HW5

February 16, 2023

1 Homework 5: Embeddings and Model Selection

1.1 Bonus Exercise 1: Sentiment Analysis using GloVe Embeddings and Linear Models

The goal of this exercise is to perform sentiment analysis using the data from the demo, except using GloVe embeddings. Please see this week's demo for details on the data.

Download the data from here. Note that you will need an active Kaggle account to access the data.

We will use movie reviews from Rotten Tomatoes. The sentiment labels are: - 0 - negative - 1 - somewhat negative - 2 - neutral - 3 - somewhat positive - 4 - positive

[ ]:
```
import pandas as pd
import numpy as np
import torch
from torchvision.datasets import MNIST, FashionMNIST
from torch.nn.functional import cross_entropy
import time
import scipy.stats
```

[ ]:
```
class GloVeWordEmbedding:
    def __init__(self):
        self.idx_to_token, self.idx_to_vec = self._load_embedding()
        self.unknown_idx = 0
        self.token_to_idx = {token: idx for idx, token in
                             enumerate(self.idx_to_token)}

    def _load_embedding(self):
        idx_to_token, idx_to_vec = ['<unk>'], []
        with open('glove.6B.50d/vec.txt') as f: # use encoding='utf8' on widows
            for line in f:
                elems = line.rstrip().split(' ')
                token, elems = elems[0], [float(elem) for elem in elems[1:]]
                # Skip header information, such as the top row in fastText
                if len(elems) > 1:
                    idx_to_token.append(token)
                    idx_to_vec.append(elems)
        idx_to_vec = [[0] * len(idx_to_vec[0])] + idx_to_vec
```

```
        return idx_to_token, np.asarray(idx_to_vec)

    def __getitem__(self, tokens):
        # "tokens" is a list of words
        # use as object[tokens]
        # map token -> index -> vector
        indices = [self.token_to_idx.get(token, self.unknown_idx)
                   for token in tokens]
        vecs = self.idx_to_vec[np.asarray(indices)]
        return vecs

    def __call__(self, tokens):
        # Use as object(tokens)
        return self.__getitem__(tokens)

    def __len__(self):
        return len(self.idx_to_token)
```

[ ]:
```
glove_embedding = GloVeWordEmbedding()
```

[ ]:
```
filename = '../data/sentiment-analysis-on-movie-reviews/train.tsv'
# keep one example per sentence (original data labels each phrase)
data = pd.read_csv(filename, sep='\t').groupby('SentenceId').first()
data = data.drop(columns=['PhraseId'])

#Remove examples that are empty
data = data.replace(" ", np.nan)
data.dropna(axis=0, inplace=True)

data.head(4)
```

SentenceId	Phrase	Sentiment
1	A series of escapades demonstrating the adage …	1
2	This quiet , introspective and entertaining in…	4
3	Even fans of Ismail Merchant 's work , I suspe…	1
4	A positively thrilling combination of ethnogra…	3

1.1.1 Train-test split and featurize

[ ]:
```
data = data.sample(frac=1)  # shuffle
train_data = data[:7000]
test_data = data[7000:]
print(train_data.shape, test_data.shape)
```

```
(7000, 2) (1528, 2)
```

[ ]:
```
train_data["Phrase"].values[0]
```

```
'Clumsy , obvious , preposterous , the movie will likely set the cause of woman␣
↪warriors back decades .'
```

Let $\varphi(w)$ denote the embedding of word w. Recall that GloVe is a global embedding which does not depend on the context of the word.

For a piece of text denoted by words $T = (w_1, \cdots, w_n)$, we summarize it by the vector

$$\psi(T) := \frac{1}{n} \sum_{i=1}^{n} \varphi(w_i)$$

[ ]:
```
def featurize(x): # x is pd.Series with text
    # TODO: your code here
    # Return a matrix with the same number of rows as x
    # Each row contains one feature vector for the entire text
    matrix = np.zeros((x.shape[0], 50))

    for i,sentence in enumerate(x):
        vecs = glove_embedding[sentence.split()]
        matrix[i,:] = np.mean(vecs, axis=0)

    return matrix
```

