Programming Assignment #2

T81-559: Applications of Deep Neural Networks, Washington University September 16, 2017

Listing 1 shows a sample submission skeleton that you can use as a starting point for this assignment.

Listing 1: Sample Submission Skeleton

```python


path = "./data/"

# Helpful functions

# Encode text values to dummy variables(i.e. [1,0,0],[0,1,0],[0,0,1] for red,green,blue)
def encode_text_dummy(df, name):
    dummies = pd.get_dummies(df[name])
    for x in dummies.columns:
        dummy_name = "{}-{}".format(name, x)
        df[dummy_name] = dummies[x]
    df.drop(name, axis=1, inplace=True)


# Encode text values to a single dummy variable.  The new columns (which do not replace the old) will have a 1
# at every location where the original column (name) matches each of the target_values.  One column is added for
# each target value.
def encode_text_single_dummy(df, name, target_values):
    for tv in target_values:
        l = list(df[name].astype(str))
        l = [1 if str(x) == str(tv) else 0 for x in l]
        name2 = "{}-{}".format(name, tv)
        df[name2] = l


# Encode text values to indexes(i.e. [1],[2],[3] for red,green,blue).
def encode_text_index(df, name):
    le = preprocessing.LabelEncoder()
    df[name] = le.fit_transform(df[name])
    return le.classes_


```

```python
# Encode a numeric column as zscores
def encode_numeric_zscore(df, name, mean=None, sd=None):
    if mean is None:
        mean = df[name].mean()

    if sd is None:
        sd = df[name].std()

    df[name] = (df[name] - mean) / sd


# Convert all missing values in the specified column to the median
def missing_median(df, name):
    med = df[name].median()
    df[name] = df[name].fillna(med)


# Convert all missing values in the specified column to the default
def missing_default(df, name, default_value):
    df[name] = df[name].fillna(default_value)


# Convert a Pandas dataframe to the x,y inputs that TensorFlow needs
def to_xy(df, target):
    result = []
    for x in df.columns:
        if x != target:
            result.append(x)
    # find out the type of the target column.  Is it really this hard? :(
    target_type = df[target].dtypes
    target_type = target_type[0] if hasattr(target_type, '__iter__') else target_type
    # Encode to int for classification, float otherwise. TensorFlow likes 32 bits.
    if target_type in (np.int64, np.int32):
        # Classification
        dummies = pd.get_dummies(df[target])
        return df.as_matrix(result).astype(np.float32), dummies.as_matrix().astype(np.float32)
    else:
        # Regression
        return df.as_matrix(result).astype(np.float32), df.as_matrix([target]).astype(np.float32)

# Nicely formatted time string
def hms_string(sec_elapsed):
    h = int(sec_elapsed / (60 * 60))
```

```python
    m = int((sec_elapsed % (60 * 60)) / 60)
    s = sec_elapsed % 60
    return "{}:{:>02}:{:>05.2f}".format(h, m, s)


# Regression chart.
def chart_regression(pred,y,sort=True):
    t = pd.DataFrame({'pred' : pred, 'y' : y.flatten()})
    if sort:
        t.sort_values(by=['y'],inplace=True)
    a = plt.plot(t['y'].tolist(),label='expected')
    b = plt.plot(t['pred'].tolist(),label='prediction')
    plt.ylabel('output')
    plt.legend()
    plt.show()

# Remove all rows where the specified column is +/- sd standard deviations
def remove_outliers(df, name, sd):
    drop_rows = df.index[(np.abs(df[name] - df[name].mean()) >= (sd * df[name].std()))]
    df.drop(drop_rows, axis=0, inplace=True)


# Encode a column to a range between normalized_low and normalized_high.
def encode_numeric_range(df, name, normalized_low=-1, normalized_high=1,
                         data_low=None, data_high=None):
    if data_low is None:
        data_low = min(df[name])
        data_high = max(df[name])

    df[name] = ((df[name] - data_low) / (data_high - data_low)) \
               * (normalized_high - normalized_low) + normalized_low

# Solution

def encode_toy_dataset(filename):
    df = pd.read_csv(filename, na_values=['NA', '?'])
    encode_numeric_zscore(df, 'length')
    encode_numeric_zscore(df, 'width')
    encode_numeric_zscore(df, 'height')
    encode_text_dummy(df, 'metal')
    encode_text_dummy(df, 'shape')
    return df

# Encode the toy dataset
def question1():
    print()
```

```
    print("***Question 1***")

    path = "./data/"

    filename_read = os.path.join(path,"toy1.csv")
    filename_write = os.path.join(path,"submit-jheaton-prog2q1.csv")
    df = encode_toy_dataset(filename_read) # You just have to implement encode_toy_dataset above
    df.to_csv(filename_write,index=False)
    print("Wrote {} lines.".format(len(df)))


# Model the toy dataset, no cross validation
def question2():
    print()
    print("***Question 2***")

def question3():
    print()
    print("***Question 3***")

    # Z-Score encode these using the mean/sd from the dataset (you got this in question 2)
    testDF = pd.DataFrame([
            {'length':1, 'width':2, 'height': 3},
            {'length':3, 'width':2, 'height': 5},
            {'length':4, 'width':1, 'height': 3}
        ])


def question4():
    print()
    print("***Question 4***")


def question5():
    print()
    print("***Question 5***")


question1()
question2()
question3()
question4()
question5()
```

