1a.

1a) 1. What is the difference between increasing circle by 2 and decreasing ~~it~~ triangle by 2?

Solution: There is no difference between increasing circle by 2 and decreasing triangle by 2.

When θ_{circle} is increased by 2,

$$p(\text{solid circle}) = \frac{e^{(\theta_{circle} + \theta_{solid})} \cdot e^2}{e^{(\theta_{circle} + \theta_{solid})} \cdot e^2 + e^{(\theta_{circle} + \theta_{striped})} \cdot e^2 + e^{(\theta_{triangle} + \theta_{solid})} + e^{(\theta_{triangle} + \theta_{striped})}}$$

~~p(striped cir~~

When $\theta_{triangle}$ is decreased by 2

$$p(\text{solid circle}) = \frac{e^{(\theta_{circle} + \theta_{solid})}}{e^{(\theta_{circle} + \theta_{solid})} + e^{(\theta_{circle} + \theta_{striped})} + e^{(\theta_{triangle} + \theta_{solid})} \cdot e^{-2} + e^{(\theta_{triangle} + \theta_{striped})} e^{-2}}$$

Multiplying numerator and denominator by e^2

$$p(\text{solid circle}) = \frac{e^{(\theta_{circle} + \theta_{solid})} \cdot e^2}{e^{(\theta_{circle} + \theta_{solid})} \cdot e^2 + e^{(\theta_{circle} + \theta_{striped})} \cdot e^2 + e^{(\theta_{triangle} + \theta_{solid})} + e^{(\theta_{triangle} + \theta_{striped})}}$$

which is same as when θ_{circle} is increased by 2.

The same holds for other probabilities

2. What happens if you increase both circle and triangle by the same amount?

Solution: Nothing happens if we increase both circle and triangle by the same amount.

Let's assume we increase both θ_{circle} and $\theta_{triangle}$ by 2,

then,

$$p(\text{solid circle})' = \frac{e^{(\theta_{circle} + \theta_{solid}) \cdot e^2}}{e^{(\theta_{circle} + \theta_{solid}) \cdot e^2} + e^{(\theta_{circle} + \theta_{striped}) \cdot e^2} + e^{(\theta_{triangle} + \theta_{solid}) \cdot e^2} + e^{(\theta_{triangle} + \theta_{striped}) \cdot e^2}}$$

$$= \frac{e^{(\theta_{circle} + \theta_{solid})}}{e^{(\theta_{circle} + \theta_{solid})} + e^{(\theta_{circle} + \theta_{striped})} + e^{(\theta_{triangle} + \theta_{solid})} + e^{(\theta_{triangle} + \theta_{striped})}}$$

$$= p(\text{solid circle})$$

Same holds for all probabilities.

3. Does it help at all to have 4 sliders, or could you do just as well with 2 of them? Which 2?

Solution: There is no need for 4 sliders. Everything can be done with 2 sliders.

$$p(\text{solid circle}) = \frac{e^{(\theta_{solid} + \theta_{circle})}}{e^{(\theta_{circle} + \theta_{solid})} + e^{(\theta_{circle} + \theta_{striped})} + e^{(\theta_{triangle} + \theta_{solid})} + e^{(\theta_{triangle} + \theta_{striped})}}$$

$$p(\text{striped circle}) = \frac{e^{(\theta_{circle} + \theta_{striped})}}{e^{(\theta_{circle} + \theta_{solid})} + e^{(\theta_{circle} + \theta_{striped})} + e^{(\theta_{triangle} + \theta_{solid})} + e^{(\theta_{triangle} + \theta_{striped})}}$$

$$p(\text{solid triangle}) = \frac{e^{(\theta_{triangle} + \theta_{solid})}}{e^{(\theta_{circle} + \theta_{solid})} + e^{(\theta_{circle} + \theta_{striped})} + e^{(\theta_{triangle} + \theta_{solid})} + e^{(\theta_{triangle} + \theta_{striped})}}$$

$$p(\text{striped triangle}) = \frac{e^{(\theta_{triangle} + \theta_{striped})}}{e^{(\theta_{circle} + \theta_{solid})} + e^{(\theta_{circle} + \theta_{striped})} + e^{(\theta_{triangle} + \theta_{solid})} + e^{(\theta_{triangle} + \theta_{striped})}}$$