[ ]:
```
x_train = featurize(train_data['Phrase'])
y_train = train_data['Sentiment'].values

x_test = featurize(test_data['Phrase'])
y_test = test_data['Sentiment'].values
```

1.1.2 Train a simple logistic regression classifier to test performance

[ ]:
```
# We will reduce the data dimensionality
# This step is optional and only perform it if you
# find that it helps the test accuracy
from sklearn.decomposition import PCA
pca = PCA(n_components=0.99, random_state=1).fit(x_train)  # keep 99% of the␣
↪explained variance
x_train = pca.transform(x_train)
x_test = pca.transform(x_test)
```

[ ]:
```
from sklearn.linear_model import LogisticRegressionCV, LogisticRegression


# TODO: Tune C with cross-validation
# clf = LogisticRegression(random_state=0, C=1.0).fit(x_train, y_train)
clf = LogisticRegressionCV(Cs = 10, cv=5, random_state=42).fit(x_train,y_train)
```

```
print("Best value of C = ", clf.C_[0])

# Re-fit model on entire training data with best value for C
model = LogisticRegression(random_state=0, C=clf.C_[0]).fit(x_train, y_train)

y_train_pred = model.predict(x_train)
y_test_pred = model.predict(x_test)

print('Train accuracy:', (y_train_pred == y_train).mean())
print('Test accuracy:', (y_test_pred == y_test).mean())
```

```
Best value of C =  2.782559402207126
Train accuracy: 0.3954285714285714
Test accuracy: 0.39986910994764396
```

2 Bonus Exercise 2:

The exercise: Perform the same McNemar test as in Part 2 but now compare the same ConvNet from week 2 to a ConvNet with just 1 hidden layer (of hidden size = 16). Use a significance $\alpha = 0.05$. Train each network with SGD for 30 passes with an appropriate learning rate.

[ ]:
```
# download dataset (~117M in size)
train_dataset = FashionMNIST('../data', train=True, download=False)
X_train = train_dataset.data # torch tensor of type uint8
y_train = train_dataset.targets # torch tensor of type Long
test_dataset = FashionMNIST('../data', train=False, download=False)
X_test = test_dataset.data
y_test = test_dataset.targets

# choose a subsample of 10% of the data:
idxs_train = torch.from_numpy(
    np.random.choice(X_train.shape[0], replace=False, size=X_train.shape[0]//
 ↪10))
X_train, y_train = X_train[idxs_train], y_train[idxs_train]
# idxs_test = torch.from_numpy(
#     np.random.choice(X_test.shape[0], replace=False, size=X_test.shape[0]//
 ↪10))
# X_test, y_test = X_test[idxs_test], y_test[idxs_test]

print(f'X_train.shape = {X_train.shape}')
print(f'n_train: {X_train.shape[0]}, n_test: {X_test.shape[0]}')
print(f'Image size: {X_train.shape[1:]}')

# Normalize dataset: pixel values lie between 0 and 255
# Normalize them so the pixelwise mean is zero and standard deviation is 1

X_train = X_train.float()  # convert to float32
```

```
X_train = X_train.view(-1, 784)
mean, std = X_train.mean(axis=0), X_train.std(axis=0)
X_train = (X_train - mean[None, :]) / (std[None, :] + 1e-6)  # avoid divide by␣
 ↪zero

X_test = X_test.float()
X_test = X_test.view(-1, 784)
X_test = (X_test - mean[None, :]) / (std[None, :] + 1e-6)

n_class = np.unique(y_train).shape[0]
```

```
X_train.shape = torch.Size([6000, 28, 28])
n_train: 6000, n_test: 10000
Image size: torch.Size([28, 28])
```

