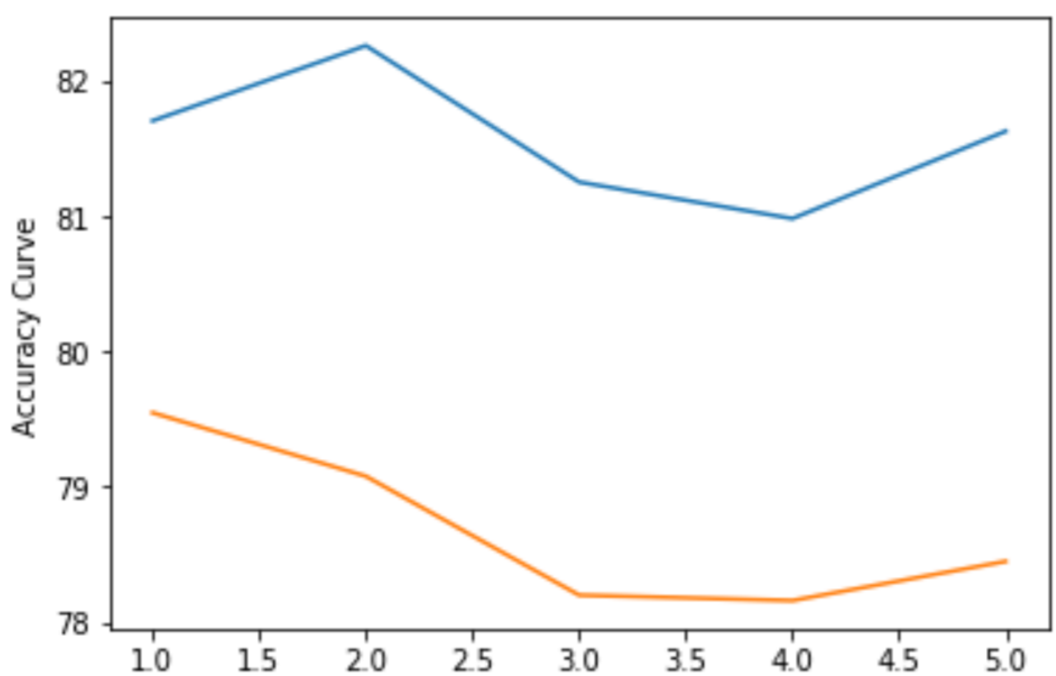
Perceptron Algorithm

1. Implement the feature function f(x; y) with the bag-of-words representations discussed in lecture 2. You can use some preprocessing tricks to reduce the vocabulary size, such as (1) convert all characters into lowercase, and (2) discard low-frequency words or map them into a special token unk.

For this I have taken all the Train set, Dev set and Test set data in variables – trn, trn_label_int, dev, dev_label_int, tst respectively. I imported stopwords and words from from nltk.corpus to filter out stopwords and filter out words not present in words. Also while filtering, I removed special characters and numbers and converted each text to lowercase. I converted the resultant main_list to a dataframe to calculate the frequency of each word/text and added the frequency as a column to the dataframe by grouping on each unique word. Then finally I removed low frequency words, i.e., frequency<4, and high frequency words, i.e, frequency>5000. The resultant vector/feature set was 7715

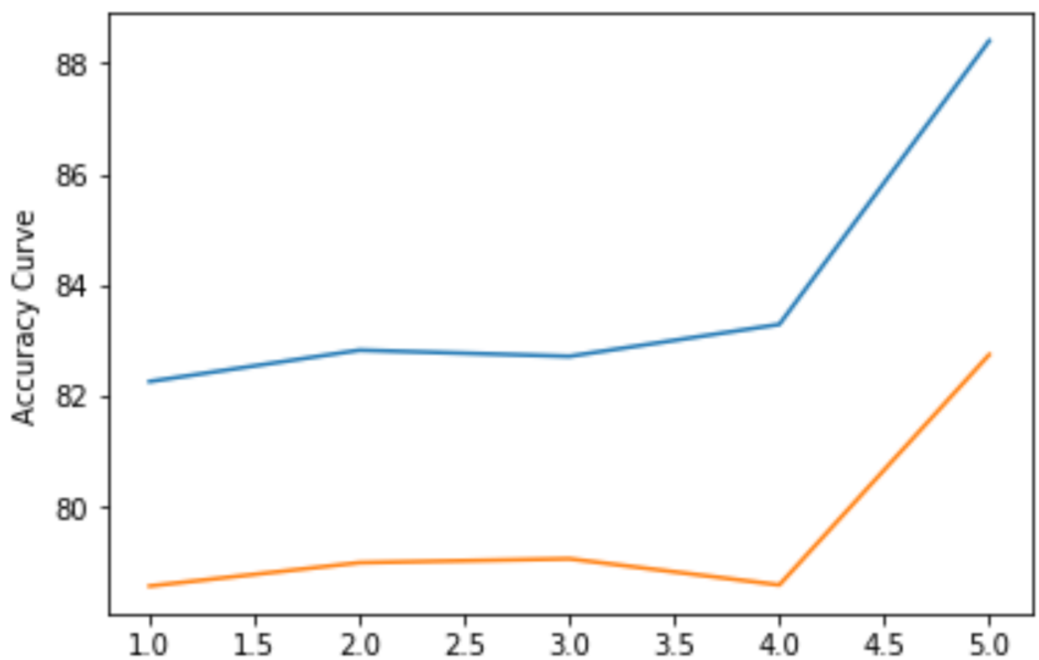
2. Implement the Perceptron algorithm described in JE section 2.2.1 Algorithm 3.
_ Plot the accuracy curves on both training and development sets per training epochs, for at least 5 training epochs.

```
from matplotlib import pyplot as plt
epoch=[1,2,3,4,5]
plt.ylabel('Accuracy Curve')
plt.plot(epoch, train_accuracy)
plt.plot(epoch, dev_accuracy)
plt.show()
```



