Q1:

Training Accuracy: 0.9715
Dev Accuracy: 0.7565

Q2 Comparing to the Training Accuracy and Dev Accuracy we will find out that the training accuracy is always larger than the Dev accuracy. That's because the model is trained using the training dataset and the goal of the model is to make sure the train data set are as correct as possible. At the same time the train accuracy cannot be 100% for there will be overfit for the model and it will be not good for the Dev data set. The Dev dataset is totally unknown for the model so the accuracy will be lower.

Q3

Class 0 (World News) Top 5 tokens: ['...', '-', 'afp', 'minister', 'president']

Class 1(Sports): Top 5 tokens: ['saturday', 'team', 'sunday', 'night', 'olympic']

Class 2(Business): Top 5 tokens: ['company', 'oil', 'business', 'reuters', 'inc.']

Class 3(Tech/Science): Top 5 tokens: ['ap', 'scientists', 'space', 'space.com', '\\\\']

As the result shown above, we can get that the "minister" and "president" are the words that pretty related to the world news which means when we see these kind of words, we can highly put it into world news part.

Q4

Train Accuracy: 0.9715

Dev Accuracy: 0.7518

Time elapsed: 2.90 seconds

```
(base) yinghuang@YingdeMacBook-Air hw0 % python /Users/yinghuang/Desktop/BU/NLP/hw0/code/lang_classifier.py --mode
l LR
Loading data...
Loaded 1650 examples from /Users/yinghuang/Desktop/BU/NLP/hw0/data/train.tsv
Loaded 850 examples from /Users/yinghuang/Desktop/BU/NLP/hw0/data/dev.tsv
Vocabulary built with 5000 words.
Training logistic regression...
Epoch 1, Loss: 0.7813
Epoch 2, Loss: 0.5141
Epoch 3, Loss: 0.4158
Epoch 4, Loss: 0.3508
Epoch 5, Loss: 0.3064
Epoch 6, Loss: 0.2706
Epoch 7, Loss: 0.2462
Epoch 8, Loss: 0.2238
Epoch 9, Loss: 0.2059
Epoch 10, Loss: 0.1904
Train Accuracy: 0.9715
Dev Accuracy: 0.7518
Time elapsed: 2.90 seconds
```

Q5:

Vocabulary size	Epochs	Learning rate	Train Accuracy	Dev Accuracy	Time
5000	10	0.01	0.9715	0.7518	2.90s
5500	10	0.01	0.9745	0.7529	2.90s
6000	10	0.01	0.9727	0.7482	2.91s
6500	10	0.01	0.9800	0.7482	3.02s
7000	10	0.01	0.9788	0.7471	3.04s
5000	15	0.01	0.9879	0.7553	4.11s
5000	20	0.01	0.9900	0.7612	5.36s
5000	25	0.01	0.9964	0.7541	6.70s
5000	30	0.01	0.9988	0.7529	8.09s
5000	10	0.02	0.9921	0.7506	2.88s
5000	10	0.03	0.9988	0.7541	2.87s
5000	10	0.04	0.9988	0.7600	2.87s
5000	10	0.05	0.9994	0.7565	2.86s

The best performance I achieved was a Development Accuracy of 0.7612 with a Training Accuracy of 0.9900. The hyperparameters used were Vocabulary Size = 5000, Learning Rate = 0.01, and Epochs = 20.

When the learning rate was higher, the model converged much faster, but the development accuracy did not improve further, maybe due to overfitting. If the learning rate were too low, the model would fail to converge within the limited epochs; if too high, the loss would oscillate and fail to decrease.

This occurs because the learning rate determines the step size in gradient descent. A large step size speeds up learning but risks overshooting the optimal weights, while a small step size ensures stability but results in very slow convergence.

