Machine Translation

Final Project Report

Anwesha Das

Deepti Panuganti

Dhruvil Shah

Aryeh Stiefel

Extension One: Linear Regression with KNN

We got the ideas for our extensions from the *Findings of the 2014 Workshop on Statistical Machine Translation* paper, by Bojar et. al. The first extension we implemented was Linear Regression, using the Python library, `scikit`. Using this method, on top of KNN, we were able to beat the baseline. Linear regression generated weights for the features and then recomputed KNN using the weighted values for each feature.

We chose this extension because it built off of the baseline. It took the baseline and changed the weighted importance of the features, hence improving the accuracy.

We improved on this by adding new features. The different features we tried were as follows:

1. Percentage of content words in source sentence.
2. Percentage of content words in target sentence.
3. The ratio of percentages of the of content words in source and target.
4. The number of untranslated words in the target sentence.

Adding features 1 and 2 didn't improve the accuracy by a lot. We then tried adding them instead as a ratio (feature 3 in the above list). This also, didn't significantly improve the results, suggesting that the content words weren't very indicative of the goodness of the translation.

The number of untranslated words, was an informative feature and improved the accuracy to 48%.

Extension Two: Decision Trees

In addition to linear regression, another way of finding the importance of the features is through decision trees. We chose to implement decision trees with the weighted feature values yielded by linear regression. It was

implemented using the Python library, `scikit`.

First we fit a linear regression model to the data and used the coefficients returned as weights for each feature. Then, using this weighted values for each feature we trained a decision tree, picking the features that yielded the highest information gain. We used the decision tree to then score the test sentences. This extension also succeeded in beating the baseline.

Extension Three: Support Vector Machines with RBF Kernel

The third extension we implemented was Support Vector Machines (SVMs) with radial basis function (RBF) kernel. We again used the Python library, `scikit`. We chose SVMs because it appeared in a lot of papers and yielded good results.

We used grid search and three-fold-cross-validation to find the optimal parameters, `c` and `gamma`. We used three-fold-cross-validation because we tried various k-folds and three yielded the highest accuracy.

Extension Four: Random Forests

The fourth extension we implemented was Random Forests. We chose random forests because it is a way of building upon decision trees, in an ensemble fashion, taking the majority class.

We used a random forest classifier optimizing the parameters `n_estimators`, `criterion`, and `min_samples_leaf` using grid search with cross validation. A five-fold-cross-validation yielded our best results. Using random forests we were able to achieve a 50% accuracy.

Results

Method	% Accuracy
Default	23.1111
Baseline KNN	44.0000
Linear Regression with KNN	45.5556
Linear Regression with KNN and New Features	48.00
Decision Trees	34.6667
Decision Trees with Linear Regression	46.8889
Random Forests	50.0000
Support Vector Machines	38.4444
Support Vector Regression	45.5556
Naive Bayes Classifier	41.1111