A Tutorial on ParsingReranker

11-711: Algorithms for NLP

Fall 2017

Outline

- Machine Learning Overview
- Implementation Details
  - Feature Extractor
  - Training
- Suggested Features

Machine Learning Overview

- **Input:** K-best List $\mathcal{L} = \{T_1, T_2, \ldots, T_K\}$
  - $\mathcal{L}$ comes from a parser that parses a sentence S
  - $\mathcal{T}(S)$ is the correct tree of S
- **Output:** Best tree $T^* = \arg\max_{T \in \mathcal{L}} F_1(\mathcal{T}(S), T)$
  - In general $\mathcal{T}(S) \notin \mathcal{L}$
- **Approach:** Learn a scoring function $s(T_i)$
  - *Hopefully* $s(T) > s(T')$ if $F_1(T) > F_1(T')$
  - Return $\arg\max_{T \in \mathcal{L}} s(T)$

How to Learn the Scoring Function?

- **Feature Map:** $\phi : T \in \text{Tree} \rightarrow \phi(T) \in \mathbb{R}^d$
- **Linear Scoring Function:**
  - $s(T) = w^\top \phi(T)$, where $w \in \mathbb{R}^d$ are learnable parameters
- **Learn w:** minimize a training loss $\ell(w)$
  - MaxEnt: L-BFGS
  - Perceptron
  - Margin:
    - Primal SVM: sub-gradient descent
    - Dual SVM: coordinate ascent

Outline

- Machine Learning Overview
- Implementation Details
  - Feature Extractor
  - Training
- Suggested Features

Where to Write Your Code?

- Your code are in `/edu/berkeley/nlp/assignments/rerank/student`

```
// AwesomeParsingRerankerFactory.java
public class AwesomeParsingRerankerFactory implements ParsingRerankerFactory {
  public ParsingReranker trainParserReranker(
      Iterable<Pair<KbestList, Tree<String>>> kbestListsAndGoldTrees) {
    return new MyAwesomeParsingReranker(kbestListsAndGoldTrees);
  }
}

// BasicParsingRerankerFactory.java
public class BasicParsingRerankerFactory implements ParsingRerankerFactory {
  public ParsingReranker trainParserReranker(
      Iterable<Pair<KbestList, Tree<String>>> kbestListsAndGoldTrees) {
    return new MyBasicParsingReranker();
  }
}
```

- You need to implement **at least two** training algorithms.

Outline

- Machine Learning Overview
- Implementation Details
  - Feature Extractor
  - Training
- Suggested Features

Feature Extractor

- An excerpt from

`edu/berkeley/nlp/assignments/rerank/SimpleFeatureExtractor.java`

```
public class SimpleFeatureExtractor {
  public int[] extractFeatures(KbestList kbestList, int idx,
      Indexer<String> featureIndexer, boolean addFeaturesToIndexer) {
    // TREE PROCESSING
    Tree<String> tree = kbestList.getKbestTrees().get(idx);

    // FEATURE COMPUTATION
    List<Integer> feats = new ArrayList<Integer>();
    // Feature 1: position in the k-best list
    addFeature("Posn=" + idx, feats, featureIndexer, addFeaturesToIndexer);
```

```
    // More features. Yours will go here

    int[] featsArr = new int[feats.size()];
    for (int i = 0; i < feats.size(); i++) {
      featsArr[i] = feats.get(i).intValue();
    }
    return featsArr;
  }

  private void addFeature(String feat, List<Integer> feats,
      Indexer<String> featureIndexer, boolean addNew) {
    if (addNew || featureIndexer.contains(feat)) {
      feats.add(featureIndexer.addAndGetIndex(feat));
    }
  }
}
```

Indexer

- An excerpt from `edu/berkeley/nlp/util/Indexer.java`

```
public class Indexer<E> /* ... */ {
  List<E> objects;
  Map<E, Integer> indexes;
}
```

- A list that supports efficient index lookups

Tree Processing Helpers: Tree

- An excerpt from `edu/berkeley/nlp/ling/Tree.java`

```
public class Tree<L> implements Serializable {
  L label;
  List<Tree<L>> children;

  public List<Constituent<L>> toConstituentList() {
    List<Constituent<L>> constituentList = new ArrayList<Constituent<L>>();
    toConstituentCollectionHelper(this, 0, constituentList);
    return constituentList;
```

```
  }
}
```

