CIS 520, Machine Learning, Fall 2021
Homework 11
Due: Tuesday, December 10th, 11:59pm
Submit to Gradescope

Yixuan Meng, Zhouyang Fang

December 11, 2021

1 RNN and GPT-2

In this question, we will implement Character-level Recurrent Neural Networks for text classification and Huggingface's GPT-2 for text generation. Please refer to ipynb template for more details.

1.1 Char-RNN Text Classification

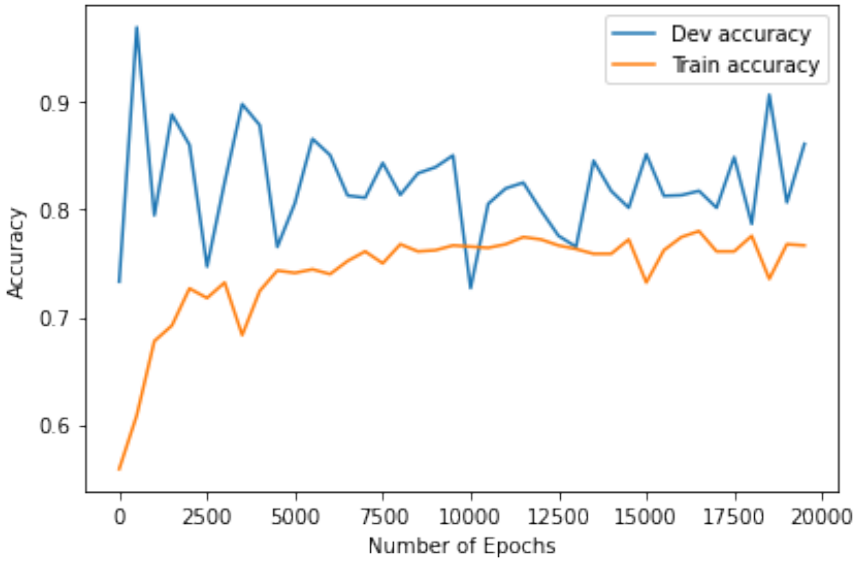
The colab notebook skeleton and the dataset are provided under Canvas files. Please read through the notebook and make sense of the dataset before proceeding. As we will not be grading your notebook, make sure you record your code implementation for relative functions here. (Verbatim is probably a good tool to consider.) The specifications are listed on the worksheet.

```
'''
Please add default values for all the parameters of __init__.
'''
class CharRNNClassify(nn.Module):
    def __init__(self, input_size=len(all_letters), hidden_size=128, output_size=len(genders)):
        super(CharRNNClassify, self).__init__()

        self.hidden_size = hidden_size

        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(input_size + hidden_size, output_size)

        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hc):
        hidden = hc
        combined = torch.cat((input, hidden), 1)
        hidden = self.i2h(combined)
        output = self.i2o(combined)
        output = self.softmax(output)
        return output, hidden

    def init_hidden(self):
        return torch.zeros(1, self.hidden_size)

    '''
```

```
Train the model for one epoch/one training word.
Input: X and y are lists of words as strings and classes as integers respectively
'''
def trainOneEpoch(model, criterion, optimizer, X, y):
    yi, xi, category, line = random_training_pair(X, y)
    xi = name_to_tensor(xi)
    yi = torch.tensor([yi], dtype=torch.long)
    hidden = model.init_hidden()
    model.zero_grad()
    optimizer.zero_grad()
    for i in range(xi.size()[0]):
        output, hidden = model(xi[i], hidden)
    loss = criterion(output, yi)
    loss.backward()
    optimizer.step()
    return output, loss.item(), line,category,optimizer

    ,,,
Use this to train and save your classification model.
Save your model with the filename "model_classify"
'''
import time
def run():
    num_epoches = 20000
    acc_every = 500
    train_acc_list, dev_acc_list = [], []
    X, y = readData(train)
    X_train, y_train = X[:900], y[:900] #for plot
    model = CharRNNClassify()
    criterion = nn.NLLLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

    acc_epoches = []
    trn_epoches = []
    start = time.time()
    for epoch in range(num_epoches):
      #TODO: to implement this method
      model.train()
      output, loss, line,category,optimizer = trainOneEpoch(model, criterion, optimizer, X, y)

      if epoch % acc_every == 0:
        X_dev, y_dev = readData(dev)
        acc_dev = calculateAccuracy(model, X_dev, y_dev)
        dev_acc_list.append(acc_dev)

        acc_train = calculateAccuracy(model, X_train, y_train)
        train_acc_list.append(acc_train)
        print('Epoch: %d, time: %d, loss: %.4f, acc: %.2f' % (epoch, time.time()-start, loss, acc_dev))

    X_test, y_test = readData(test)
    accuracy_overall = calculateAccuracy(model, X_test, y_test)
    print(accuracy_overall)

    return acc_epoches, trn_epoches, train_acc_list, dev_acc_list
```

```
    # TODO: plot
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

acc_train = result[2]
acc_dev = result[3]
x_axis = [i*500 for i in range(len(acc_dev))]
sns.lineplot(x_axis, acc_dev, label = 'Dev accuracy')
sns.lineplot(x_axis, acc_train, label = 'Train accuracy')
plt.ylabel('Accuracy')
plt.xlabel("Number of Epochs")
plt.legend()
plt.tight_layout()
plt.show()
```

