Identifying Key Entities in Recipe Data – Report Document

1. Problem Statement

This project focuses on developing a **Named Entity Recognition (NER)** model using **Conditional Random Fields (CRF)** to extract meaningful entities from culinary recipe data. The primary goal is to automatically identify and classify text tokens into key categories such as **ingredients**, **quantities**, and **units**.

By converting unstructured recipe text into a structured format, this model can enable advanced applications in:

- Recipe management systems
- Dietary tracking tools
- E-commerce platforms (e.g., smart grocery lists)

The dataset comprises various culinary recipes, each with structured ingredient lists. These lists contain tokens labeled with their respective roles (e.g., "2" as quantity, "cups" as unit, "flour" as ingredient). This diversity supports the development of systems capable of understanding and analyzing culinary content effectively.

Identifying Key Entities in Recipe Data

Business Objective: The goal of this assignment is to train a Named Entity Recognition (NER) model using Conditional Random Fields (CRF) to extract key entities from recipe data. The model will classify words into predefined categories such as ingredients, quantities and units, enabling the creation of a structured database of recipes and ingredients that can be used to power advanced features in recipe management systems, dietary tracking apps, or e-commerce platforms.

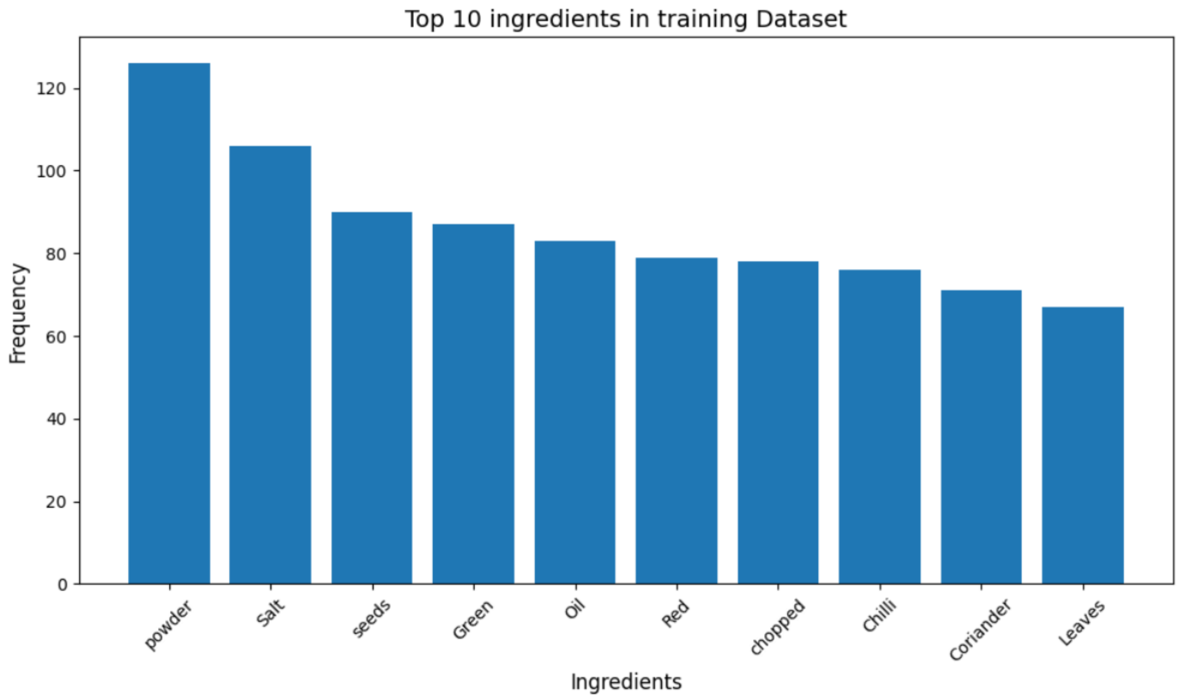
Data Description

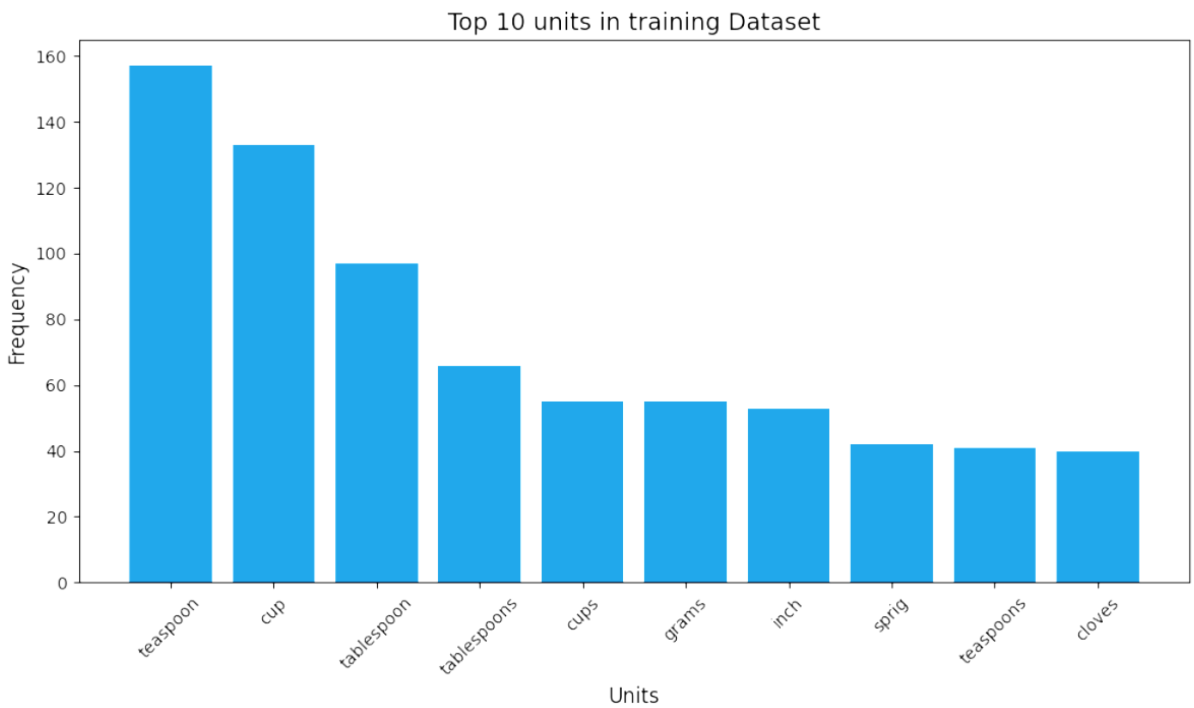
The given data is in JSON format, representing a **structured recipe ingredient list** with **Named Entity Recognition (NER) labels**. Below is a breakdown of the data fields:

```
[
    {
        "input": "6 Karela Bitter Gourd Pavakkai Salt 1 Onion 3 tablespoon Gram flour besan 2 teaspoons Turmeric powder Haldi Red
        "pos": "quantity ingredient ingredient ingredient ingredient ingredient quantity ingredient quantity unit ingredient ingr
    },
    {
      "input": "2-1/2 cups rice cooked 3 tomatoes teaspoons BC Belle Bhat powder 1 teaspoon chickpea lentils 1/2 cumin seeds whit
      "pos": "quantity unit ingredient ingredient quantity ingredient unit ingredient ingredient ingredient ingredient quantity u
    }
]
```

Key	Description
input	Contains a raw ingredient list from a recipe.
pos	Represents the corresponding part-of-speech (POS) tags or NER labels, identifying quantities, ingredients, and units.