Q6

(a)Training Accuracy: 0.9527

Development Accuracy: 0.6741

```
(base) yinghuang@YingdeMacBook-Air hw0 % python /Users/yinghuang/Desktop/BU/
NLP/hw0/code/lang_classifier.py --model BIGRAM
Loading data...
Loaded 1650 examples from /Users/yinghuang/Desktop/BU/NLP/hw0/data/train.tsv
Loaded 850 examples from /Users/yinghuang/Desktop/BU/NLP/hw0/data/dev.tsv
Training logistic regression...
Epoch 1, Loss: 0.8943
Epoch 2, Loss: 0.6352
Epoch 3, Loss: 0.5215
Epoch 4, Loss: 0.4458
Epoch 5, Loss: 0.3905
Epoch 6, Loss: 0.3486
Epoch 7, Loss: 0.3153
Epoch 8, Loss: 0.2878
Epoch 9, Loss: 0.2658
Epoch 10, Loss: 0.2466
Train Accuracy: 0.9527
Dev Accuracy: 0.6741
Time elapsed: 3.89 seconds
```

(b)Bigrams create a huge number of possible features, most of which appear very sparsity. This causes the model to memorize specific phrases unique to the training data instead of learning general patterns. As a result, it overfits the training set and fails to generalize to the new examples in the development set.

Q7

(a) Feature Description & Results

Feature 1 Description: Symbol Count (Business Indicators). I counted the frequency of currency symbols (like $, €, £) and percentage signs (%) in the text. I think that these symbols appear significantly more often in Business news compared to World News or Sports.

Feature 2 Description: Average Word Length. I calculated the average length of all tokens in an example. My thought was that Tech/Science and World News articles tend to use longer, more complex vocabulary such as the long technical word, whereas Sports articles tend to use shorter, more conversational words.

Code Location: See the CustomFeaturizer class in models.py.

Development Accuracy: 0.7353

```
(base) yinghuang@YingdeMacBook-Air hw0 % python /Users/yinghuang/Desktop/BU/NLP/hw0/code/lang_classifier.py --mode
l CUSTOM
Loading data...
Loaded 1650 examples from /Users/yinghuang/Desktop/BU/NLP/hw0/data/train.tsv
Loaded 850 examples from /Users/yinghuang/Desktop/BU/NLP/hw0/data/dev.tsv
Vocabulary built with 5000 words.
Custom Featurizer: Vocab size increased to 5002 (added 2 custom features)
Training logistic regression...
Epoch 1, Loss: 0.7649
Epoch 2, Loss: 0.5263
Epoch 3, Loss: 0.4248
Epoch 4, Loss: 0.3586
Epoch 5, Loss: 0.3130
Epoch 6, Loss: 0.2767
Epoch 7, Loss: 0.2511
Epoch 8, Loss: 0.2272
Epoch 9, Loss: 0.2091
Epoch 10, Loss: 0.1928
Train Accuracy: 0.9533
Dev Accuracy: 0.7353
Dev macro-F1: 0.4527
Time elapsed: 4.01 seconds
```

(b) The performance of my custom featurizer (0.7353) is slightly lower than the original Bag-of-Words model (0.7494). This might because that the new features might be redundant or introduced some noise. For example, the Bag-of-Words model likely already captures business contexts through words like 'money' or 'market', making the '$' symbol less critical. Also, 'average word length' might vary too much within classes to be a reliable predictor.

Q8

(a) Macro F1 Score on Dev Set: 0.5070

```
(base) yinghuang@YingdeMacBook-Air hw0 % python /Users/yinghuang/Desktop/BU/NLP/hw0/code/lang_classifier.py --mode
l LR
Loading data...
Loaded 1650 examples from /Users/yinghuang/Desktop/BU/NLP/hw0/data/train.tsv
Loaded 850 examples from /Users/yinghuang/Desktop/BU/NLP/hw0/data/dev.tsv
Vocabulary built with 5000 words.
Training logistic regression...
Epoch 1, Loss: 0.7761
Epoch 2, Loss: 0.5205
Epoch 3, Loss: 0.4156
Epoch 4, Loss: 0.3509
Epoch 5, Loss: 0.3058
Epoch 6, Loss: 0.2719
Epoch 7, Loss: 0.2441
Epoch 8, Loss: 0.2243
Epoch 9, Loss: 0.2057
Epoch 10, Loss: 0.1910
Train Accuracy: 0.9721
Dev Accuracy: 0.7529
Dev macro-F1: 0.5070
Time elapsed: 2.93 seconds
```

(b) Yes, there is a significant difference. My accuracy (0.7529) is much higher than my macro F1 score (0.5070). This large discrepancy indicates that the dataset is highly imbalanced, and the classifier is performing well on the majority classes but poorly on the minority classes. Accuracy is misleading here because it is dominated by the frequent classes. The low Macro F1 score reveals that the model fails to generalize effectively to the under-represented categories, as Macro F1 treats all classes equally regardless of their size.