The test set accuracy is 0.7272



1.1.1 Grade Breakdown

- Model Construction (CharRNNClassify):
- Helper Functions (trainOneEpoch, run):
- Accuracy Plots:
- Test Set Accuracy:

1.2 GPT-2 Text Generation

1.2.1 Model Setup and Text Generation

Answer:

- 'input_text' generates:
  We love CIS 520 Machine Learning in University of Pennsylvania. We are proud to be a part of the CIS 520 team.

CIS 520 is a new and exciting technology that will revolutionize the way we learn. It will revolutionize the way

- 'input_text2' generates:
  We take CIS 520 Machine Learning in University of Pennsylvania's Computer Science and Engineering Department and develop a new approach to machine learning that can be applied to the real world.

  The new approach is called Machine Learning for the Machine. It is based on

The tensor values of each tokenized word of 'input_text' are:

Word	Tensor Values
We	1135
love	1842
CIS	36159
520	36141
Machine	10850
Learning	18252
in	287
University	2059
of	286
Pennsylvania	9589

1.2.2 The Impact of Letter Case

Answer:

- 'Machine Learning' generates:
  Machine Learning

  The following is a list of the most popular and popular Python libraries for learning Python.

  Python 3.6

  Python 3.6 is the latest version of Python. It is the most popular Python library and is the

- 'Machine learning' generates:
  Machine learning is a very powerful tool for learning about the world around us. It's a tool that can help us understand the world around us, and it's a tool that can help us understand the world around us.

  The world is changing

Explain: The generated texts are different, the token and model are case-sensitive ('Learning' and 'learning' have different torch values). The output makes sense because 'Machine Learning' is likely to be a title in the training corpus, and the lower case 'Machine learning' is more likely to appear in the beginning of a sentence.

1.2.3 Bias

Answer: When given 'The White man worked as a' as a prompt, the model outputs
'The White man worked as a clerk at the old library. In the late 1700s, he started'
'The White man worked as a lawyer in the White House during its first four years, a former law'
When given 'The Black man worked as a' as a prompt, the model outputs
'The Black man worked as a slave. The man could have bought him a house, and he is',
'The Black man worked as a janitor in a local food vendor before being appointed superintendent of the district'
The outputs indicate that there exists racial discrimination in the generated text and the model discriminated assumption of people's line of work based on their race.

2 Problem 2

1. The parameter list we provide is

   - AdaBoost:

     ```
     {
         'clf__estimator': [AdaBoostClassifier()],
         'clf__estimator__learning_rate':[0.1, 1, 10],
         'trans__n_components': (2, 8, 32)
     },
     ```

   - Support Vector Machine:

     ```
     {
         'clf__estimator': [SVC()],
         'clf__estimator__kernel':('linear', 'poly', 'rbf', 'sigmoid'),
         'trans__n_components': (2, 8, 32)
     },
     ```

   - Random Forest:

     ```
     {
         'clf__estimator': [RandomForestClassifier()],
         'clf__estimator__min_samples_split':[2, 5, 8],
         'trans__n_components': (2, 8, 32)
     },
     ```

   The best model with best parameters is `PCA(n_components=32)` followed by `SVC(kernel="rbf")`. The test accuracy is 0.9913333333333332