Dividing numerator and denominator by $e^{(\theta_{triangle} + \theta_{striped})}$,

$$p(\text{solid circle}) = \frac{e^{(\theta_{solid} + \theta_{circle} - \theta_{triangle} - \theta_{striped})}}{e^{(\theta_{solid} + \theta_{circle} - \theta_{triangle} - \theta_{striped})} + e^{(\theta_{circle} - \theta_{triangle})} + e^{(\theta_{solid} - \theta_{striped})} + 1}$$

$$p(\text{striped circle}) = \frac{e^{(\theta_{circle} - \theta_{triangle})}}{e^{(\theta_{circle} + \theta_{solid} - \theta_{triangle} - \theta_{striped})} + e^{(\theta_{circle} - \theta_{triangle})} + e^{(\theta_{solid} - \theta_{striped})} + 1}$$

$$p(\text{solid triangle}) = \frac{e^{(\theta_{triang}} e^{(\theta_{solid} - \theta_{striped})}}{e^{(\theta_{circle} + \theta_{solid} - \theta_{triangle} - \theta_{striped})} + e^{(\theta_{circle} - \theta_{triangle}} + e^{(\theta_{solid} - \theta_{striped})} + 1}$$

$$p(\text{striped triangle}) = \frac{1}{e^{(\theta_{circle} + \theta_{solid} - \theta_{triangle} - \theta_{striped})} + e^{(\theta_{circle} - \theta_{triangle})} + e^{(\theta_{solid} - \theta_{striped})} + 1}$$

Let $\theta_{circle} - \theta_{triangle}$ be α_{circle}

$\theta_{solid} - \theta_{striped}$ be α_{solid}.

$$p(\text{solid circle}) = \frac{e^{(\alpha_{circle} + \alpha_{solid})}}{e^{(\alpha_{circle} + \alpha_{solid})} + e^{(\alpha_{circle})} + e^{(\alpha_{solid})} + 1}$$

$$p(\text{striped circle}) = \frac{e^{(\alpha_{circle})}}{e^{(\alpha_{circle} + \alpha_{solid})} + e^{(\alpha_{circle})} + e^{(\alpha_{solid})} + 1}$$

$$p(\text{solid triangle}) = \frac{e^{(\alpha_{solid})}}{e^{(\alpha_{circle} + \alpha_{solid})} + e^{(\alpha_{circle})} + e^{(\alpha_{solid})} + 1}$$

$$p(\text{striped triangle}) = \frac{1}{e^{(\alpha_{circle} + \alpha_{solid})} + e^{(\alpha_{circle})} + e^{(\alpha_{solid})} + 1}$$

Thus, proved 2 sliders are enough.

Any pair of sliders, except 1) circle and triangle and 2) solid and striped are enough.

1b.

We know that the derivative of the log likelihood with respect to a parameter is simply the difference of the observed feature count and the expected feature count. Hence feature matching is always possible by making the observed feature count equal to the expected feature counts and calculating the feature weights. Note that this maximizes the log likelihood as well since we are making the derivative zero. Hence this implies that a good move in the log likelihood game implies a good move in the matching game and vice versa.

1c.

In the previous case, if we write down the equations of the probabilities of each of the shapes (solid circle, striped circle, solid triangle and striped triangle) and equate them to the observed count/ total shapes, then we can easily check that there are 4 variables and 5 equations (because of the constraint that sum of probabilities should be 1) and hence the equations can never be exactly satisfied. On the other hand, if we add one more feature, then we have 5 variables and 5 equations and hence we can get the exact shape count as required. Therefore, the added feature in lesson 4 is important.

No, this is not the only additional feature that could have been added. As a matter of fact, any combination of 2 features would have been sufficient for this purpose.

