Why people care about spelling

- Spelling errors can cause misunderstandings
- Standard spelling makes it easy to organize words & text:
  - e.g., Without standard spelling, how would you look up things in a lexicon or thesaurus?
  - e.g., Optical character recognition software (OCR) can use knowledge about standard spelling to recognize scanned words even for hardly legible input.
- Standard spelling makes it possible to provide a single text, accessible to a wide range of readers (different backgrounds, speaking different dialects, etc.).
- Using standard spelling can make a good impression in social interaction.

How are spell checkers used?

- **Interactive spelling checkers** = spell checker detects errors as you type.
  - It may or may not make suggestions for correction.
  - It needs a “real-time” response (i.e., must be fast)
  - It is up to the human to decide if the spell checker is right or wrong, and so we may not require 100% accuracy (especially with a list of choices)
- **Automatic spelling correctors** = spell checker runs on a whole document, finds errors, and corrects them
  - A much more difficult task.
  - A human may or may not proofread the results later.

**Detection vs. Correction**

- There are two distinct tasks:
  - **Error detection** = simply find the misspelled words
  - **Error correction** = correct the misspelled words
    - e.g., It might be easy to tell that *ater* is a misspelled word, but what is the correct word? *water?* *later?* *after?*
    - Note that detection is a prerequisite for correction.

Error causes

Keyboard mistypings

- **Space bar issues**
  - **Run-on errors** = two separate words become one
    - e.g., *the fuzz becomes the fuzz*
  - **Split errors** = one word becomes two separate items
    - e.g., *equalization becomes equalization*
  - Note that the resulting items might still be words:
    - e.g., *a tollway becomes a toll way*
  - **Word insertion**
    - e.g., *a tollway becomes a tollway*
  - **Word deletion**
    - e.g., *atoll way becomes a tollway*
  - **Word substitution**
    - e.g., *fuzz becomes the fuzz*
  - **Run-on errors**
    - e.g., *equalization becomes equalization*

Keyboard proximity

- e.g., *Jack becomes Hack since h and j are next to each other on a typical American keyboard*

Physical similarity

- similarity of shape, e.g., mistaking two physically similar letters when typing up something handwritten
  - e.g., *light for fight*
**Error causes**

**Phonetic errors**

- errors based on the sounds of a language (not necessarily on the letters)

  - *homophones* = two words which sound the same
    - e.g., red/read (past tense), cite/site/sight, they/their/there
  - letter/word substitution: replacing a letter (or sequence of letters) with a similar-sounding one
    - e.g., John *kracked* his nuckles. instead of John *cracked* his knuckles.

**Knowledge problems**

- not knowing a word and guessing its spelling (can be phonetic)
  - e.g., *sientist*
- not knowing a rule and guessing it
  - e.g., Do we double a consonant for ed words?
    - label → labeled or labelled?
    - hopped vs. hoped
- knowing something is odd about the spelling, but guessing the wrong thing
  - e.g., typing *sicsors* for the non-regular scissors

**Challenges & Techniques for spelling correction**

Before we turn to how we detect spelling errors, we’ll look briefly at three issues:

- **Tokenization**: What is a word?
- **Inflection**: How are some words related?
- **Productivity of language**: How many words are there?

How we handle these issues determines how we build a dictionary.

And then we’ll turn to the techniques used:

- Non-word error detection
- Isolated-word error correction
- Context-dependent word error detection and correction → grammar correction

**Inflection**

- A word in English may appear in various guises due to word *inflections* = word endings which are fairly systematic for a given part of speech
  - plural noun ending: *the boys* → *the boys*
  - past tense verb ending: *walk* → *walked*
- This can make spell-checking hard:
  - There are exceptions to the rules: “*mans*,” “*runned*”
  - There are words which look like they have a given ending, but they don’t: Hans, deed

**Productivity**

- part of speech change: nouns can be verbified
  - *email* is a common new verb coined after the noun *email*
- morphological productivity: prefixes and suffixes can be added
  - e.g., I can speak of un-*email*-able for someone who you can’t reach by email.
- words entering and exiting the lexicon, e.g.:
  - thou, or *spleet’split* (Hamlet III.3.10) are on their way out
  - New words all the time: omnishambles, phablet, supersize, ...
Non-word error detection

And now the techniques ...