[ ]:
```
import torch

class ConvNet_2layer(torch.nn.Module):
    def __init__(self,num_classes=10):
        super().__init__()
        self.conv_ensemble_1 = torch.nn.Sequential(
            torch.nn.Conv2d(1, 16, kernel_size=5, padding=2),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2))
        self.conv_ensemble_2 = torch.nn.Sequential(
            torch.nn.Conv2d(16, 32, kernel_size=5, padding=2),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2))
        self.fc = torch.nn.Linear(7*7*32, 10)

    def forward(self, x):
        x = x.view(-1, 1, 28, 28)
        out = self.conv_ensemble_1(x)
        out = self.conv_ensemble_2(out)
        out = out.view(out.shape[0], -1)
        out = self.fc(out)
        return out

class ConvNet_1layer(torch.nn.Module):
    def __init__(self,num_classes=10):
        super().__init__()
        # TODO: Fill in for the 1-layer CNN, hidden size = 16
        self.conv_ensemble_1 = torch.nn.Sequential(
            torch.nn.Conv2d(1, 16, kernel_size=5, padding=2),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2))
        self.fc = torch.nn.Linear(14*14*16, 10)
```

```
    def forward(self, x):
        # TODO: Your Code Here
        x = x.view(-1, 1,28, 28)
        out = self.conv_ensemble_1(x)
        out = out.view(out.shape[0], -1)
        out = self.fc(out)
        return out

# Some utility functions to compute the objective and the accuracy
def compute_objective(model, X, y):
    score = model(X)
    # PyTorch's function cross_entropy computes the multinomial logistic loss
    return cross_entropy(input=score, target=y, reduction='mean')

def sgd_one_pass(model, X, y, learning_rate, verbose=False):
    num_examples = X.shape[0]
    average_loss = 0.0
    for i in range(num_examples):
        idx = np.random.choice(X.shape[0])
        # compute the objective.
        # Note: This function requires X to be of shape (n,d). In this case,␣
 ↪n=1
        objective = compute_objective(model, X[idx:idx+1], y[idx:idx+1])
        average_loss = 0.99 * average_loss + 0.01 * objective.item()
        if verbose and (i+1) % 100 == 0:
            print(average_loss)

        # compute the gradient using automatic differentiation
        gradients = torch.autograd.grad(outputs=objective, inputs=model.
 ↪parameters())

        # perform SGD update. IMPORTANT: Make the update inplace!
        for (w, g) in zip(model.parameters(), gradients):
            w.data -= learning_rate * g.data


from tqdm.auto import trange # range + progress bar
def sgd_n_passes(model, X_train, y_train, X_val, y_val, n_passes,␣
 ↪learning_rate):
    for i in trange(n_passes):
        sgd_one_pass(model, X_train, y_train, learning_rate)
    return compute_prediction_performance(model, X_val, y_val)

@torch.no_grad()
def compute_prediction_performance(model, X, y):
    # return a boolean vector of the same length as y
```

```
    # each entry is True if correctly predicted, else False
    # TODO: your code here
    score = model(X)
    preds = torch.argmax(score, dim=1)

    return preds == y
```

[ ]:
```
model1 = ConvNet_1layer()
performance1 = sgd_n_passes(
    model1, X_train, y_train, X_test, y_test, n_passes=30, learning_rate=2e-3
)
# boolean vector of length n_test
```

```
  0%|          | 0/30 [00:00<?, ?it/s]
```

[ ]:
```
model2 = ConvNet_2layer()
performance2 = sgd_n_passes(
    model2, X_train, y_train, X_test, y_test, n_passes=30, learning_rate=2.5e-3
)
# boolean vector of length n_test
```

```
  0%|          | 0/30 [00:00<?, ?it/s]
```

[ ]:
```
print('accuracy of ConvNet_1layer:', performance1.sum().item()/y_test.shape[0])
print('accuracy of ConvNet_2layer:', performance2.sum().item()/y_test.shape[0])
```

```
accuracy of ConvNet_1layer: 0.8633
accuracy of ConvNet_2layer: 0.8722
```

[ ]:
```
N01 = (~performance1 & performance2).sum().item()
N10 = (performance1 & ~performance2).sum().item()
```

[ ]:
```
T = (abs(N01-N10)-1)**2/(N10+N01)
threshold = scipy.stats.chi2(df=1).ppf(0.95)

print(f'Test statistic: {T}, threshold: {threshold}')

if T > threshold:
    print('Null rejected')
else:
    print('Failed to reject the null')
```

```
Test statistic: 9.790139064475348, threshold: 3.8414588206941205
Null rejected
```