1d.

If no regularization is present, then the feature parameters can take any value to model the training data which can lead to overfitting because of the inherent noise in the data. We can see from the lesson, if no regularization is there then the values of the feature parameters is quite high. Once we use the l1 regularization, it induces sparseness. It can be observed that one of the feature values (hollow) became zero when we solved the optimization problem using l1 regularization. On the other hand, if we use l2 regularization it simply results in small values of the feature parameters for all of them.

1e.

As the number of tokens or training data (N) increases, the log likelihood will be more affected by it than the regularizer term.

$F(\theta) = (\sum_{i=1}^{N} \log p(y_i|x_i))\text{-}C.R(\theta)$

Hence the model designed will try to match the empirical data. As more and more training data is added keeping the regularizer term constant, the model will tend to overfit as the regularizer's effect will continue to decrease. On the other hand, if the regularizer term is increased continuously, then it will dominate and most feature parameters will tend to 0. Hence if C is very large compared to the training data all feature parameters will be zero which is the limit and in that case the model will result in equal data for all shapes irrespective of the training data i.e. the training data will not matter anymore.

1f.

In Lesson 10 there are 9 parameters compared to the 6 parameters in lesson 9 and hence lesson 10 involves a more complicated model. For N=5 there is not much difference between the two models but for N=5000 and no regularization or with C=1, it turns out that lesson 10 can almost exactly model the training data whereas the model in lesson 9 fails to model the observed data for 2 of the shapes. Even though lesson 10 is modelling the training data exactly, there exists some smoothness because of the regularizer involved and as C is increased we can find a gradual decrease in the exact modelling of the training data. This also concludes the fact that as the complexity of the model is increased the training error can go to zero but we might over fit.

1g.

Adding features for contexts that has no relation with the training data outcomes does not change the model at all. The log likelihood or the objective function is completely insensitive to the feature weights of those contexts. This can be observed in lesson 14. Now, if we indeed want to use the context for our prediction, we need to use features that resemble bigram features and conjoin both the context and the outcome. Now the objective function is sensitive to these feature weights. Hence in a real-world scenario, we should use contexts that are related to the outcome or conjoin them to affect the model. Hence the context should be properly selected so that it does not become practically useless.

1h.

I am answering this question pertaining to lesson 16.

The model can fit the data moderately well. As it turns out among the bigram features same event and same shape has the highest weight and same fill has the smallest weight. Again, as the training data increases, the model can better fit the data because of lower variance and as the regularization coefficient is increased, the features start tending to zero. Hence, if we use l1 regularization, with C=10, it shows us that the bigram features are quite important compared to some other features like triangle & solid or square & striped which go to zero. With l2 regularization, similar thing happens except more number of features go close to zero other than going exactly to zero. As we increase C, the importance of the training data gradually decreases and the weight of all features gradually tend to zero.

2.

2a) Considering there are N training samples and taking the log-likelihood over all training samples,

$$L = \prod_{i=1}^{N} p(z_i)$$

where, according to maxent model,

$$p(z_i) = \frac{e^{\theta_k^T f(z_i)}}{\sum_{j=1}^{V} e^{\theta_k^T f(j)}} = \frac{e^{\theta^T f(z)}}{\tilde{Z}}$$

Now, $\log L = \sum_{i=1}^{N} \log p(z_i)$

$$= \sum_{i=1}^{N} \left(\theta_k^T f(z_i) - \log \tilde{Z} \right)$$

$$\frac{\partial \log L}{\partial \theta_K} = \sum_{i=1}^{N} f_K(z_i) - n \frac{\partial}{\partial \theta_K} \log \tilde{Z} = 0$$

$$\therefore \frac{\partial}{\partial \theta_k} \log \tilde{Z} = \frac{1}{n} \sum_{i=1}^{n} f_k(z_i)$$

$$\frac{\partial}{\partial \theta_k} \log \tilde{Z} = \frac{1}{\tilde{Z}} \cdot \frac{\partial}{\partial \theta_k} \sum_{j=1}^{V} e^{\theta^T f(j)}$$

