What's in a Word: Classification of English-Translated Texts by Source Language

Katherine Wu
Stanford University
450 Serra Mall
kjwu00@stanford.edu

Eunji Lee
Stanford University
450 Serra Mall
eunjilee1@stanford.edu

Jerry Qu *
Stanford University
450 Serra Mall
jerryqu@stanford.edu

1. Introduction

As more and more people from many nations learn to speak English, an increasing amount of text is translated from foreign languages to English. We are building a system that facilitates this cross-cultural communication by identifying the source languages of translated texts from around the world. We have built a corpus of professionally-translated short stories in six different source languages, and our goal is for our model to classify brief passages of those stories by their original language.

1.1. Motivation

In many ways, language can be seen as a product of the world surrounding it. Different languages came into play because people needed to express, argue, negotiate, and question. As time passed, they evolved based on changing circumstances and shifting borders. As the world grew increasingly closer in the past few centuries, English has emerged as a "universal" language: one that many people know and speak [9]. This is why many texts have been translated into English; as a result, people from all over the world have access to each other's cultures, stories, and experiences.

However, a common issue with cross-cultural communication via text is the introduction of "translationese", which is a linguistic term used to refer to an original language's signature left in a translated text [8]. Some examples of translationese include certain sentence structures, the frequent use of a particular word or part of speech, etc.

It follows that well-translated texts have less noticeable translationese, while poorly-translated texts have very obvious traces of the source language in the resulting translation (a human-identifiable example of translationese could be an unnatural sentence structure or the frequent misuse of articles). A system that can classify translated text by source language would be useful in evaluating the effectiveness of both human translators-in-training and machine translation models; an unusually high accuracy in classifying texts by a given translator may indicate poor translation skills.

While human translators often have professional certifications and years of education, machine translation models often lack the sophistication to capture the nuances in translating literature [15]. This would provide a metric to evaluate models on not only its ability to accurately convey the information contained in a text, but also how "elegantly" it can do so: a factor that is often difficult to measure.

1.2. Task

Our task is to automatically detect the source language of a text that was translated to English (or possibly originally written in English itself). This could potentially help place cultural context to "unknown" texts that may have been translated, in addition to helping evaluate the effectiveness of machine translation models that are developed in the future.

1.3. Scope

We aim to create a system that can classify an 100 word fragment of an English-translated fictional story into one of six source languages (L1s): Spanish, French, Korean, Portugeuese, Russian, or English (original language).

1.4. Input and Output

The input of our system will be the 100 word fragment the text once it has been translated into English. Our output will be the predicted original language of the text. For example, an input can be an 100 word .txt file of a fragment of the English language short story "The Match" by Colson Whitehead, and the output would be English, because the short story was originally written in English. Another example is the input being the raw English-language translated text of the short story "The Shower" by Hwang Seon-Won, with the output being Korean, because the story was originally written in Korean.

*Jerry is our project TA. Thanks for the help!

1.5. Success

The success of our system is determined by the percentage of the texts which are correctly classified into their original language.

2. Related Work

Many previous attempts to build systems to classify English-translated texts by their original languages used a support vector machine (SVM). However, in addition to a SVM, we are implementing logistic regression and a recurrent neural network. Similarly to many previous attempts, we use n-gram features (words-unigram and words-bigram), however, we do not use different text representations (such as part of speech) or document level features that other attempts have used. Furthermore, classification among four or more classes has rarely been attempted, and our problem involves six class classification.

Baroni and Bernardini [6] worked on the binary classification problem of classifying magazine articles as either translated into or originally written in Italian. For the translated texts, they used texts which were originally written in English, Arabic, French, Spanish, and Russian, then translated into Italian. Baroni and Bernardini considered different features to use to represent the text, considering different n-gram models (unigram, bigram, and trigram) and different unit types (wordform, lemma, part-of-speech tag, mixed). This results in 24 different models, which they then tested different combinations of. Their best model was unigram wordform, bigram lemma, bigram mixed, trigram tagged, and it achieved 77.5% accuracy, 98.5% precision, 55% recall, and 71.3% F. This approach is complementary to our approach: one of our approaches is also the SVM, and we also try using different n-grams. However, their work also differs from ours, as we are attempting a more difficult problem (6-class classification in comparison to their two-class classification). Furthermore, we are exploring more models, whereas Baroni and Bernardini explore different text representations and combinations of different SVMs with different features.