- **non-word error detection** is essentially the same thing as word recognition = splitting up "words" into true words and non-words.
- How is non-word error detection done?
  - using a dictionary (construction and lookup)
  - n-gram analysis

Dictionary construction

- Do we include inflected words? i.e., words with prefixes and suffixes already attached.
  - Lookup can be faster
  - But takes more space & doesn't account for new formations (e.g., google → googled)
- Want the dictionary to have only the word relevant for the user → **domain-specificity**
  - e.g., For most people memoeize is a misspelled word, but in computer science this is a technical term
- Foreign words, hyphenations, derived words, proper nouns, and new words will always be problems
  - We cannot predict these words until humans have made them words.
  - Dictionary should be dialectally consistent.
    - e.g., include only color or colour but not both

Dictionaries

Intuition:

- Have a complete list of words and check the input words against this list.
- If it’s not in the dictionary, it’s not a word.

Two aspects:

- **Dictionary construction** = build the dictionary (what do you put in it?)
- **Dictionary lookup** = lookup a potential word in the dictionary (how do you do this quickly?)

N-gram analysis

- An *n*-gram here is a string of *n* letters.
  - a 1-gram (unigram)
  - at 2-gram (bigram)
  - ate 3-gram (trigram)
  - late 4-gram
- We can use this *n*-gram information to define what the possible strings in a language are.
  - e.g., *po* is a possible English string, whereas *kvt* is not.

This is more useful to correct optical character recognition (OCR) output, but we’ll still take a look.

Bigram array

- We can define a **bigram array** = information stored in a tabular fashion.
- An example, for the letters *k*, *l*, *m*, with examples in parentheses

<table>
<thead>
<tr>
<th></th>
<th>k</th>
<th>l</th>
<th>m</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>0</td>
<td>1 (tackle)</td>
<td>1 (Hackman)</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>1 (elk)</td>
<td>1 (hello)</td>
<td>1 (alms)</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>0</td>
<td>1 (hamlet)</td>
<td>1 (hammer)</td>
<td></td>
</tr>
</tbody>
</table>

- The first letter of the bigram is given by the vertical letters (i.e., down the side), the second by the horizontal ones (i.e., across the top).
- This is a **non-positional bigram array** = the array 1s and 0s apply for a string found anywhere within a word (beginning, 4th character, ending, etc.).

Positional bigram array

- To store information specific to the beginning, the end, or some other position in a word, we can use a **positional bigram array** = the array only applies for a given position in a word.
- Here’s the same array as before, but now only applied to word endings:

<table>
<thead>
<tr>
<th></th>
<th>k</th>
<th>l</th>
<th>m</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>1 (elk)</td>
<td>1 (hall)</td>
<td>1 (elm)</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Isolated-word error correction

- Having discussed how errors can be detected, we want to know how to correct these misspelled words:
  - The most common method is isolated-word error correction = correcting words without taking context into account.
  - Note: This technique can only handle errors that result in non-words.
  - Knowledge about what is a typical error helps in finding correct word.

Isolated-word error correction methods

- Many different methods are used; we will briefly look at four methods:
  - rule-based methods
  - similarity key techniques
  - probabilistic methods
  - minimum edit distance
- The methods play a role in one of the three basic steps:
  1. Detection of an error (discussed above)
  2. Generation of candidate corrections
     - rule-based methods
     - similarity key techniques
  3. Ranking of candidate corrections
     - probabilistic methods
     - minimum edit distance

Knowledge about typical errors

- Word length effects: most misspellings are within two characters in length of original
  - When searching for the correct spelling, we do not usually need to look at words with greater length differences.
- First-position error effects: the first letter of a word is rarely erroneous
  - When searching for the correct spelling, the process is sped up by being able to look only at words with the same first letter.