Identifying Key Entities in Recipe Data – Report Document





7.2 *Evaluation of Training Dataset using CRF model* [4 marks]

Evaluate on training dataset using CRF by using flat classification report and confusion matrix

```
# Evaluate the CRF Model on the training dataset
from sklearn_crfsuite import metrics

# Predict labels on the training data
y_train_pred = crf_model.predict(X_train_weighted_features)

# Evaluate using classification report
print("Classification Report on Training Set:")
print(metrics.flat_classification_report(y_pred=y_train_pred, y_true=y_train_labels))
```

```
Classification Report on Training Set:
              precision    recall  f1-score   support

  ingredient       1.00      1.00      1.00      5229
    quantity       1.00      0.99      1.00       965
        unit       0.99      1.00      1.00       815

    accuracy                           1.00      7009
   macro avg       1.00      1.00      1.00      7009
weighted avg       1.00      1.00      1.00      7009
```

```
# specify the flat classification report by using training data for evaluation
from sklearn_crfsuite import metrics

# Predict labels on training data
y_train_pred = crf_model.predict(X_train_weighted_features)

# Generate classification report
train_report = metrics.flat_classification_report(
    y_true=y_train_labels,
    y_pred=y_train_pred,
    labels=["quantity", "unit", "ingredient"],  # You can reorder or customize
    digits=4
)

# Print the report
print("Flat Classification Report on Training Data:\n")
print(train_report)
```

```
Flat Classification Report on Training Data:

              precision    recall  f1-score   support

    quantity     1.0000    0.9938    0.9969       965
        unit     0.9927    1.0000    0.9963       815
  ingredient     1.0000    1.0000    1.0000      5229

    accuracy                         0.9991      7009
   macro avg     0.9976    0.9979    0.9977      7009
weighted avg     0.9992    0.9991    0.9991      7009
```

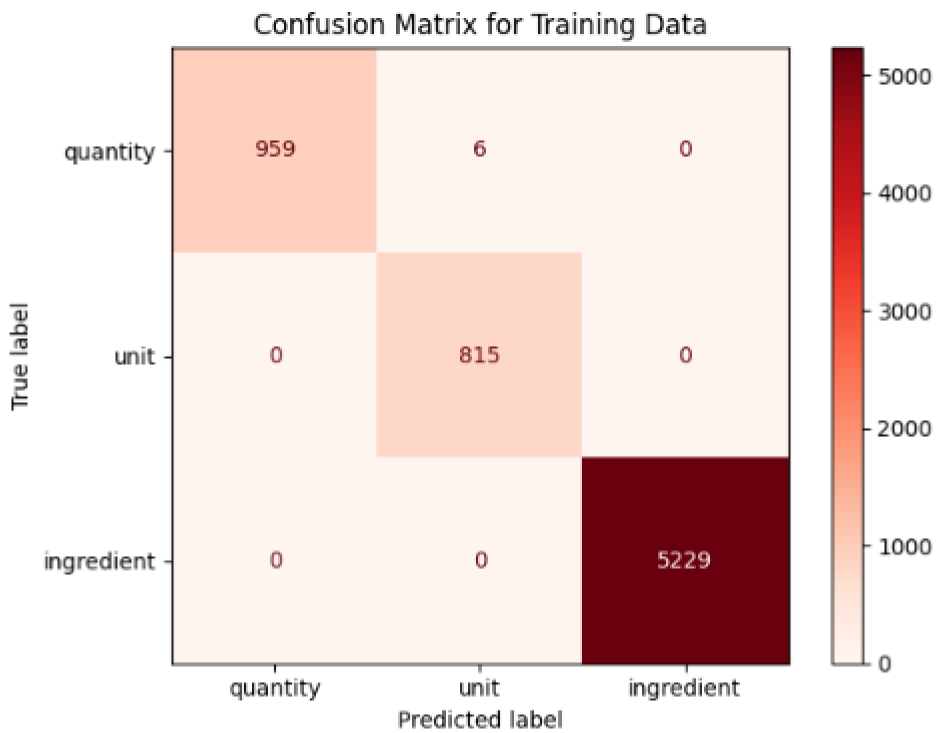
```python
# create a confusion matrix on training datset
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Flatten the lists of label sequences
y_train_true_flat = [label for seq in y_train_labels for label in seq]
y_train_pred_flat = [label for seq in y_train_pred for label in seq]

# Define label order
label_order = ["quantity", "unit", "ingredient"]

# Create confusion matrix
cm = confusion_matrix(y_train_true_flat, y_train_pred_flat, labels=label_order)

# Display confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=label_order)
disp.plot(cmap=plt.cm.Reds, values_format='d')
plt.title("Confusion Matrix for Training Data")
plt.tight_layout()
plt.show()
```



```
# specify flat classification report
from sklearn_crfsuite import metrics

# Generate classification report on validation data
val_report = metrics.flat_classification_report(
    y_true=y_val_labels,
    y_pred=y_val_pred,
    labels=["quantity", "unit", "ingredient"],  # Specify consistent label order
    digits=4
)

# Print the report
print("Flat Classification Report on Validation Data:\n")
print(val_report)
```

```
Flat Classification Report on Validation Data:

              precision    recall  f1-score   support

    quantity     0.9952    0.9928    0.9940       416
        unit     0.9914    0.9942    0.9928       347
  ingredient     1.0000    1.0000    1.0000      2146

    accuracy                         0.9983      2909
   macro avg     0.9955    0.9957    0.9956      2909
weighted avg     0.9983    0.9983    0.9983      2909
```