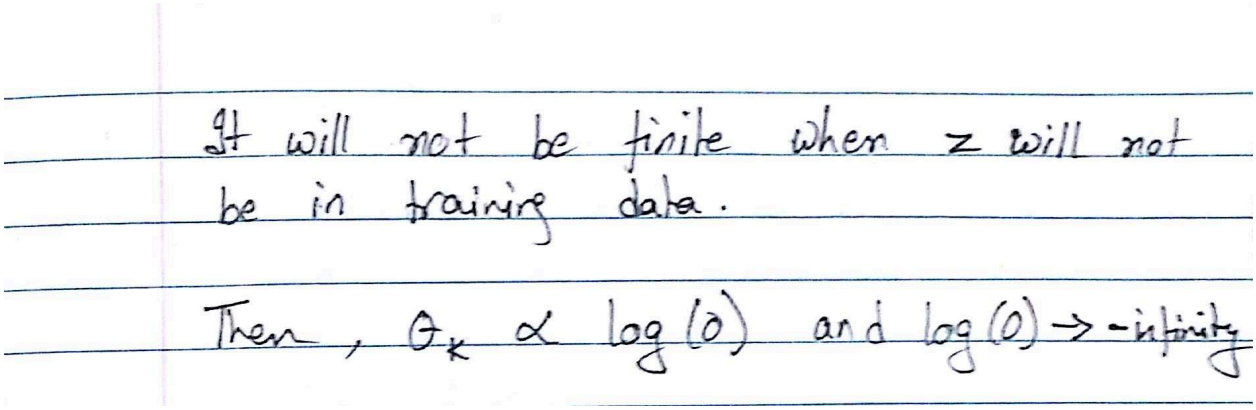
$$= \sum_{j=1}^{V} \frac{1}{\tilde{Z}} e^{\theta^T f(j)} \cdot f_k(j)$$

$$\therefore \sum_{j=1}^{V} \frac{1}{\tilde{Z}} e^{\theta^T f(j)} f_k(j) = \frac{1}{n} \sum_{i=1}^{n} f_k(z_i)$$

$$\Rightarrow \frac{e^{\theta_k}}{\tilde{Z}} = \frac{1}{n} \sum_{i=1}^{1} \mathbb{1}(z_i = k)$$

$\theta_k \propto \log$ (count of $z=k$ in the training data)

θ^* is not guaranteed to be finite.



3.

In this paper, Ronald Rosenfeld has described their approach in adaptive language modelling. They used a Maximum Entropy(ME) model, a selective unigram cache, a conditional bigram cache, and a conventional static trigram. They have used ARPA's official WSJ corpus for experiments and reported perplexity and word error rate results. To create the selective unigram cache, they stored all words that occurred in the history of the document. The motivation behind unigram cache is very interesting. Once a word occurs in a document, its probability of re-occurring is greatly improved. But this also depends on prior frequency of the word. The occurrence of a common word provides little new information whereas, the occurrence of a rare word provides a lot of information. Also, the occurrence of a more common word deviates less from the expectations of the static model, and hence requires very less modification to it. Similarly, bigram and trigram caches are created by storing consecutive word pairs and word triples respectively. The training data consisted of 38 million words of Wall Street Journal (WSJ) text from 1987-1989. To measure the impact of the amount of training data on language model adaptation, they experimented with systems based on varying amounts of training data. The adaptive language model was based on four component language models: (i)A conventional backoff trigram model (ii)A Maximum Entropy model (iii)A selective unigram cache (iv)A conditional bigram cache. All these four models were combined linearly with varying weights. They chose the specific weights by minimizing the perplexity of unseen data but later concluded that this did not always correspond with minimizing error rate. To evaluate error rate reduction, they used the Nov93 ARPA S1 evaluation set. They figured out that the main disadvantage of the Maximum Entropy framework is the computational requirements of training the ME model. But these are not severe for modest amounts of training data. The approach is thus particularly applicable in domains with a modest amount of LM training data. They have also demonstrated that the ME model significantly improves on the conventional static trigram.