Rule-based methods

One can generate correct spellings by writing rules:
- Common missspelling rewritten as correct word:
  - e.g., hte → the
- Rules
  - based on inflections:
    - e.g., VCing → VCCing, where
      - V = letter representing vowel, 
        - basically the regular expression [aeiou]
      - C = letter representing consonant, 
        - basically [bcdfghjklmnpqrstvwxyz]
    - based on other common spelling errors (such as keyboard effects or common transpositions):
      - e.g., CsC → CaC
      - e.g., cie → CaC

Similarity key techniques (SOUNDEX)

- Problem: How can we find a list of possible corrections?
- Solution: Store words in different boxes in a way that puts the similar words together.
- Example:
  1. Start by storing words by their first letter (first letter effect),
     - e.g., punc starts with the code P.
  2. Then assign numbers to each letter
     - e.g., 0 for vowels, 1 for b, p, l, f (all bilabials), and so forth, e.g., punc → P052
  3. Then throw out all zeros and repeated letters, e.g., P052 → P52.
  4. Look for real words within the same box,
     - e.g., punk is also in the P52 box.

How is a mistyped word related to the intended?

For ranking errors, it helps to know:

Types of operations

- insertion = a letter is added to a word
- deletion = a letter is deleted from a word
- substitution = a letter is put in place of another one
- transposition = two adjacent letters are switched

Note that the first two alter the length of the word, whereas the second two maintain the same length.

http://en.wikipedia.org/wiki/Soundex
To calculate minimum edit distance, we set up a minimal number of operations it would take to convert one word into another. For example, we can take the following five steps to convert junk to haiku:

1. junk → juh (deletion)
2. juk → huk (transposition)
3. huk → hiku (insertion)
4. hiku → haiku (insertion)

But is this the minimal number of steps needed?

To be able to compute the edit distance of two words at all, we need to ensure there is a finite number of steps. This can be accomplished by requiring that letters cannot be changed back and forth a potentially infinite number of times, i.e., we limit the number of changes to the size of the material we are presented with, the two words.

Idea: Never deal with a character in either word more than once.

Result:

We could delete each character in the first word and then insert each character of the second word.

Thus, we will never have a distance greater than \( \text{length(word1)} + \text{length(word2)} \)

### Computing edit distances

#### Figuring out the upper bound

To be able to compute the edit distance of two words at all, we need to ensure there is a finite number of steps. This can be accomplished by

- requiring that letters cannot be changed back and forth a potentially infinite number of times, i.e., we limit the number of changes to the size of the material we are presented with, the two words.

Idea: Never deal with a character in either word more than once.

Result:

We could delete each character in the first word and then insert each character of the second word.

Thus, we will never have a distance greater than \( \text{length(word1)} + \text{length(word2)} \)

### Computing edit distances

#### Using a graph to map out the options

To calculate minimum edit distance, we set up a directed, acyclic graph, a set of nodes (circles) and arcs (arrows).

Horizontal arcs correspond to deletions, vertical arcs correspond to insertions, and diagonal arcs correspond to substitutions (a letter can be “substituted” for itself).

#### An example graph

Say, the user types in fyre.

We want to calculate how far away fry is (one of the possible corrections). In other words, we want to calculate the minimum edit distance (or minimum edit cost) from fyre to fry.

As the first step, we draw the following directed graph:

### Computing edit distances

Adding numbers to the example graph

The graph is acyclic = for any given node, it is impossible to return to that node by following the arcs.

We can add identifiers to the states, which allows us to define a topological order:

### Computing edit distances

Adding costs to the arcs of the example graph

We need to add the costs involved to the arcs.

In the simplest case, the cost of deletion, insertion, and substitution is 1 each (and substitution with the same character is free).

Instead of assuming the same cost for all operations, in reality one will use different costs, e.g., for the first character or based on the confusion probability.
Computing edit distances
How to compute the path with the least cost

We want to find the path from the start (A) to the end (T) with the least cost.

- The simple but dumb way of doing it:
  - Follow every path from start (A) to finish (T) and see how many changes we have to make.
  - But this is very inefficient! There are many different paths to check.

Computing edit distances
The smart way to compute the least cost

- The smart way to compute the least cost uses dynamic programming = a program designed to make use of results computed earlier
  - We follow the topological ordering.
  - As we go in order, we calculate the least cost for that node:
    - We add the cost of an arc to the cost of reaching the node this arc originates from.
    - We take the minimum of the costs calculated for all arcs pointing to a node and store it for that node.
  - The key point is that we are storing partial results along the way, instead of recalculating everything, every time we compute a new path.

