Sure! Let's dive into **Class 19: NLP Projects** with a focus on **Word Spelling Correction**. This will include:

1. **Project Overview**
2. **Mathematical Foundation**
3. **Implementation (with Python code)**
4. **Best Practices: Evaluation, Debugging, Optimization**

---

## ✅ 1. Project: Word Spelling Correction

### 🎯 Goal:

Given a misspelled word, suggest the correct word from a known vocabulary.

---

## ✅ 2. Mathematical Foundation

### 2.1 Problem Setup

We want to find the correct word c for a possibly incorrect word w.
Using **Bayes' Theorem**:

$$\text{argmax}_{c \in \text{Candidates}} \; P(c|w) = \text{argmax}_{c \in \text{Candidates}} \; P(w|c) \cdot P(c)$$

Where:

- P(c) = Probability of the word c (Language Model)
- P(w|c) = Probability that c was mistakenly typed as w (Error Model)

---

### 2.2 Edit Distance (Levenshtein Distance)

Measures the **number of edits** (insertions, deletions, substitutions, transpositions) to change one word to another.

For example:

Edit distance("aple", "apple") = 1  (insertion of 'p')

---

### 2.3 Candidate Generation

Generate all possible words c from the dictionary that are within a certain edit distance (usually 1 or 2) of w.

---

## ✅ 3. Python Implementation

We will:

- Build a vocabulary from a corpus.
- Define edit distance operations.
- Rank candidates based on probability.

---

**Step 1: Build Vocabulary and Word Probabilities**

```
import re
from collections import Counter

def words(text):
    return re.findall(r'\w+', text.lower())

# Sample corpus (you can use a large one like Wikipedia or big.txt)
corpus = open('big.txt').read()  # You can download from Norvig's site
WORD_COUNTS = Counter(words(corpus))

# Total number of words
TOTAL_WORDS = sum(WORD_COUNTS.values())

def P(word):
    # Probability of word
    return WORD_COUNTS[word] / TOTAL_WORDS
```

---

**Step 2: Generate Candidates (Edit Distance 1)**

```
def edits1(word):
    letters = 'abcdefghijklmnopqrstuvwxyz'
    splits     = [(word[:i], word[i:]) for i in range(len(word) + 1)]
    deletes    = [L + R[1:]               for L, R in splits if R]
```

```
    transposes = [L + R[1] + R[0] + R[2:]   for L, R in splits if len(R)>1]
    replaces   = [L + c + R[1:]             for L, R in splits if R for c in letters]
    inserts    = [L + c + R                 for L, R in splits for c in letters]
    return set(deletes + transposes + replaces + inserts)
```

---

**Step 3: Filter Candidates Present in Dictionary**

```
def known(words):
    return set(w for w in words if w in WORD_COUNTS)


def candidates(word):
    return (
        known([word]) or
        known(edits1(word)) or
        known(e2 for e1 in edits1(word) for e2 in edits1(e1)) or
        [word]
    )
```

---

**Step 4: Suggest Correct Spelling**

```
def correct(word):
    return max(candidates(word), key=P)
```

---

✅ **Example Usage:**

```
print(correct("speling"))  # spelling
print(correct("korrectud"))  # corrected
print(correct("bycycle"))  # bicycle
```

---

✅ **4. Best Practices**

✅ **A. Model Evaluation**

**Metrics:**

- **Accuracy**: % of misspelled words corrected
- **Precision/Recall** (in broader spell check applications)

```
test_words = {"speling": "spelling", "korrectud": "corrected", "bycycle": "bicycle"}
corrected = [correct(w) for w in test_words]
accuracy = sum([corrected[i] == list(test_words.values())[i] for i in range(len(corrected))]) / len(corrected)
print("Accuracy:", accuracy)
```

---

✅ **B. Debugging**

- Add logs to inspect candidates and scores.
- Print intermediate probabilities and edit distances.

```
def debug_correct(word):
    print("Candidates and probabilities:")
    for c in candidates(word):
        print(f"{c}: {P(c):.8f}")
    return correct(word)

debug_correct("speling")
```

---

✅ **C. Optimization Tips**

1. **Use Tries**: For faster prefix search and candidate filtering.
2. **Restrict Search Space**: Limit edit distance to 1 or 2.
3. **Parallelize**: Use multiprocessing for large-scale correction.
4. **Use Language Models**: Use bigram/trigram probabilities (e.g., P(c | prev_word) for contextual spelling correction).
5. **Deep Learning Enhancement**:
    - Use LSTM/Transformer to predict word sequences.
    - BERT-like masked language models for in-context correction.

---

✅ **Optional: Transformer-based Contextual Correction**

```
from transformers import pipeline

# Uses BERT for masked language modeling
nlp_fill = pipeline("fill-mask")

def correct_sentence(sentence):
    # Replace incorrect word with [MASK] for prediction
    return nlp_fill(sentence)

print(correct_sentence("He went to the [MASK] to buy milk."))
```

---

✅ **Summary**

| Section | Key Takeaways |
|---|---|
| 🎯 Goal | Correct misspelled words using edit distance and probability |
| 🧠 Math | Bayes' Theorem: `argmax P(w |
| 🧮 Methods | Candidate generation using edit distance |
| 👨‍💻 Code | Python using Counter, edit distance, probability scoring |
| ✅ Best Practices | Evaluate with accuracy, optimize with Tries or language models |

---