

6.047 Problem Set 4 Writeup

Matthew Feng

November 4, 2018

1 Simulated GWAS

(a) β_i and SNP odds ratio

(b), (c)

See `simulated_gwas.py`.

(d) Bonferroni Correction

If we are using a significance level of $\alpha = 0.05$, then the p -value needed for significance according to the Bonferroni correction is $\frac{0.05}{m}$. The Bonferroni correction is necessary when performing GWAS because we are conducting multiple significance tests simultaneously; if we continued to only use α , then by chance alone $m \times 0.05$ SNPs would be statistically significant at the $\alpha = 0.05$ level.

(e) Hyperparameters

Accuracy (AC) ($\frac{TP+TN}{TP+TN+FP+FN}$)

Precision (PRC) ($\frac{TP}{TP+FP}$)

Recall (RCL) ($\frac{TP}{TP+FN}$)

| Hyperparameters | TP | FP | TN | FN | AC | PRC | RCL |
|---|----|----|------|-----|-------|------|------|
| $n = 10000, k = 100$ $m = 1000, s = 0.25$ | 8 | 0 | 900 | 92 | 0.908 | 1.00 | 0.08 |
| $n = 1000, k = 100$ $m = 1000, s = 0.25$ | 0 | 1 | 899 | 100 | 0.899 | 0.00 | 0.00 |
| $n = 100000, k = 100$ $m = 1000, s = 0.25$ | 49 | 0 | 900 | 51 | 0.959 | 1.00 | 0.49 |
| $n = 10000, k = 100$ $m = 10000, s = 0.25$ | 8 | 0 | 9900 | 92 | 0.991 | 1.00 | 0.08 |
| $n = 10000, k = 100$ $m = 1000, s = 0.5$ | 10 | 0 | 900 | 90 | 0.910 | 1.00 | 0.10 |
| $n = 10000, k = 100$ $m = 1000, s = 0.1$ | 3 | 0 | 900 | 97 | 0.903 | 1.00 | 0.03 |

The Bonferroni correction is conservative in categorizing SNPs as contributors to the “affected” phenotype; that is, recall tends to be very low (across all hyperparameter settings), while precision is high (typically 1.00).

As the number of samples increases (i.e. the number of people n), accuracy, precision, and recall all improve. As the standard deviation s of the disease related SNPs increases, so too does recall. When the number of SNPs m increases, accuracy increases as the model tends to correctly classify negatives, but recall remains low.

2 Finding eQTLs

(a) Principal Components Analysis

(b) Finding eQTLs via Linear Regression

For every SNP x_i , we find the mean and variance (μ_i, σ_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 statistically 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).

(c) Additional Datasets

3 Convolutional Neural Networks

(a) Implementation

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

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

import numpy as np

from datetime import datetime

import alternative_models as models

import argparse

BATCH_SIZE = 10
NUM_EPOCHS = 20
KERNEL_SIZE = (4, 4)
POOL_SIZE = (4, 6)
HIDDEN_UNITS = 32
CONV_FILTERS = 32
MODEL_NAME = None

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), ...]
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

MODEL_FUNC = create_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 = MODEL_FUNC()
model.compile(loss="categorical_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])

start = datetime.now()

if MODEL_NAME == "lstm":
    X_train = X_train.squeeze()
    X_train = np.swapaxes(X_train, 1, 2)
    X_test = X_test.squeeze()
    X_test = np.swapaxes(X_test, 1, 2)

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=str)
parser.add_argument(
    "-p", "--pool",
    help="height,width tuple representing size of pool.",
    type=str)
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)
parser.add_argument(
    "-m", "--model",
    help="Use a particular model implemented in alternative_models.py",
    type=str)

args = parser.parse_args()

if args.batch_size:
    BATCH_SIZE = args.batch_size

if args.epochs:
    NUM_EPOCHS = args.epochs

if args.kernel:
    height, width = map(int, args.kernel.split(","))
    KERNEL_SIZE = (height, width)

if args.pool:
    height, width = map(int, args.pool.split(","))
    POOL_SIZE = (height, width)

if args.hidden_units:
    HIDDEN_UNITS = args.hidden_units

if args.num_filters:
    CONV_FILTERS = args.num_filters

```

```

    if args.model:
        MODEL_NAME = args.model
        MODEL_FUNC = getattr(models, args.model)

    main()

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

```

(b) Layers

(c) Hyperparameters

| Model | Hyperparameters | Train Accuracy | Test Accuracy | Training Time |
|-------|----------------------|----------------|---------------|---------------|
| (1) | epochs = 300 | 100.00% | 94.80% | 00:13:02 |
| | batch_size = 10 | | | |
| | kernel_size = (4, 4) | | | |
| | pool_size = (4, 6) | | | |
| | hidden_units = 32 | | | |
| (2) | num_filters = 32 | 99.32% | 95.60% | 00:35:37 |
| | epochs = 50 | | | |
| | batch_size = 10 | | | |
| | kernel_size = (4, 4) | | | |
| | pool_size = (4, 6) | | | |
| (3) | hidden_units = 32 | 97.56% | 93.50% | 00:02:34 |
| | num_filters = 1024 | | | |
| | epochs = 50 | | | |
| | batch_size = 10 | | | |
| | kernel_size = (4, 4) | | | |
| | pool_size = (4, 24) | | | |
| | hidden_units = 512 | | | |
| | num_filters = 32 | | | |

(d) Architectures

| Model | Architecture | AC (Train) | AC (Test) | Time |
|-----------------|----------------------|------------|-----------|----------|
| deep_conv_net | Conv2D(32, (4, 6)) | 99.36% | 98.10% | 00:10:25 |
| | Conv2D(64, (4, 3)) | | | |
| | MaxPool((4, 6)) | | | |
| | Conv2D(128, (4, 3)) | | | |
| | Conv2D(1024, (1, 1)) | | | |
| | Dense(64) | | | |
| fully_connected | Dense(2) | 99.58% | 93.20% | 00:03:02 |
| | Dense(1024) | | | |
| | Dense(256) | | | |
| | Dense(512) | | | |
| | Dense(2) | | | |
| lstm | LSTM(64) | 98.40% | 98.30% | 00:23:17 |
| | Dense(2) | | | |