Listing 2 shows what the output from this assignment would look like. Your numbers might differ from mine slightly. Every question, except 2, also generates an output CSV file. For your submission please include your Jupyter notebook and any generated CSV files that the questions specified. Name your output CSV files something such as **submit-jheaton-prog2q1.csv**. Submit a ZIP file that contains your Jupyter notebook and 4 CSV files to Blackboard. This will be 5 files total.

Listing 2: Expected Output

```
***Question 1***
Wrote 10001 lines.

***Question 2***
Epoch 00144: early stopping
Final score (RMSE): 75.46247100830078

***Question 3***
length: (5.5258474152584744, 2.8609014041584113)
width: (5.5340465953404658, 2.8598366585224158)
height: (5.5337466253374661, 2.8719829476156122)
     height    length     width
0 -0.882205 -1.581907 -1.235659
1 -0.185856 -0.882861 -1.235659
2 -0.882205 -0.533338 -1.585338

***Question 4***
Fold #1
Epoch 00060: early stopping
Fold score (RMSE): 0.21216803789138794
Fold #2
Epoch 00061: early stopping
Fold score (RMSE): 0.14340682327747345
Fold #3
Epoch 00028: early stopping
Fold score (RMSE): 0.3336745500564575
Fold #4
Epoch 00058: early stopping
Fold score (RMSE): 0.2133668214082718
Fold #5
Epoch 00077: early stopping
Fold score (RMSE): 0.1796143352985382
Final, out of sample score (RMSE): 0.22570167481899261

***Question 5***
Fold #1
Epoch 00182: early stopping
```

```
Fold score: 0.3625
Fold #2
Epoch 00425: early stopping
Fold score: 0.9875
Fold #3
Epoch 00169: early stopping
Fold score: 0.975
Fold #4
Epoch 00111: early stopping
Fold score: 0.8987341772151899
Fold #5
Epoch 00203: early stopping
Fold score: 0.8227848101265823
Final, out of sample score: 0.8090452261306532
```

Question 1

Use the dataset found here for this question: [click for toy dataset].

Encode the **toy1.csv** dataset. Generate dummy variables for the shape and metal. Encode height, width and length as z-scores. Include, but do not encode the weight. If this encoding is performed in a function, named **encode_toy_dataset**, you will have an easier time reusing the code from question 1 in question 2.

Write the output to a CSV file that you will submit with this assignment. The CSV file will look similar to Listing 3.