- Returns a list of `Consituent<L>`

Tree Processing Helpers: AnchoredTree

- An excerpt from `edu/berkeley/nlp/ling/AnchoredTree.java`

```
public class AnchoredTree<L> implements Serializable {
  final L label;
  final int startIdx;
  final int endIdx;
  final List<AnchoredTree<L>> children;

  public static <L> AnchoredTree<L> fromTree(Tree<L> tree) {
    // implementation
  }

  public List<AnchoredTree<L>> toSubTreeList() {
    return getPreOrderTraversal();
  }
}
```

- Labels each node in `Tree`with its span in S
- Can also turn `AnchoredTree` into `List<L>`

Tree Processing Helpers: SurfaceHeadFinder

- From `edu/berkeley/nlp/assignments/rerank/SurfaceHeadFinder.java`

```
public class SurfaceHeadFinder {
  private Map<String,Boolean> searchDirections;
  private Map<String,Set<String>> validHeads;
```

```
  public SurfaceHeadFinder() {
    this.searchDirections = new HashMap<String,Boolean>();
    searchDirections.put("ADJP", true); // More rules: searchDirections.put(...)

    this.validHeads = new HashMap<String,Set<String>>();
    validHeads.put("ADJP", new HashSet<String>(Arrays.asList(
        new String[] { "NNS", "NN", "$", "JJ", ... }))); // More rules: validHeads.put(...)
  }

  public int findHead(String label, List<String> preterminals) {
    if (label.equals("NP")) {
      return searchFindLastBefore(preterminals, true, npHeads, npBlockers);
    } else if (label.equals("PRN")) {
      return (preterminals.size() > 1 ? 1 : 0);
    } else if (validHeads.containsKey(label)) {
      return searchFindFirst(preterminals, true, validHeads.get(label));
    } else {
      return 0;
    }
  }
}
```

- **Input:** List of pre-terminals
- **Ouptut:** Syntactic head of the list

Outline

- Machine Learning Overview
- Implementation Details
  - Feature Extractor
  - Training
- Suggested Features

Training: MaxEnt and L-BFGS minimizer

- **MaxEnt** objective:

$$\ell(w; \{T_1, T_2, \ldots, T_K\}, \mathcal{T}) = \log \frac{\exp\{w^\top \phi(T^*)\}}{\sum_{i=1}^K \exp\{w^\top \phi(T_i)\}} - \frac{\lambda}{2}\|w\|^2$$

$$T^* = \text{argmax}_{T \in \{T_1, T_2, \ldots, T_K\}} F_1(T, \mathcal{T})$$

- L-BFGS interfaces in `edu/berkeley/nlp/math`

```
// Function.java
public interface Function {
  int dimension();
  double valueAt(double[] x);
}

// DifferentiableFunction.java
public interface DifferentiableFunction extends Function {
 double[] derivativeAt(double[] x);
}

// LBFGSMinimizer.java
public class LBFGSMinimizer implements GradientMinimizer, Serializable {
  public double[] minimize(DifferentiableFunction function, double[] initial, double tolerance) {
    // dark magic...
  }
}
```

Review: SVM in 5 minutes

- Have N training examples.
- Primal:

$$\min_w \frac{\lambda\|w\|^2}{2} + C\sum_{i=1}^N \left(\max_{T \in \mathcal{L}_i}\{w^\top \phi(T) + \ell(T)\} - w^\top \phi(T_i^*)\right)$$

- Dual:

$$\text{minimize:} \quad J(w, \xi) = \frac{\lambda\|w\|^2}{2} + C\sum_{i=1}^N \xi_i$$

$$\text{satisfying:} \quad w^\top \phi(T_i^*) \geq w^\top \phi(T) + \ell_i(T) - \xi_i, \forall i = 1, \ldots, N$$

- Why? **Largrange mutipliers** and **Karush-Kuhn-Tucker conditions.**

Training: Primal SVM with sub-gradient descent

- Location: `edu/berkeley/nlp/assignments/rerank`

```
// edu/berkeley/nlp/assignments/rerank/PrimalSubgradientSVMLearner.java
public class PrimalSubgradientSVMLearner<D> {
  public PrimalSubgradientSVMLearner(double stepSize, double regConstant, int numFeatures) { ... }

  public IntCounter train(IntCounter initWeights,
      final LossAugmentedLinearModel<D> model, List<D> data, int iters) {
    // More dark magic
    // But you have to read and understand to implement your updates
    // To this soon: LossAugmentedLinearModel<T>

    OnlineMinimizer minimizer = new AdagradMinimizer(stepSize, regConstant, iters);
    return minimizer.minimize(objs, initWeights.toArray(numFeatures), true, null);
  }
}

// edu.berkeley.nlp.assignments.rerank/AdagradMinimizer.java
public class AdagradMinimizer implements OnlineMinimizer {
  public IntCounter minimize(List<DifferentiableFunction> functions, double[] initial,
      boolean verbose, Callback iterCallbackFunction) {
    // dark magic...
    double[] result = new double[initial.length];
    for (int i = 0; i < result.length; ++i) {
      result[i] = lazyResult.getCount(i);
    }
    return IntCounter.wrapArray(result, result.length);
  }
}
```

