UVA CS 4774

HW4 Review

Presented By: Zhe Wang

Professor: Dr. Yanjun Qi

November 24, 2020

Common errors in HW4

Data Preprocessing for Naive Bayes Classifier

- Tokenize the document
- Remove stopping words
- Stemming

Example:

```
def read_text_from_file(...) :
   review = ...
   word_tokens = tokenizer.tokenize(review.lower())
   word_tokens = [w for w in word_tokens if not w in stop_words]
   word_tokens = [ps.stem(w) for w in word_tokens]
   return word_tokens

# ['actual', 'fan', 'origin', '1961', 'live', 'action', 'disney', 'flick', 'name', 'star', 'hayley',
'mill', 'twice', 'pair', ...]
```

Attention:

The correct preprocessing will remove punctuations.

Multinomial Naive Bayes Classifier

During training:

- Classify all training data into PosData and NegData

```
# shape of PosData is [700, 100]
# shape of NegData is [700, 100]
```

- Calculate the frequency of each word in both PosData and NegData

```
PosWordsSum = np.sum(PosData, axis=0)   # [1, 100]
NegWordsSum = np.sum(NegData, axis=0)  # [1, 100]
```

- Parameter estimation (with smoothing)

$$p(word[i] \mid y = 1) = thetaPos[i] = \frac{PosWordsSum[i] + \alpha}{sum(PosWordsSum) + \alpha VocSize}$$

A quick way to check your code: sum(thetaPos)=1, sum(thetaNeg)=1

During testing:

$$X_{test} = [a_1, a_2, \cdots, a_{100}]$$

$$P(Y = 1 \,|\, X_{test}) \propto P(Y = 1)\theta_{pos,1}^{a_1}\theta_{pos,2}^{a_2}\cdots\theta_{pos,100}^{a_{100}}$$

$$P(Y = 0 \,|\, X_{test}) \propto P(Y = 0)\theta_{neg,1}^{a_1}\theta_{neg,2}^{a_2}\cdots\theta_{neg,100}^{a_{100}}$$

For numerical stability, log is suggested.

```
IN: 0.9**10000 == 0
Out: True
```

$$log(P(Y = 1 \,|\, X_{test})) \propto log(P(Y = 1)) + a_1 \log(\theta_{pos,1}) + a_2 \log(\theta_{pos,2}) + \cdots + a_{100} \log(\theta_{pos,100})$$

$$log(P(Y = 0 \,|\, X_{test})) \propto log(P(Y = 0)) + a_1 \log(\theta_{neg,1}) + a_2 \log(\theta_{neg,2}) + \cdots + a_{100} \log(\theta_{neg,100})$$

```
prob_pos = sum(log(thetaPos) * Xtest[i]) + log(0.5)
prob_neg = sum(log(thetaNeg) * Xtest[i]) + log(0.5)
if prob_pos > prob_neg:
    yPredict[i] = 1
else:
    yPredict[i] = 0
```

Bernoulli Naive Bayes Classifier

During testing:

- Binarize the frequency

$$X_{test} = [a_1, a_2, \cdots, a_{100}] \Longrightarrow \hat{X}_{test} = [1,0,\cdots,1]$$

- Calculate the conditional probability

$$P(Y=1 \mid X_{test}) \propto P(Y=1)\theta_{pos,1}(1-\theta_{pos,2})\cdots\theta_{pos,100}$$

$$P(Y=0 \mid X_{test}) \propto P(Y=0)\theta_{neg,1}(1-\theta_{neg,2})\cdots\theta_{neg,100}$$

- Comparison

```
for i in range(len(Xtest)):
  pos_log = log(0.5),    neg_log = log(0.5)
  for j in range(len(Xtest[i])):
    if Xtest[i][j] > 0:        # binarize, then calculate the conditional probability
      p_pos = log(thetaPosTrue[j]),    p_neg = log(thetaNegTrue[j])
    else:
      p_pos = log(1 - thetaPosTrue[j]),    p_neg = log(1 - thetaNegTrue[j])
    pos_log += p_pos
    neg_log += p_neg
# comparison
```

Model fine-tuning

- **What** is model fine-tuning?

  Model has already been trained on a specific dataset and a specific task. You want to slightly tweak it to adapt to your dataset and task.

  BERT is pertained on BooksCorpus (800M words) and English Wikipedia (2,500M words).

- **When** do we use model fine-tuning?

  The dataset you care about is relatively small.

  BERT base uncased: number of trainable parameters ~ 110M

  (In HW4, we have only 1400 training data)

- **How**? Both TensorFlow and PyTorch provide pertained models in their model zoo.

  resnet50 = models.resnet50(pretrained=True)

  resnet50 = tf.keras.applications.ResNet50(weights='imagenet')

- **Advice I:** Starting from a relatively small learning rate when you are fine-tuning a model

- **Advice II**: In your final project, if you use state-of-the-art models, starting from pertained model.

Common errors in HW2/HW3/HW4: Procedures for model selection

```
1. load train_set, test_set

2. new_train_set, new_valid_set = data_split(train_set)

3. valid_losses = []

4. for hyper_param in hyper_param_set:
        theta* = model_train(new_train_set, hyper_param)
        valid_loss = model_eval(new_valid_set, theta*, hyper_param)
        valid_losses.append(valid_loss)

5. best_hyper = argmin(valid_losses)

6. theta* = model_train(train_set, best_hyper)

7. reported_performace = model_eval(test_set, theta*, best_hyper)
```

If there is no model selection, only 1, 6, 7 will be remained.

Potential issues in HW5

Correct Procedures for model selection

```
1.  load train_set, test_set

2. new_train_set, new_valid_set = data_split(train_set)

3. valid_losses = []

4. for hyper_param in hyper_param_set:
        theta* = model_train(new_train_set, hyper_param)
        valid_loss = model_eval(new_valid_set, theta*, hyper_param)
        valid_losses.append(valid_loss)

5. best_hyper = argmin(valid_losses)

6. theta* = model_train(train_set, best_hyper)

7. reported_performace = model_eval(test_set, theta*, best_hyper)
```

Potential issues

- In the unlabeled sample set ”salary.2Predict.csv”, last column includes a fake field for class labels. You are required to generate/ predict labels for samples in ”salary.2Predict.csv”. **Remember not to shuffle your prediction outputs!**

- **Include the CV classification accuracy results in** your pdf report by performing **3-fold cross validation (CV)** on the labeled set ”salary.labeled.csv” (including about 38k samples) using **at least three different SVM kernels** you pick.

- **Provide details** about the kernels you have tried and their performance (e.g. 3CV classification accuracy) including results on **both CV train accuracy and CV test accuracy into the writing.** (**Highly recommended**: table with each row containing kernel choice, kernel parameter, CV train accuracy and CV test accuracy)**.**

- **Feel free to use some existing libraries for easier data pre-processing.**