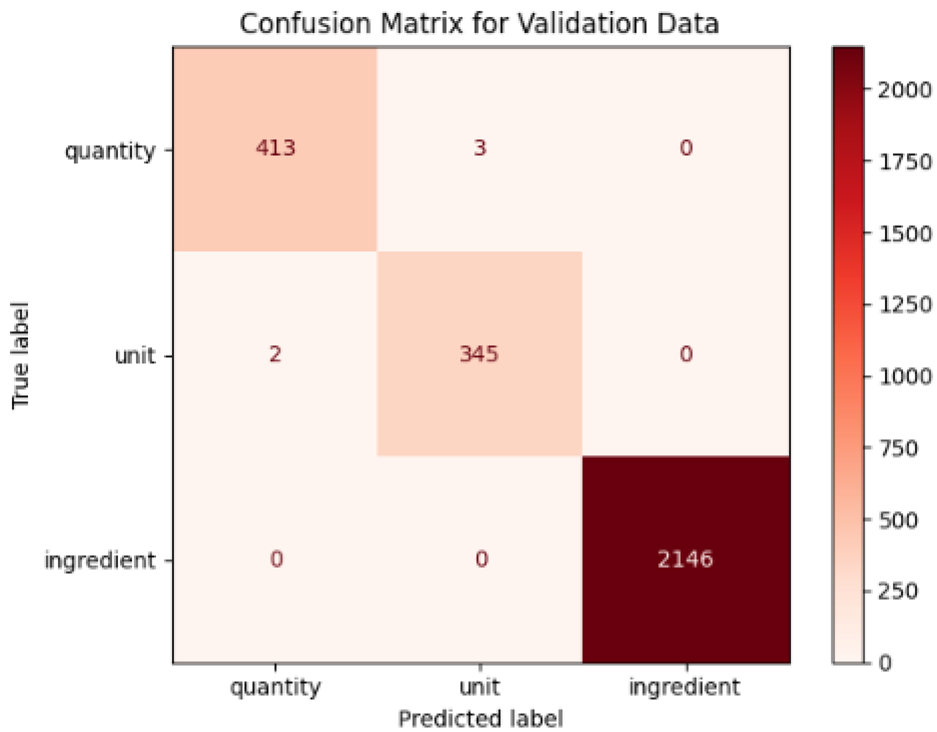
```python
# create a confusion matrix on validation dataset
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Flatten the predicted and true label sequences
y_val_true_flat = [label for seq in y_val_labels for label in seq]
y_val_pred_flat = [label for seq in y_val_pred for label in seq]

# Define label order for consistency
label_order = ["quantity", "unit", "ingredient"]

# Create confusion matrix
cm_val = confusion_matrix(y_val_true_flat, y_val_pred_flat, labels=label_order)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm_val, display_labels=label_order)
disp.plot(cmap=plt.cm.Reds, values_format='d')
plt.title("Confusion Matrix for Validation Data")
plt.tight_layout()
plt.show()
```



```python
# Create DataFrame and Print Overall Accuracy
import pandas as pd

# Step 1: Create DataFrame from error details
error_df = pd.DataFrame(detailed_error_data)

# Step 2: Calculate total number of tokens in validation data
total_val_tokens = sum(len(seq) for seq in y_val_labels)

# Step 3: Calculate total number of errors
num_errors = len(error_df)

# Step 4: Compute accuracy
accuracy = (total_val_tokens - num_errors) / total_val_tokens

# Step 5: Output
print("Validation Error DataFrame:")
print(error_df.head())

print(f"\n Overall Accuracy on Validation Data: {accuracy:.4f}")
```

```
Validation Error DataFrame:
    token true_label predicted_label prev_token next_token  class_weight
0     for   quantity            unit        Oil   kneading      2.421071
1  Little   quantity            unit       meat      extra      2.421071
2     for   quantity            unit      Honey    glazing      2.421071
3       a       unit        quantity      Haldi      pinch      2.866667
4      to       unit        quantity         10         12      2.866667

 Overall Accuracy on Validation Data: 0.9983
```

```python
# Analyse errors found in the validation data by each label
# and display their class weights along with accuracy
# and display the error dataframe with token, previous token, next token, true label, predicted label and context
# Count errors by true label
error_by_label = error_df['true_label'].value_counts().to_dict()