Listing 3: Question 2 Output Sample

```
height,length,width,weight,metal-bronze,metal-gold,metal-platinum,metal-silver,metal-tin,shape-box,shape-cylinder,shape-sphere
-0.18585564084337075, -0.18381430131795315, -0.1866234261937586, 729.63,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0
-0.5340303145851667, -0.18381430131795315, 0.16305509393694917, 2530.8,0.0,0.0,1.0,0.0,0.0,1.0,0.0,0.0
-1.2303796620687586, -1.2323841890415879, -1.5853375067165896, 58.37,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0
-0.8822049883269626, -0.8828608931337095, -0.8859804664551741, 74.15,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0
...
```

Question 2

Use the dataset found here for this question: [click for toy dataset].

Use the encoded dataset from question 1 and train a neural network to predict weight. Use 25% of the data as validation and 75% as training, make sure you shuffle the data. Report the RMSE error for the validation set. No CSV file need be generated for this question.

Question 3

Use the dataset found here for this question: [click for toy dataset].

Using the **toy1.csv** dataset calculate and report the mean and standard deviation for height, width and length. Calculate the z-scores for the dataframe given by Listing 4. Make sure that you use the mean and standard deviations you reported for this question. Write the results to a CSV file.

Listing 4: Question 3 Input Data

```
    testDF = pd.DataFrame([
            {'length':1, 'width':2, 'height': 3},
            {'length':3, 'width':2, 'height': 5},
            {'length':4, 'width':1, 'height': 3}
        ])
...
```

Your resulting CSV file should look almost exactly like Listing 5.

Listing 5: Question 3 Output Sample

```
height,length,width
-0.8822049883269626,-1.5819074849494659,-1.2356589865858818
-0.18585564084337075,-0.8828608931337095,-1.2356589865858818
-0.8822049883269626,-0.5333375972258314,-1.5853375067165896
```

Question 4

Use the dataset found here for this question: [click for iris dataset].

Usually the **iris.csv** dataset is used to classify the species. Not this time! Use the fields species, sepal-l, sepal-w, and petal-l to predict petal-w. Use a 5-fold cross validation and report ONLY out-of-sample predictions to a CSV file. Make sure to shuffle the data. Your generated CSV file should look similar to Listing 6. Encode each of the inputs in a way that makes sense (e.g. dummies, z-scores).

Listing 6: Question 4 Output Sample

```
sepal_l,sepal_w,petal_l,petal_w,species-Iris-setosa,species-Iris-↩
    versicolor,species-Iris-virginica,0,0
```

```
0.30995914214417364, -0.5903951331558184, 0.5336208818725668, ↩
    1.2,0.0,1.0,0.0,1.2,1.444551944732666
-0.1730940663922016, 1.7038864723719687, -1.1658086782311483, ↩
    0.3,1.0,0.0,0.0,0.3,0.\
...
```

Question 5

Use the dataset found here for this question: [click for auto mpg dataset].

Usually the **auto-mpg.csv** dataset is used to regress the mpg. Not this time! Use the fields to predict how many cylinders the car has. Treat this as a classification problem, where there is a class for each number of cylinders. Use a 5-fold cross validation and report ONLY out-of-sample predictions to a CSV file. Make sure to shuffle the data. Your generated CSV file should look similar to Listing 7. Encode each of the inputs in a way that makes sense (e.g. dummies, z-scores). Report the final out of sample accuracy score.

Listing 7: Question 4 Output Sample

```
mpg,cylinders,displacement,horsepower,weight,acceleration,year,origin,name↩
    ,ideal,predict
-0.7055506566787514, 8, 1.0892327311042995, 0.6722714619460141, ↩
    0.6300768256149949, -1.2938698102195594, 70, -0.7142457922976494,↩
    chevrolet chevelle malibu,8,8
-1.0893794720944747, 8, 1.5016242793620063, 1.5879594901955474, ↩
    0.8532590135498572, -1.4751810504376373, 70, -0.7142457922976494,buick↩
     skylark 320,8,8
-0.7055506566787514, 8, 1.1947282434492943, 1.19552176380289, ↩
    0.5497784722839334, -1.6564922906557151, 70, -0.7142457922976494,↩
    plymouth satellite,8,8
...
```