Probabilistic methods

When converting from one word to another, a lot of words will be the same distance.

e.g., for the misspelling will, all of the following are one edit distance away:
- will
- wild
- wilt
- nil
Probabilities will help to tell them apart

The Noisy Channel Model

Probabilities can be modeled with the noisy channel model

Hypothesized Language: X
Noisy Channel: X \rightarrow Y
Actual Language: Y

Goal: Recover X from Y

- The noisy channel model has been very popular in speech recognition, among other fields
(Thanks to Mike White for the slides on the Noisy Channel Model)

Noisy Channel Spelling Correction

Correct Spelling: X
Typos, Mistakes: X \rightarrow Y
Misspelling: Y

Goal: Recover correct spelling X from misspelling Y
- Noisy word: Y = observation (incorrect spelling)
- We want to find the word (X) which maximizes: P(X|Y), i.e., the probability of X, given that Y has been seen

Example

Correct Spelling: donald

Transposition: id \rightarrow dl

Misspelling: donadi

Goal: Recover correct spelling donald from misspelling donadi (i.e., P(donald|donadi))
\[
p(x|y) \text{ is the probability of } x \text{ given } y
\]
- Let's say that yogurt appears 20 times in a text of 10,000 words.
  \[ p(\text{yogurt}) = \frac{20}{10,000} = 0.002 \]
- Now, let's say frozen appears 50 times in the text, and yogurt appears 10 times after it.
  \[ p(\text{yogurt}|\text{frozen}) = \frac{10}{50} = 0.20 \]

\[ \text{Bayes Rule: } P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)} \]

With \(X\) as the correct word and \(Y\) as the misspelling ...

**Finding the Correct Spelling**

Goal: for a given misspelling \(y\), find correct spelling \(x = \arg \max_x Pr(y|x)Pr(x)\)

1. List “all” possible candidate corrections, i.e., all words with one insertion, deletion, substitution, or transposition
2. Rank them by their probabilities

Example: calculate for donald

\[ Pr(\text{donald}|\text{donald})Pr(\text{donald}) \]

and see if this value is higher than for any other possible correction.

**Obtaining probabilities**

Confusion probabilities

- It is impossible to fully investigate all possible error causes and how they interact, but we can learn from watching how often people make errors and where.
- One way is to build a confusion matrix: a table indicating how often one letter is mistyped for another.

<table>
<thead>
<tr>
<th></th>
<th>r</th>
<th>s</th>
<th>t</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>n/a</td>
<td>12</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td></td>
<td>14</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>t</td>
<td></td>
<td>11</td>
<td>37</td>
<td>n/a</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(cf. Kernighan et al 1999)
Obtaining probabilities

Using a spelling error-annotated corpus:
- These matrices are calculated by counting how often, e.g., “ab” was typed instead of “a” in the case of insertion.

To get $P(Y|X)$, then, we find the probability of this kind of typo in this context. For insertion, for example ($X_p$ is the $p^{th}$ character of $X$):

\[
(2) \quad P(Y|X) = \frac{\text{ins}[X_{p-1}, Y_p]}{\text{count}[X_{p-1}]} \]

Spelling correction for web queries

A nice little side topic ...

Spelling correction for web queries is hard because it must handle:
- Proper names, new terms, etc. (blog, shrek, nsync)
- Frequent and severe spelling errors
- Very short contexts

Algorithm (2)

- Compute the set of all close alternatives for each word in the query
  - Look at word unigrams and bigrams from the logs; this handles concatenation and splitting of words
  - Use weighted edit distance to determine closeness
  - Search sequence of alternatives for best alternative string, using a noisy channel model

Constraint:
- No two adjacent in-vocabulary words can change simultaneously

The formal algorithm (just for fun)

Given a string $s_0$, find a sequence $s_1, s_2, \ldots, s_n$ such that:
- $s_0 = s_{n-1}$ (stopping criterion)
- $\forall i \in 0 \ldots n-1$, $\text{dist}(s_i, s_{i-1}) \leq \delta$ (only a minimal change)
- $P(s_{i-1}|s_i) = \max P(f(s_i))$ (the best change)

Some resources ...

