

# Design Project Report

# Machine Learning

---

Tang Xiaoyue 1002968, Zhu Bo 1002856, Li Zihao 1002966  
10th, December, 2019

# Table of Contents

## Table of Contents

### Approaches

Part II - Estimates the emission parameters

Data Processing:

Smoothing:

Then we perform naive analysis to produce the tags:

Part III - Implement the Viterbi algorithm

Estimates the transition parameters:

Then we calculate the probability:

Then we implement Viterbi Algorithm:

Part IV - Implement the 7-th best Viterbi algorithm

Top 7 scores stored:

Recursively cross multiplication:

Part V - Better design for developing systems for sequence labeling

### Results

– EN/

1. Dev.p2.out

2. Dev.p3.out

3. Dev.p4.out

4. Dev.p5.out

– AL/

1. Dev.p2.out

2. Dev.p3.out

3. Dev.p4.out

4. Dev.p5.out

– CN/

1. Dev.p2.out

2. Dev.p3.out

– SG/

1. Dev.p2.out

2. Dev.p3.out

### Reference

## Approaches

### Part II - Estimates the emission parameters

$$e(x|y) = \frac{\text{Count}(y \rightarrow x)}{\text{Count}(y)}$$

According to this equation, for this part we want to calculate the number of times that y occurs in the training set and the number of times that y tag lead to word x.

Data Processing:

```
def dataProcessing(filePath):
    tags = {}
    words = {}
    labelWords = {}
    estimates = {}
    for line in open(filePath, encoding='utf-8', mode='r'):
        segmentedLine = line.rstrip()
        if segmentedLine:
            segmentedLine = segmentedLine.rsplit(' ', 1) #x
            word = segmentedLine[0] #y
            tag = segmentedLine[1]
            if word not in words:
                words[word] = 1
            else:
                words[word] += 1
            if tag not in tags:
                tags[tag] = 1
                labelWords[tag] = {word: 1}
            else:
                tags[tag] += 1
                if word not in labelWords[tag]:
                    labelWords[tag][word] = 1
                else:
                    labelWords[tag][word] += 1

    return tags, words, labelWords, estimates
```

### Smoothing:

For the words not in training set, we set it to be #UNK# and we replace those words that appear less than k times in the training set with a special word token #UNK# before training

```
if word not in words or words[word] < k:
    labelWords[tag]['#UNK#'] += labelWords[tag][word]
```

Then we perform native analysis to produce the tags:

```
for line in open(inputPath, encoding='utf-8', mode='r'):
    word = line.rstrip()
    if word:
        prediction = ('', 0.0)
        for tag in estimates:
            if word in estimates[tag] and estimates[tag][word] >
prediction[1]:
                prediction = (tag, estimates[tag][word])
        if prediction[0]:
            f.write('%s %s\n' % (word, prediction[0]))
        else:
            f.write('%s %s\n' % (word, unkPrediction[0]))
    else:
        f.write('\n')
```

## Part III - Implement the Viterbi algorithm

$$q(y_i|y_{i-1}) = \frac{Count(y_{i-1}, y_i)}{Count(y)}$$

In this part, we want to get the transition probabilities by the formula above. Therefore, we need to count the times each of these 2 tag sequence occurred and the number of times the previous tag occurred in the training set.

Estimates the transition parameters:

```
for line in open(filePath, encoding='utf-8', mode='r'):
    previousState = currentState if (currentState != '##STOP##') else
'##START##' # y(i-1)
    segmentedLine = line.rstrip()

    if segmentedLine:
```

```

        segmentedLine = segmentedLine.rsplit(' ', 1)
        currentState = segmentedLine[1]
    else:
        if previousState == '##START##': break
        currentState = '##STOP##'
    if previousState not in tags:
        tags[previousState] = 1
        transitionTag[previousState] = {currentState: 1}
    else:
        tags[previousState] += 1
        if currentState not in transitionTag[previousState]:
            transitionTag[previousState][currentState] = 1
        else:
            transitionTag[previousState][currentState] += 1

