) + [1] * len(positive_data))
    y = keras.utils.to_categorical(y)

    X_neg = X[:len(negative_data), ...]
```

```python
    X_pos = X[len(negative_data):, ...]
    y_neg = y[:len(negative_data), ...]
    y_pos = y[len(negative_data):, ...]

    return X_neg, X_pos, y_neg, y_pos

def create_model():
    model = Sequential()
    model.add(Conv2D(CONV_FILTERS,
                     input_shape=(4, 100, 1),
                     kernel_size=KERNEL_SIZE,
                     activation="relu",
                     padding="same"))
    model.add(MaxPool2D(pool_size=POOL_SIZE))
    model.add(Flatten())
    model.add(Dense(HIDDEN_UNITS, activation="relu"))
    model.add(Dense(2, activation="softmax")) # same as 1 output sigmoid
    return model

def main():
    np.random.seed(1)

    TRAIN_TEST_FRAC = 0.9
    DATASET_SIZE = 5000
    # 10000 x (4, 100, 1) images total (5000 examples each)
    SPLIT = int(TRAIN_TEST_FRAC * DATASET_SIZE)

    Xn, Xp, yn, yp = get_x_y_data()
    shuffled_order = np.arange(0, DATASET_SIZE)
    np.random.shuffle(shuffled_order)
    Xn, Xp = Xn[shuffled_order, ...], Xp[shuffled_order, ...]
    yn, yp = yn[shuffled_order, ...], yp[shuffled_order, ...]

    X_train = np.vstack((Xn[:SPLIT, ...], Xp[:SPLIT, ...]))
    y_train = np.vstack((yn[:SPLIT, ...], yp[:SPLIT, ...]))

    X_test = np.vstack((Xn[SPLIT:, ...], Xp[SPLIT:, ...]))
    y_test = np.vstack((yn[SPLIT:, ...], yp[SPLIT:, ...]))

    print(X_train.shape)

    # define model
    model = create_model()
    model.compile(loss="categorical_crossentropy",
        optimizer="adam",
        metrics=["accuracy"])
```

```
    start = datetime.now()
    model.fit(X_train, y_train, epochs=NUM_EPOCHS, batch_size=BATCH_SIZE)
    end = datetime.now()

    scores = model.evaluate(X_test, y_test)

    print("\n{}: {:.2f}%".format(model.metrics_names[1], scores[1] * 100))
    print("elapsed: {}".format(str(end - start)))

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "-b", "--batch_size",
        help="Number of examples per batch",
        type=int)
    parser.add_argument(
        "-e", "--epochs",
        help="Number of epochs to train over",
        type=int)
    parser.add_argument(
        "-k", "--kernel",
        help="(height, width) tuple representing size of kernel.",
        type=tuple)
    parser.add_argument(
        "-p", "--pool",
        help="(height, width) tuple representing size of pool.",
        type=tuple)
    parser.add_argument(
        "-u", "--hidden_units",
        help="Number of hidden units for the Dense layer",
        type=int)
    parser.add_argument(
        "-f", "--num_filters",
        help="Number of filters for the Conv layer",
        type=int)

    args = parser.parse_args()

    if args.batch_size:
        BATCH_SIZE = args.batch_size

    if args.epochs:
        NUM_EPOCHS = args.epochs

    if args.kernel:
```

```python
        if len(args.kernel) != 2:
            print("Kernel size must be tuple of length 2")
            quit()
        KERNEL_SIZE = args.kernel

    if args.pool:
        if len(args.pool) != 2:
            print("Pooling size must be tuple of length 2")
            quit()

        POOL_SIZE = args.pool

    if args.hidden_units:
        HIDDEN_UNITS = args.hidden_units

    if args.num_filters:
        CONV_FILTERS = args.num_filters

    main()

# acc: 94.70%
# elapsed: 0:00:56.455801
```

### (b) Layers

### (c) Hyperparameters

| Model | Hyperparameters | Train Accuracy | Test Accuracy | Training Time |
|---|---|---|---|---|
| (1) | `epochs` $= 300$<br>`batch_size` $= 10$<br>`kernel_size` $= (4, 4)$<br>`pool_size` $= (4, 6)$<br>`hidden_units` $= 32$<br>`num_filters` $= 32$ | 100% | 94.8% | 00:13:02 |
| (2) | `epochs` $= 50$<br>`batch_size` $= 10$<br>`kernel_size` $= (4, 4)$<br>`pool_size` $= (4, 6)$<br>`hidden_units` $= 32$<br>`num_filters` $= 1024$ | c | d | |
| (3) | `epochs` $= 50$<br>`batch_size` $= 10$<br>`kernel_size` $= (4, 4)$<br>`pool_size` $= (4, 24)$<br>`hidden_units` $= 512$<br>`num_filters` $= 32$ | c | d | |

### (d) Architectures