Want to try some of these things for yourself?
- How to Write a Spelling Corrector by Peter Norvig: http://norvig.com/spell-correct.html
  - 21 lines of Python code (other programming languages also available)
- Birkbeck spelling error corpus: http://www.ota.ox.ac.uk/headers/0643.xml
Context-sensitive corrections are those that take into account the surrounding context.

- **power crd → power cord**
- **video crd → video card**
- **platnuin rings → platinum rings**

**Known Words**
- **golf war → gulf war**
- **sap opera → soap opera**

**Grammar correction—what does it correct?**

- **Syntactic errors** = errors in how words are put together in a sentence: the order or form of words is incorrect, i.e., ungrammatical.

  - **Local** syntactic errors: 1-2 words away
    - e.g., *The study was conducted mainly by John Black*. A verb is where a preposition should be.
    - **Long-distance** syntactic errors: (roughly) 3 or more words away
      - e.g., *The kids who are most upset by the little totem is going home early.*
      - Agreement error between subject kids and verb is

- **Non-word corrections**

  - e.g., *They are leaving in about fifteen minuets to go to her house.*
  - *minuets and minutes* are both plural nouns, but only one makes sense here.
Rule-based grammar correctors

We can write regular expressions to target specific error patterns. For example:

- To a certain extend, we have achieved our goal.
  - Match the pattern some or certain followed by extend, which can be done using the regular expression some|certain extend
  - Change the occurrence of extend in the pattern to extent.

See, e.g., http://www.languagetool.org/

Syntax

- Syntax = the study of the way that sentences are constructed from smaller units.
- There cannot be a “dictionary” for sentences since there is an infinite number of possible sentences:

  (3) The house is large.
  (4) John believes that the house is large.
  (5) Mary says that John believes that the house is large.

There are two basic principles of sentence organization:

- Linear order
- Hierarchical structure (Constituency)

Constituency

- What are the “meaningful units” of a sentence like Most of the ducks play extremely fun games?
  - Most of the ducks
  - of the ducks
  - extremely fun
  - extremely fun games
  - play extremely fun games
- We refer to these meaningful groupings as constituents of a sentence.

Beyond regular expressions

- But what about correcting the following:
  - A baseball teams were successful.
  - We should see that A is incorrect, but a simple regular expression doesn’t work because we don’t know where the word teams might show up.
  - A wildly overpaid, horrendous baseball teams were successful. (Five words later; change needed.)
  - A player on both my teams was successful. (Five words later; no change needed.)
- We need to look at how the sentence is constructed in order to build a better rule.

Linear order

- Linear order = the order of words in a sentence.
- A sentence can have different meanings, based on its linear order:

  (6) John loves Mary.
  (7) Mary loves John.

Languages vary as to what extent this is true, but linear order in general is used as a guiding principle for organizing words into meaningful sentences.

- Simple linear order as such is not sufficient to determine sentence organization though. For example, we can’t simply say “The verb is the second word in the sentence.”

  (8) I eat at really fancy restaurants.
  (9) Many executives eat at really fancy restaurants.

Hierarchical structure

- Constituents can appear within other constituents
- Constituents shown through brackets:
  [[Most of [the ducks]]] [play [[extremely fun] games]]
- Constituents displayed as a syntactic tree:

```
  a
  b
  c
  d
  e
  f
  g
```

- Most
cplay
ofthe
ducks
extremely
fun
games
extremely fun games

The joggers ran through the park.

the people that we interviewed

Susan

a huge, lovable bear

some children

students

my friends from Brazil

Some options:

you

most dogs

e.g., Nouns like mouse cannot.

verbs: eat, drink, sleep, ...

nouns: gas, food, lodging, ...

adjectives: quick, happy, brown, ...

adverbs: quickly, happily, well, westward

prepositions: on, in, at, to, into, of, ...

determiners/articles: a, an, the, this, these, some, much, ...