```

Then we calculate the probability:

```

for transition in transitionTag[tag]:
    transitionProbability[tag][transition] =
float(transitionTag[tag][transition]) / tags[tag]

```

Then we implement Viterbi Algorithm:

### 1. Initialization

```

# Initialization stage
for label in tags:
    if label not in transitionEstimates['##START##']: continue
    if sequence[0] in m_training:
        if sequence[0] in emissionEstimates[label]:
            emission = emissionEstimates[label][sequence[0]]
        else:
            emission = 0.0
    else:
        emission = emissionEstimates[label]['#UNK#']

    pi[0][label] = [transitionEstimates['##START##'][label] * emission]

```

### 2. Recursion

```

for k in tqdm.tqdm(range(1, len(sequence))):
    for label in tags:

```

```

        for transTag in tags:
            if label not in transitionEstimates[transTag]: continue
            score = pi[k-1][transTag][0] *
transitionEstimates[transTag][label]
            if score > pi[k][label][0]:
                pi[k][label] = [score, transTag]
        if sequence[k] in m_training:
            if sequence[k] in emissionEstimates[label]:
                emission = emissionEstimates[label][sequence[k]]
            else:
                emission = 0.0
        else:
            emission = emissionEstimates[label]['#UNK#']
        pi[k][label][0] *= emission

```

### 3. Final

```

# Finally
result = [0.0, '']
for transTag in tags:
    if '##STOP##' not in transitionEstimates[transTag]: continue
    score = pi[-1][transTag][0] * transitionEstimates[transTag]['##STOP##']
    if score > result[0]:
        result = [score, transTag]

```

After getting all the scores, we feed the prediction back to write our prediction:

```

# Backtracking
if not result[1]: # for those weird cases where the final probability is 0
    return

prediction = [result[1]]
for k in reversed(range(len(sequence))):
    if k == 0: break # skip ##START## tag
    prediction.insert(0, pi[k][prediction[0]][1])

return prediction

```

## Part IV - Implement the 7-th best Viterbi algorithm

## Top 7 scores stored:

For 1-th best Viterbi, each node will have a  $\pi$  value that stores the best score, while for 7-th best Viterbi, we store 7 scores as well as a metadata 'parent\_order' inside each  $\pi$ .

To make it more clear,

- $\pi$  will look like this: `pi = [{layer0}, {layer2}, ..., {layerN}]`
- Each layer contains: `layer = {tag: [[score, parent_word, parent_order], ... [score, parent_word, parent_order]]}`
- Where parent\_order points to which of the 7 top scores on the parent node gives the 7-th score on the current layer

## Recursively cross multiplication:

What is also different from the 1-th Viterbi is that, when calculating the score, we do cross multiplication which is shown here

```
def recursive(k, prev_layer, word, tags, m_training, emissionEstimates,
transitionEstimates):
    layer = {tag: [] for tag in tags}
    # layer = {tag: [[score, word, parent_order], ... [score, word,
parent_order]]}
    for c_tag in tags:
        temp_scores = []
        for p_tag in tags:
            if c_tag not in transitionEstimates[p_tag]:
                continue # only compare p_tags which can transition to
c_tag

            if word in m_training: # if this word is not #UNK#
                # and this emission can be found
                if word in emissionEstimates[c_tag]:
                    emission = emissionEstimates[c_tag][word]
                else: # but this emission doesn't exist
                    emission = 0.0000000001
            else: # if this word is #UNK#
                emission = emissionEstimates[c_tag]['#UNK#']

            # n scores for each prev_node
            for order in range(0, len(prev_layer[p_tag])):
                # score = prev_layer*a*b
```

```

        temp_score = prev_layer[p_tag][order][0] * \
            transitionEstimates[p_tag][c_tag] * emission
        # 7*n scores with their parents
        temp_scores.append([temp_score, p_tag, order])
    # sort by temp_score
    temp_scores.sort(reverse=True)

    kbests = min(k, len(temp_scores))
    for kbest in range(kbests): # get top k best
        layer[c_tag].append(temp_scores[kbest])

    return layer

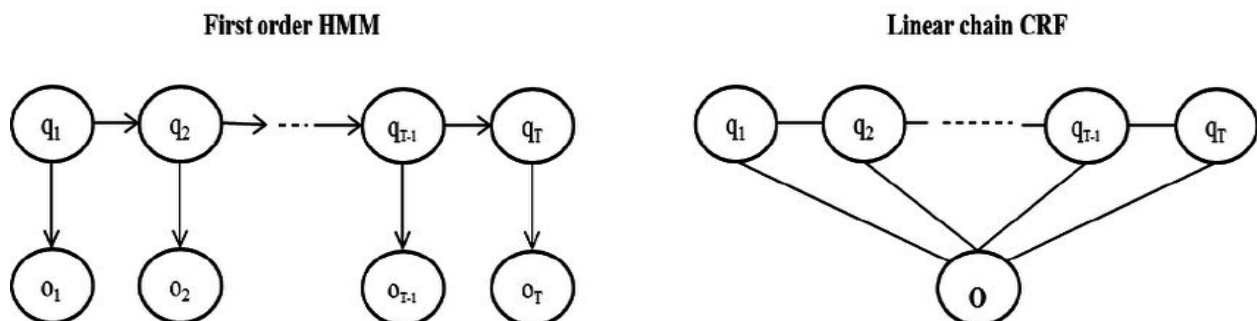
```