3. Implement the averaged Perceptron algorithm described in JE section 2.2.2.
Plot the accuracy curves on both training and development sets per training epochs, for at least 5 training epochs.

```
epoch=[1,2,3,4,5]
plt.ylabel('Accuracy Curve')
plt.plot(epoch, train_avg_accuracy)
plt.plot(epoch, dev_avg_accuracy)
plt.show()
```



Logistic Regression

1. Use both the LogisticRegression function and the CountVectorizer function with their default settings to train a classifier and report
_ the size of your feature set - 47963
_ the classification accuracy on both training and development sets –

- Train_accuracy * 100 -> 97.82333333333332
- Dev_accuracy * 100 -> 88.0

2. Change the argument ngram range in function CountVectorizer from (1; 1) to (1; 2), then re-train your classi_er with this large feature set. Report
_ the size of your feature set - 776704
_ the classification accuracy on both training and development sets –

- Train_accuracy * 100 -> 99.99333333333334
- Dev_accuracy * 100 -> 90.44

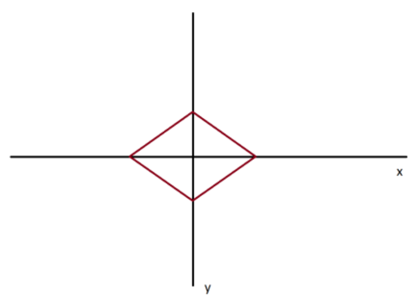
3. The regularization parameter lamda =1/c in the LogisticRegression function is 1:0. In practice, we need to tune this parameter in order to find the best model. Try different lamda's with the rich feature set built in step 2. For all the lamda's, report the corresponding classification accuracy on both training and development sets.

For lamda in [0.961, 0.963, 0.965, 0.967, 0.969, 0.971, 0.973] –

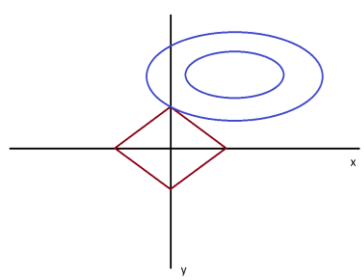
Train Accuracy	Dev accuracy
99.99333333333334	90.45
99.99333333333334	90.45
99.99333333333334	90.46
99.99333333333334	90.46
99.99333333333334	90.45
99.99333333333334	90.45
99.99333333333334	90.45

4. Similar to L2 regularization, L1 regularization adds the parameter constraint with its L1 norm. Theoretically, L1 regularization tends to give sparse solutions, which means most of components in theta will be 0 or close to 0.
_In lecture 3, we used contour plots to explain how L2 works. Please use a similar way to explain why L1 regularization prefers sparse solutions.

L1 regularization makes the vector x smaller(sparse) as most of its elements would be zero/useless and rest elements non-zero and useful. L1 is defined as sum of absolute values of a vectors all elements,i.e, |x|+|y|. If we draw all points that have a L1 norm equals to a constant c, it should be something like this –



Only on the axis, the points are sparse, i.e, either x or y is zero.



I you notice, the probability of touching a tip is very high. Lp norm when 0 <= p < 1 gives the best result. This reduces the problem of over-fitting.

_Try different lamda's and report the corresponding classification accuracy on both training and development sets.

For lamda's in [0.963,0.965,0.967,0.968] and penalty='l1', we get –

Train Accuracy	Dev accuracy
99.29333333333334	89.78
99.29	89.81
99.28333333333333	89.80
99.28	89.80

5. In lecture 3, we talked different ways to refine the feature set in order to obtain a better classification performance on the development set. Together with the options provided in the LogisticRegression, please try different combinations of these tricks/arguments and _nd the best model as you can do. Classification accuracy will be an important criterion for evaluating your answers here.
_ Report the accuracy on both training and development sets with your best model, and explain how you obtain this model.

Train Accuracy	Dev accuracy
99.91	88.41
99.91	88.42
99.90666666666667	88.41
99.90333333333334	88.40
97.45333333333333	88.39

1. While vectorizing, via CountVectorize, converted all text into lowercase, removed words with frequency <=2 , ngram_range=(1,3) and removed stop_words. This gave me a feature set of 224424.
2. Selected lamda in lamda5=[0.94, 0.95, 0.968, 1, 8.5]
3. For the classifier used C=1/lamda and solver='lbfgs' like below -
   classifier5[i] = LR(C=c_list5[i],solver="lbfgs")