Categories

▶ We would also like some way to say that

the ducks, and extremely fun games

are the same type of grouping, or constituent, whereas

of the ducks

seems to be something else.

For this, we will talk about different categories

Lexical

Phrasal

Determining lexical categories

How do we determine which category a word belongs to?

▶ Distribution: Where can these kinds of words appear in a sentence?

▶ e.g., Nouns like mouse can appear after articles (“determiners”) like some, while a verb like eat cannot.

▶ Morphology: What kinds of word prefixes/suffixes can a word take?

▶ e.g., Verbs like walk can take an ing ending to mark them as past tense. A noun like mouse cannot.

(We’ll discuss this more with Language Tutoring Systems)

Lexical categories

Lexical categories are simply word classes, or what you may have heard as parts of speech. The main ones are:

verbs: eat, drink, sleep, ...

nouns: gas, food, lodging, ...

adjectives: quick, happy, brown, ...

adverbs: quickly, happily, well, westward

prepositions: on, in, at, to, into, of, ...

determiners/articles: a, an, the, this, these, some, much, ...

Phrasal categories

What about phrasal categories?

▶ What other phrases can we put in place of The joggers in a sentence such as the following?

The joggers ran through the park.

Some options:

Susan

students

you

most dogs

some children

a huge, lovable bear

my friends from Brazil

the people that we interviewed

Since all of these contain nouns, we consider these to be noun phrases, abbreviated with NP.

Building a tree

Other phrases work similarly (S = sentence, VP = verb phrase, PP = prepositional phrase, AdjP = adjective phrase):

Phrase Structure Rules

▶ We can give rules for building these phrases. That is, we want a way to say that a determiner and a noun make up a noun phrase, but a verb and an adverb do not.

▶ Phrase structure rules are a way to build larger constituents from smaller ones.

▶ e.g., S → NP VP

This says:

A sentence (S) constituent is composed of a noun phrase (NP) constituent and a verb phrase (VP) constituent. (hierarchy)

The NP must precede the VP. (linear order)
Some other possible English rules

- NP → Det N (the cat, a house, this computer)
- NP → Det AdjP N (the happy cat, a really happy house)

For phrase structure rules, as shorthand parentheses are used to express that a category is optional.

- We thus can compactly express the two rules above as one rule: NP → Det (AdjP) N
- Note that this is different and has nothing to do with the use of parentheses in regular expressions.

- AdjP → (Adv) Adj (really happy)
- VP → V (laugh, run, eat)
- VP → V NP (love John, hit the wall, eat cake)
- VP → V NP NP (give John the ball)
- PP → P NP (to the store, at John, in a New York minute)
- NP → NP PP (the cat on the stairs)

Phrase Structure Rules and Trees

With every phrase structure rule, you can draw a tree for it.

**Lexicon:**
- VI → saw
- Det → the
- N → dragon
- N → boy
- Adj → young

**Syntactic rules:**
- S → NP VP
- VP → VI NP
- NP → Det N
- N → Adj N

Context-free grammars

A context-free grammar (CFG) is essentially a collection of phrase structure rules.

- It specifies that each rule must have:
  - a left-hand side (LHS): a single non-terminal element = (phrasal and lexical) categories
  - a right-hand side (RHS): a mixture of non-terminal and terminal elements = actual words

- A CFG tries to capture a natural language completely.

Why "context-free"? Because these rules make no reference to any context surrounding them. i.e. you can’t say “PP → P NP” when there is a verb phrase (VP) to the left.

Trace of a top-down parse

Using these context-free rules, we can get a computer to parse a sentence = assign a structure to a sentence.

There are many, many parsing techniques out there.

- **Top-down:** build a tree by starting at the top (i.e. S → NP VP) and working down the tree.
- **Bottom-up:** build a tree by starting with the words at the bottom and working up to the top.
We need a verb somewhere.

The structure of this sentence is NP PP, but that doesn’t make up a whole sentence.

John at the pizza.

We need a verb somewhere.

A baseball teams were successful.

A followed by PLURAL NP: change A → The

John at the pizza.

The structure of this sentence is NP PP, but that doesn’t make up a whole sentence.

We need a verb somewhere.