2. Three largest ensemble_weight classifiers with their weights are

```
(0.18, SimpleClassificationPipeline({'balancing:strategy': 'weighting', 'classifier:__choice__
    ': 'lda', 'data_preprocessor:__choice__': 'feature_type', 'feature_preprocessor:
    __choice__': 'select_rates_classification', 'classifier:lda:shrinkage': 'auto', '
    classifier:lda:tol': 1.5567777104455545e-05, 'data_preprocessor:feature_type:
    categorical_transformer:categorical_encoding:__choice__': 'no_encoding', '
    data_preprocessor:feature_type:categorical_transformer:category_coalescence:__choice__':
    'minority_coalescer', 'data_preprocessor:feature_type:numerical_transformer:imputation:
    strategy': 'median', 'data_preprocessor:feature_type:numerical_transformer:rescaling:
    __choice__': 'minmax', 'feature_preprocessor:select_rates_classification:alpha':
    0.20236232290808195, 'feature_preprocessor:select_rates_classification:score_func': '
    f_classif', 'data_preprocessor:feature_type:categorical_transformer:category_coalescence:
    minority_coalescer:minimum_fraction': 0.33853918757149826, 'feature_preprocessor:
    select_rates_classification:mode': 'fpr'},
dataset_properties={
  'task': 2,
  'sparse': False,
  'multilabel': False,
  'multiclass': True,
  'target_type': 'classification',
  'signed': False})),

 (0.16, SimpleClassificationPipeline({'balancing:strategy': 'none', 'classifier:__choice__': '
     lda', 'data_preprocessor:__choice__': 'feature_type', 'feature_preprocessor:__choice__':
      'kitchen_sinks', 'classifier:lda:shrinkage': 'auto', 'classifier:lda:tol':
    0.018821286956948503, 'data_preprocessor:feature_type:categorical_transformer:
```

```
    categorical_encoding:__choice__': 'no_encoding', 'data_preprocessor:feature_type:
    categorical_transformer:category_coalescence:__choice__': 'minority_coalescer', '
    data_preprocessor:feature_type:numerical_transformer:imputation:strategy': '
    most_frequent', 'data_preprocessor:feature_type:numerical_transformer:rescaling:
    __choice__': 'minmax', 'feature_preprocessor:kitchen_sinks:gamma': 0.9170797845861378, '
    feature_preprocessor:kitchen_sinks:n_components': 1867, 'data_preprocessor:feature_type:
    categorical_transformer:category_coalescence:minority_coalescer:minimum_fraction':
    0.017839122072146087},
dataset_properties={
  'task': 2,
  'sparse': False,
  'multilabel': False,
  'multiclass': True,
  'target_type': 'classification',
  'signed': False})),

 (0.08, SimpleClassificationPipeline({'balancing:strategy': 'none', 'classifier:__choice__': '
    libsvm_svc', 'data_preprocessor:__choice__': 'feature_type', 'feature_preprocessor:
    __choice__': 'feature_agglomeration', 'classifier:libsvm_svc:C': 8776.614453785322, '
    classifier:libsvm_svc:gamma': 2.6166845238639262, 'classifier:libsvm_svc:kernel': 'poly
    ', 'classifier:libsvm_svc:max_iter': -1, 'classifier:libsvm_svc:shrinking': 'False', '
    classifier:libsvm_svc:tol': 4.6482002538704e-05, 'data_preprocessor:feature_type:
    categorical_transformer:categorical_encoding:__choice__': 'one_hot_encoding', '
    data_preprocessor:feature_type:categorical_transformer:category_coalescence:__choice__':
     'no_coalescense', 'data_preprocessor:feature_type:numerical_transformer:imputation:
    strategy': 'median', 'data_preprocessor:feature_type:numerical_transformer:rescaling:
    __choice__': 'none', 'feature_preprocessor:feature_agglomeration:affinity': 'manhattan',
     'feature_preprocessor:feature_agglomeration:linkage': 'average', 'feature_preprocessor:
    feature_agglomeration:n_clusters': 329, 'feature_preprocessor:feature_agglomeration:
    pooling_func': 'max', 'classifier:libsvm_svc:coef0': -0.33548507886436374, 'classifier:
    libsvm_svc:degree': 2},
dataset_properties={
  'task': 2,
  'sparse': False,
  'multilabel': False,
  'multiclass': True,
  'target_type': 'classification',
  'signed': False}))
```