## Part V - Better design for developing systems for sequence labeling

In this part we decide to try CRF as our new approach.

### Step1: Understand CRF

Hidden Markov Models are generative, and give output by modeling the joint probability distribution. On the other hand, Conditional Random Fields are discriminative, and model the conditional probability distribution. CRFs don't rely on the independence assumption (that the labels are independent of each other), and avoid label bias. One way to look at it is that Hidden Markov Models are a very specific case of Conditional Random Fields, with constant transition probabilities used instead. HMMs are based on Naive Bayes, which we say can be derived from Logistic Regression, from which CRFs are derived.





## Step2: Try to build CRF

In this step, we refer to the Linear Chain CRF built by Seong-Jin-Kim

There are two main parts in the code: getting feature set (feature.py) and modeling (part5.py)

At first, function readTrainFile will get input training set, split words and tags and store them in a new list: data.

```
def readTrainFile(self, filename):
    data = list()
    segmentLine = list(open(filename,encoding='UTF-8',mode='r'))
    word = list()
    tag = list()
    for data_string in segmentLine:
        words = data_string.strip().split()
        if len(words) is 0:
            data.append((word, tag))
            word = list()
            tag = list()
        else:
            word.append(words[:-1])
            tag.append(words[-1])
    if len(word) > 0:
        data.append((word, tag))

    return data
```

After that , these data are passed to feature.py for scanning feature sets.

```
def scan(self, data):
    # data= (word,tag)
    # Constructs a feature set, and counts empirical counts.
    for word, tag in data:
        lastY = STARTING_LABEL_INDEX
        for t in range(len(word)):
            # Gets a label id
            try:
                y = self.tagSet[tag[t]]
            except KeyError:
                y = len(self.tagSet)
```

```

        self.tagSet[tag[t]] = y
        self.tagArray.append(tag[t])
    # Adds features
    self._add(lastY, y, word, t)
    lastY = y

```

Finally feature sets are generated and passed to part5.py for modeling.

```

def get_feature_list(self, word, t):
    feature_list_dic = dict()
    for feature_string in self.feature_func(word, t):
        for (lastY, y), feature_id in
self.feature_dic[feature_string].items():
            if (lastY, y) in feature_list_dic.keys():
                feature_list_dic[(lastY, y)].add(feature_id)
            else:
                feature_list_dic[(lastY, y)] = {feature_id}
    return [((lastY, y), feature_ids) for (lastY, y), feature_ids in
feature_list_dic.items()]

def serialize_feature_dic(self):
    serialized = dict()
    for feature_string in self.feature_dic.keys():
        serialized[feature_string] = dict()
        for (lastY, y), feature_id in
self.feature_dic[feature_string].items():
            serialized[feature_string]['%d_%d' % (lastY, y)] = feature_id
    return serialized