# Total tokens per label in validation set
from collections import Counter

# Flatten true labels from validation set
y_val_flat = [label for seq in y_val_labels for label in seq]
val_label_counts = Counter(y_val_flat)

# Compute per-label accuracy and show class weights
print("Error Analysis by Label:\n")
print(f"{'Label':<12}{'Errors':<10}{'Total':<10}{'Accuracy':<12}{'Class Weight'}")
print("-" * 55)

for label in val_label_counts:
    total = val_label_counts[label]
    errors = error_by_label.get(label, 0)
    acc = (total - errors) / total
    weight = penalized_weights.get(label, 1.0)
    print(f"{label:<12}{errors:<10}{total:<10}{acc:<12.4f}{weight:.4f}")
```

```
Error Analysis by Label:

Label       Errors    Total     Accuracy    Class Weight
-------------------------------------------------------
quantity    3         416       0.9928      2.4211
unit        2         347       0.9942      2.8667
ingredient  0         2146      1.0000      0.2234
```

Identifying Key Entities in Recipe Data – Report Document

9.2 *Provide insights from the validation dataset* [2 marks]

italicized text [Write your answer]

Key observations

- Ingredient Predictions Are Perfect
  - The model achieved 100% accuracy for the ingredient label with zero errors.
  - However, its low class weight (0.2234) indicates it's the most frequent label in the dataset, which might suggest model bias or overfitting for this class.
- Slight Errors in Quantity and Unit
  - The model made only 3 errors in predicting quantity and 2 errors in unit.
  - Despite their lower frequency, their class weights (2.42 and 2.87) helped the model learn to prioritize them.
  - These results confirm that the class weighting strategy was effective in addressing class imbalance.
- Strong Overall Generalization
  - With all classes achieving over 99% accuracy, the model demonstrates excellent performance and generalization on the validation set.

Recommendations:

- Consider cross-validation for deeper model's robustness analysis.
- Use real-world or augmented data to boost under-represented classes like unit and quantity.
- Keep monitoring performance when scaling to new recipes or languages.