I found the idea in this paper to be quite novel and interesting. Particularly, combining the four different component language models and determining the weights based on perplexity and

error reduction rate was an interesting approach. The paper is a significant contribution in the field of language modelling techniques.

4.

h=0.000001. The code is Q4.py.

a. The file produced is unigram_maxent.csv.

b. The file produced is bigram_maxent.csv.

5. The code is Q5.py

5. Model 1.

$$p(\text{language} \mid \text{word}) = \frac{\text{count}(\text{word}, \text{language pair}) + \lambda}{\text{count}(\text{word}) + \lambda \cdot V},$$

where V = total number of words in the vocabulary.

Here λ is the only parameter.

I have optimized ~~this~~ the log likelihood using scipy.optimize.minimize. I have used the method 'L-BFGS-B' and taken gtol = ~~1e-4~~ $1 \times e^{-4}$

To compute the gradient, I have used the package autograd.

After prediction and evaluation,

micro recall = 0.6922462

micro precision = 0.6922462

macro recall = 0.50069115

macro precision = 0.60644593

Model 2 :

$$P(\text{lang} \mid \text{word}) = P(\text{word} \mid \text{lang}) \times P(\text{lang})$$

Let the word be okay and lang be English

$$P(\text{okay} \mid \text{English}) = P(o \mid \langle bos \rangle, \langle English \rangle) \times P(k \mid o, \langle English \rangle) \times P(a \mid k, \langle English \rangle) \times P(y \mid a, \langle English \rangle)$$

$$P(k \mid o, \langle English \rangle) = \frac{e^{\theta^T f(\langle English \rangle, o, k)}}{\sum_{\beta \in \text{character vocabulary}} e^{\theta^T f(\langle English \rangle, o, \beta)}}$$

character vocabulary consists of all characters in the training data.

$f(\langle English \rangle, o, k)$ is a 32301 dimensional vector.

Number of languages in the training file X

Bi character bigram vocabulary size = 32301

~~$f_{lang, bigram}(\langle lang \rangle$~~

$$f_{lang, bigram}(\langle English \rangle, 0, k) = \frac{count(\langle English \rangle, 0, k)}{count(\langle English \rangle, 0, \text{Bi character vocabulary})}$$

~~$f_{lang, bigram} = 0$ when~~

θ is a 32301 dimensional vector which we need to optimize.

I have optimized the loglikelihood using scipy.optimize.minimize. I have used the method = 'L-BFGS-B' and gtol = $1 \times e^{-3}$.

To reduce the time taken for optimization, I have randomly selected 200 word-language pairs from the development data and run the optimization on them.

After prediction and evaluation,

micro recall = 0.40691927
micro precision = 0.40691927
macro recall = 0.23329453 6
macro ~~per~~ precision = 0.22851246186

Model 3:

$$p(\text{lang} \mid \text{word}) = \frac{e^{\theta^T f(\text{word}, \text{lang})}}{\sum_{\text{lang}} e^{\theta^T f(\text{word}, \text{lang})}}$$

$f(\text{word}, \text{lang})$ is a 64602 dimensional vector.

Number of languages in the training file x character bigram vocabulary size x 2
= 64602

There are 2 features for every language bigram pair

$f_{English, ok1}(word, lang)$

$$= \frac{count(ok, English)}{count(ok)}$$

if ok in word and lang - English

else $f_{English, ok1}(word, lang) = 0$

$f_{English, ok2} = 0$ if ok not in word and lang ≠ English

else

$$f_{English, ok2} = \hat{P}(k|o) = \frac{count(o,k)}{count(o)}$$

Here, θ is a 64602 dimensional vector which we need to optimize.

Here, too I have used the same method ~~to an~~ to optimize and randomly selected 200 word language pairs from dev data

After prediction and Evaluation;

micro recall = 0.570343725019
micro precision = 0.570343725019
macro recall = 0.301261060147 9
macro precision = 0.424778546 47