```

In part5.py, likelihood function is minimised by L-BFGS-B algorithm (from scipy).

```

def estimation(self):

    training_feature_data = self.getFeature()
    print('Start L-BFGS-B')
    # Minimize a function func using the L-BFGS-B algorithm.
    self.params, log_likelihood, information =
fmin_l_bfgs_b(func=_log_likelihood,
fprime=_gradient,x0=np.zeros(len(self.feature_set)),args=(self.training_data,
self.feature_set, training_feature_data,

```

```

        self.feature_set.get_empirical_counts(),
        self.tagSet, VARIANCE),
        callback=_callback)

print('Finish!')

```

After saving model as json file. The modeling stage is done.

For the training stage, model is loaded as json and test file is read by readTestFile.

```

def readTestFile(self, filename):
    data = list()
    segmentLine = list(open(filename,encoding='utf-8',mode='r'))
    word = list()
    for data_string in segmentLine:
        words = data_string.strip().split()
        if len(words) is 0:
            data.append((word))
            word = list()
        else:
            word.append(words)

    if len(word) > 0:
        data.append((word))

    return data

```

Data and model are passed viterbi algorithm to generate best sequences(decoding)

```

def viterbi(self, word, potential_table):
    time_length = len(word)
    max_table = np.zeros((time_length, self.labelCounts))
    argmax_table = np.zeros((time_length, self.labelCounts), dtype='int64')

    #Initialization
    t = 0
    for label_id in range(self.labelCounts):
        max_table[t, label_id] = potential_table[t][STARTING_LABEL_INDEX,
label_id]

    #Recursive

```

```

        for t in range(1, time_length):
            for label_id in range(1, self.labelCounts):
                max_value = -float('inf')
                max_label_id = None
                for prev_label_id in range(1, self.labelCounts):
                    value = max_table[t-1, prev_label_id] *
potential_table[t][prev_label_id, label_id]
                    if value > max_value:
                        max_value = value
                        max_label_id = prev_label_id
                max_table[t, label_id] = max_value
                argmax_table[t, label_id] = max_label_id

sequence = list()
next_label = max_table[time_length-1].argmax()
sequence.append(next_label)
#Final
for t in range(time_length-1, -1, -1):
    next_label = argmax_table[t, next_label]
    sequence.append(next_label)
#Decode
return [self.tagSet[label_id] for label_id in sequence[::-1][1:]]

```

However, it is different from previous viterbi algorithm as we use a probability table for the emission and transition probability.

```

def _generate_potential_table(params, labelCounts, feature_set, word,
inference=True):

    tables = list()
    for t in range(len(word)):
        table = np.zeros((labelCounts, labelCounts))
        if inference:
            for (lastY, y), score in feature_set.calc_inner_products(params,
word, t):
                if lastY == -1:
                    table[:, y] += score
                else:
                    table[lastY, y] += score

```

```

else:
    for (lastY, y), feature_ids in word[t]:
        score = sum(params[fid] for fid in feature_ids)
        if lastY == -1:
            table[:, y] += score
        else:
            table[lastY, y] += score
    table = np.exp(table)
    if t == 0:
        table[STARTING_LABEL_INDEX+1:] = 0
    else:
        table[:, STARTING_LABEL_INDEX] = 0
        table[STARTING_LABEL_INDEX, :] = 0
    tables.append(table)

return tables

```

After that, best sequence is generated and written to output file.