Training: Primal SVM with sub-gradient descent

- `IntCounter`

```
// edu/berkeley/nlp/util/IntCounter.java
// Class for vector computation
public class IntCounter {
  public static IntCounter wrapArray(double[] arrayToWrap, int size) {
    return new IntCounter(arrayToWrap, size);
  }

  private IntCounter(double[] arrayToWrap, int size) {
    this.values = arrayToWrap;
    this.keys = null;
    this.size = size;
  }
}
```

Training: Primal SVM with sub-gradient descent

- `LossAugmentedLinearModel`

```
// edu/berkeley/nlp/assignments/rerank/LossAugmentedLinearModel.java
public interface LossAugmentedLinearModel<T> {
  public class UpdateBundle {
    public UpdateBundle(IntCounter goldFeatures,
        IntCounter lossAugGuessFeatures, double lossOfGuess) {
      this.goldFeatures = goldFeatures;
      this.lossAugGuessFeatures = lossAugGuessFeatures;
      this.lossOfGuess = lossOfGuess;
    }

    public final IntCounter goldFeatures;
    public final IntCounter lossAugGuessFeatures;
    public final double lossOfGuess;
  }

  public UpdateBundle getLossAugmentedUpdateBundle(T datum, IntCounter weights);
}
```

- In `getLossAugmentedUpdateBundle`, you process `datum` and return an `UpdateBundle` object

- What should `T datum` be?
  - Something with `KbestList`.
  - Read `PrimalSubgradientSVMLearner.java` to find out.

Training: Dual SVM with Coordinate Ascent and SMO

- Faster SVM training
- Coordinate ascent:
  - Fix all but one coordinate. Update it

$$w_i = \text{argmax}_{\hat{w}_i} J(w_1, w_2, \ldots, w_{i-1}, \hat{w}_i, w_{i+1}, \ldots, w_d)$$

  - If you cannot, then fix all but *a few* coordinates

$$w_I = \text{argmax}_{\hat{w}_I} J(w_{-I}, \hat{w}_I)$$

- Sequential Maximal Optimization (SMO):
  - Do coordinate ascent very aggressively

- Good luck...

Outline

- Machine Learning Overview
- Implementation Details
  - Feature Extractor
  - Training
- Suggested Features

Charniak and Johnson, ACL 2005

- *Coarse-to-fine n-best parsing and MaxEnt discriminative reranking*
- A comprehensive list of features:
  - **CoPar:** conjunct parallelism
  - **CoLenPar**: difference in #preterminals dominated in adj. conjuncts
  - **RightBranch:** prefer right-branching trees
  - **Heavy:** Tree node-base features
  - **Neighbours:** node and non-terminals to its left, right
  - **Rules:** local trees annotated with various information
  - **Ngrams:** tuples of adj. children nodes of the same parent
  - **Heads:** tuples of head-to-head dependencies
  - **LexFunHeads:** pairs of POS-tags of nodes' lex. head & func. head

- **WProj:** preterminals and their closest max. proj. ancestors
- **Word:** lex. items and their POS-tags
- **HeadTree:** local trees of the projections of preterminal node and these projections' siblings
- **NgramTree:** subtrees rooted in the LCA of contiguous preterminal nodes

Johnson and Ural, NAACL 2010

- *Reranking the Berkeley and Brown Parsers*

- Try more features. Found the followings to be important:
  - **HeadTree:** *mentioned*
  - **Heads:** *mentioned*
  - **SynSemHeads:** same as *LexFunHeads*
  - **RBContext:** how much each subtree deviates from right-branching
  - **InterpLogCondP:** log conditional probability according to parser
  - **NNgram:** parent and n-gram sequences of children categories

Hall et al., ACL 2014

- *Less Grammar, More Features*

- Span Features:
  - **SpanBasic:** identity of the first and last words of a span
  - **SpanContext:** words immediately preceding and following a span. Similar to *Neighbors*
  - **SpanSplitPoint:** where does the split in CKY happen?
  - **SpanShape:** whether words begin with upper/lowercase, digit, or punct. Focus on words at spans boundaries

General Advice

- Do ablation study
- Watch out for correlations
  - It hurts if your features correlate too much
- DropOut training

Questions?