Lynch and Vogel [13] describe a classification system which classifies novels into one of four source languages, using an SVM classifier, a Simple Logistic classifier and the Naive Bayes classifier. Their features included 19 document level features (average sentence length, average word length, two readability metrics, word types : total works, numbers : total words, finite verbs : total words, prepositions : total words, conjunctions : total words, open-class words : total words, discourse markers : total words, nouns : total words, open-class words : closed-class words, simple sentences : complex sentences, pronouns : total words, lemmas : total words, simple sentences : complex sentences, simple sentences : total sentences, complex sentences : total sentences). They also used n-gram features (word-unigrams and part-of-speech bigrams). Like Baroni and Bernardini and this paper, they implemented an SVM, which achieved 66% accuracy. However, they found that a logistic regression, which we also implement, performed the best, achieving an accuracy of 68%. They also implemented a Naive Bayes classifier, which achieved the lowest accuracy of the three models, 54%. Lynch and Vogel's approach is both complementary and orthogonal to our approach: we also use an SVM and a logistic regression as two of our approaches. However, their work differs from ours, as we are attempting a more difficult problem (6-class classification in comparison to their four-class classification). Furthermore, we explore the use of an RNN while Lynch and Vogel explore the use of a Naive Bayes model. Furthermore, Baroni and Bernardini explore including many document level features and part-of-speech-bigrams in addition to word-unigram features whereas we use word-unigram and word-bigram features.

Kurokawa, Goutte, and Isabelle [12] classify records of government proceedings as either translated or not using an SVM classifier. Like Baroni and Bernardini (and partly like Lynch and Vogel), they consider different text representations (word, lemma, part of speech, and mixed). Like Baroni and Bernardini, Lynch and Vogel, and this paper, they consider different n-grams. They found that different text representations from best accuracy to worst accuracy was: word, lemma, mixed, part of speech. They achieve an accuracy of 90% when classifying full documents and 77% when classifying single sentences. This is complementary to our approach, as we also used an SVM to classify the texts. This showed us that we could use shorter amounts of text (sentences in comparison to blocks) and still achieve decent (although slightly worse) classification. However, we did not explore different text representations: they found that the text representation which we employed, word, performed the best. Furthermore, we are exploring both a more complex problem and testing more models.

Illisei, Inkpen, Pastor, and Mitkov [11] worked on the two-class classification problem distinguishing between translated and non-translated texts. The features they use include some which were similar to Lynch and Vogel's document level features: the use the proportion in texts of grammatical words, nouns, finite verbs, auxiliary verbs, adjectives, adverbs, numerals, pronouns, prepositions, determiners, conjunctions, and the proportion of grammatical words (determiners, prepositions, auxiliary verbs, pronouns, and interjections) to lexical words (nouns, verbs, adjectives, adverbs and numerals), and average sentence length. They also used the sentence depth as the parse three depth; the proportion of simple sentences, complex sentences, and sentences without any finite verb ambiguity as the average of senses per word; word length as the proportion of sylla-

bles per word; lexical richness; and information load as the proportion of lexical words to tokens. Illisei, Inkpen, Pastor, and Mitkov considered seven classifiers: Jrip, Decision Tree, Naive Bayes, BayesNet, SVM, Simple Logistic and a Vote meta-classifier with the Majority Voting combination rule, which considers the Decision Tree, Jrip and Simple Logistic classifiers output. Their meta-classifier achieved the highest accuracy (85.81%), followed by the decision tree (81.76%), the simple logistic (80.41%). The SVM achieved an accuracy of 73.65%. This work is partly in line with ours, as they also considered using a logistic regression and SVM. However, the work differs from ours, as they considered many more models than me did, and considered document level features while we considered word n-grams. Finally, they work on binary text classification, while we worked on a harder classification problem, 6 class classification.

3. Dataset and Corpus Gathering

Our dataset is a collection of 100-word sequences from various short stories translated from a source language to English (or written in English originally).