## Results

– EN/

### 1. Dev.p2.out

#Entity in gold data: 13179

#Entity in prediction: 19406

#Correct Entity : 9152

Entity precision: 0.4716

Entity recall: 0.6944

Entity F: 0.5617



#Correct Sentiment : 7644

Sentiment precision: 0.3939

Sentiment recall: 0.5800

Sentiment F: 0.4692

## 2. Dev.p3.out

#Entity in gold data: 13179

#Entity in prediction: 12738

#Correct Entity : 10833

Entity precision: 0.8504

Entity recall: 0.8220

Entity F: 0.8360

#Correct Sentiment : 10419

Sentiment precision: 0.8179

Sentiment recall: 0.7906

Sentiment F: 0.8040

## 3. Dev.p4.out

#Entity in gold data: 13179

#Entity in prediction: 13411

#Correct Entity : 10512



Entity precision: 0.7838

Entity recall: 0.7976

Entity F: 0.7907

#Correct Sentiment : 9958

Sentiment precision: 0.7425

Sentiment recall: 0.7556

Sentiment F: 0.7490

#### 4. Dev.p5.out

#Entity in gold data: 13179

#Entity in prediction: 12975

#Correct Entity : 11937

Entity precision: 0.9200

Entity recall: 0.9058

Entity F: 0.9128

#Correct Sentiment : 11679

Sentiment precision: 0.9001

Sentiment recall: 0.8862

Sentiment F: 0.8931



– **AL/**

### 1. Dev.p2.out

#Entity in gold data: 8408

#Entity in prediction: 19484

#Correct Entity : 2898

Entity precision: 0.1487

Entity recall: 0.3447

Entity F: 0.2078

#Correct Sentiment : 2457

Sentiment precision: 0.1261

Sentiment recall: 0.2922

Sentiment F: 0.1762

### 2. Dev.p3.out

#Entity in gold data: 8408

#Entity in prediction: 8482

#Correct Entity : 6722

Entity precision: 0.7925

Entity recall: 0.7995

Entity F: 0.7960





#Correct Sentiment : 6069

Sentiment precision: 0.7155

Sentiment recall: 0.7218

Sentiment F: 0.7187

### 3. Dev.p4.out

#Entity in gold data: 8408

#Entity in prediction: 8923

#Correct Entity : 6057

Entity precision: 0.6788

Entity recall: 0.7204

Entity F: 0.6990

#Correct Sentiment : 5074

Sentiment precision: 0.5686

Sentiment recall: 0.6035


Sentiment F: 0.5855

### 4. Dev.p5.out

#Entity in gold data: 8408

#Entity in prediction: 8462

#Correct Entity : 7613



Entity precision: 0.8997

Entity recall: 0.9054

Entity F: 0.9025

#Correct Sentiment : 7023

Sentiment precision: 0.8299

Sentiment recall: 0.8353

Sentiment F: 0.8326

– **CN/**

1. [Dev.p2.out](#)

#Entity in gold data: 1478

#Entity in prediction: 9373

#Correct Entity : 765

Entity precision: 0.0816

Entity recall: 0.5176

Entity F: 0.1410

#Correct Sentiment : 285

Sentiment precision: 0.0304

Sentiment recall: 0.1928

Sentiment F: 0.0525



## 2. Dev.p3.out

#Entity in gold data: 1478

#Entity in prediction: 712

#Correct Entity : 307

Entity precision: 0.4312

Entity recall: 0.2077

Entity F: 0.2804

#Correct Sentiment : 210

Sentiment precision: 0.2949

Sentiment recall: 0.1421

Sentiment F: 0.1918

## – SG/

## 1. Dev.p2.out

#Entity in gold data: 4537

#Entity in prediction: 18451

#Correct Entity : 2632

Entity precision: 0.1426

Entity recall: 0.5801

Entity F: 0.2290



#Correct Sentiment : 1239

Sentiment precision: 0.0672

Sentiment recall: 0.2731

Sentiment F: 0.1078

## 2. Dev.p3.out

#Entity in gold data: 4537

#Entity in prediction: 3003

#Correct Entity : 1661

Entity precision: 0.5531

Entity recall: 0.3661

Entity F: 0.4406

#Correct Sentiment : 1035

Sentiment precision: 0.3447

Sentiment recall: 0.2281

Sentiment F: 0.2745



## Reference

- <https://medium.com/ml2vec/overview-of-conditional-random-fields-68a2a20fa541>
- <https://www.zhihu.com/question/20279019/answer/273594958>
- <https://github.com/kenjewu/CRF>
- <https://github.com/lancifolia/crf>
- 1989, Lawrence R. Rabiner, A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition