Machine Translation

Final Project Report

Anwesha Das
Deepti Panuganti
Dhruvil Shah
Aryeh Stiefel

What We Did

Default

For the default system we split up the sentences based on their assigned scores. We took every sentence with a score of 1, then of `2` and then of `3`. For each of those clusters we computed the average value. Then, for each test sentence we computed its value and compared it to each of the cluster's values. The value that the sentence was closest to determined the score for that sentence.

Baseline

Then, for the baseline system we performed k nearest neighbors. We chose KNN because of its similarity to clustering, our default system. For each test sentence, we found the k closest values. Then, we took the majority score of the k nearest sentences and assigned it to the test sentence. We tried varying values of k from one to twenty one. We chose to stop increasing k because we stopped finding any significant improvement.

Extension One: Linear Regression with KNN

We then began implementing various extensions. We got the ideas for them 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.

Extension Two: Decision Trees

The second extension we implemented was Decision Trees, again using the Python library, `scikit`. First 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.

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 γ. 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 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.

In addition to implementing the `default`, `baseline` and extensions we also implemented a Naive Bayes classifier and Decision Trees without linear regression. Both were unsuccessful in beating the baseline. The decision tree implementation was similar to the above but without the coefficients from the linear regression model. Because all the features were equally weighted, it did not capture the difference in importance. In addition, the Naive Bayes classifier took a count of all the words in each of the scored training sentences. Based on the count it computed the probability that the sentence belonged to either the class of 1, 2 or 3. Then, for each test sentence we computed the probability that it belonged in one of the classes and assigned that class to it.

Results

Method	% Accuracy
Default	23.1111
Baseline KNN	44.0000
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
Linear Regression with KNN	45.5556

The baseline yielded us a 44% accuracy. We chose that score because it was achieved by a logical expansion of clustering. Our first extension was linear regression with KNN and it succeeded in beating the baseline. We chose this extension because it built off of the baseline. It took the baseline and changed the weighted importance of the features. In addition to linear regression, another way of finding the importance of the features is through decision trees. Thus, we chose to implement that method with the weighted feature values yielded by linear regression. This extension also succeeded in beating the baseline. We then began looking in different directions and noticed that SVMs were mentioned throughout many papers. As a result, we implemented SVMs with an RBF kernel to beat the baseline. The last extension we implemented was random forests and it performed the best. We chose random forests because it is a way of building upon decision trees, in an ensemble fashion, taking the majority class. Using random forests we were able to achieve a 50% accuracy.