We collected our dataset by webscraping texts from *The New York Times* [5] and *The Atlantic* [1] (for original English-language short stories), Project Gutenberg [2] (for Spanish, French, Portuguese, and Russian short stories), and *The Literature Translation Project of Korea* [4]. The works used are generally all written in the past 200 years and are fictional. See Table 1 for examples of works in the corpus.

When choosing the short stories to include in our dataset, we ensured that each was translated by a professional translator in order to provide consistency in the quality of the translations. Furthermore, we made sure to include works by multiple authors and translations by mutliple translators, in order to reduce author and translator specific signals. See Table 2 for a more detailed description of the dataset.

As shown in Table 2, each of the six language categories has a total of 100,000 words total, with 1,000 datapoints consisting of 100 words each. Therefore, there is a total of 6,000 datapoints in our dataset. We clean the raw text of our dataset to lowercase all characters and eliminate all punctuation, special characters, and Unicode characters. A single datapoint consists of the cleaned text of a consecutive and cleaned 100-word sequence of text and its original language label. See Figure 1 for an example of a datapoint where the source language (label) is Korean.

SL	Title	Author	Translator
English	"Snowbound"	Due	n/a
	"Caramel"	Jones	n/a
	"Blood Tide"	Hopp	n/a
Spanish	"A True Hildago"	Coloma	Binns
	"Juanita la Larga"	Valera	Fedorchek
	"City of the Discreet"	Baroja	Fassett
Portuguese	"Dragons"	Abreu	Treece
	"Order of the Day"	Cony	Patai
	"Bald Island"	Dourado	Lowe
French	"The Necklace"	Maupassant	Hawthorne
	"The Mummy's Foot"	Gautier	Hearn
	"Claude Gueux"	Hugo	Nottingham
Russian	"Crime and Punishment"	Dostoevsky	Garnette
	"Dead Souls"	Gogol	Magure
	"Anna Karenina"	Tolstoy	Barlett
Korean	"A Shower"	Hwang	Anthony
	"Home"	Hyun	Russell
	"Flood"	Han	Kim

Table 1. *This table shows a sampling of works from our corpus (three works per source language), along with their authors and translators, to give an idea of what types of short stories are in our corpus.*

Source Language	# Texts	Average Word Count Per Text
English	25	4,000
Spanish	22	4,545
Portuguese	22	4,545
French	21	4,761
Russian	21	4,761
Korean	32	3,125

Table 2. *This table shows a summary of the number of works per corpus and the average word count of each work. Some languages' short stories (e.g. Korean) have a shorter average length than others, so more short stories were gathered.*

```
['the', 'station', 'the', 'scene', 'reminded', 'him',
'of', 'a', 'child', 'afraid', 'to', 'lose', 'his',
'mother', 'manki', 'let', 'out', 'a', 'sigh', 'as',
'he', 'watched', 'them', 'silently', 'he', 'felt',
'pity,' for', 'bongwoo', 'who', 'was', 'hopelessly',
'infatuated', 'and', 'who', 'couldn't', 'help', 'it',
'at', 'the', 'same', 'time', 'he', 'felt',
'confined', 'and', 'trapped', 'since', 'there',
'was'], 'Korean'
```

Figure 1. *An example of the input and output values of a datapoint.*

4. Methods

Our task is to automatically detect the source language of a text that was translated to English (or possibly originally written in English itself). In particular, we are using 100-word blocks of texts whose original languages are English, Spanish, French, Korean, Russian, and Portuguese as the input to our model. We chose to use 100-word blocks of text by utilizing a human oracle: when using less than 100

words, there was not enough information in the block for even a human oracle to identify the source language with an accuracy better than the random guess baseline, but when utilizing more than 100 words, we had less than 1,000 datapoints per language, which is generally not enough for training. Each (block, language) tuple represents one data point.

We explore three primary models: multi-class logistic regression, support vector machine (SVM), and recurrent neural network (RNN). We compare the accuracies of these models to our two baseline models, random guess and k-nearest neighbor (KNN), and our oracle, human classification. For our three primary models and our baseline KNN model, we test both unigram features and bigram features. For multi-class logistic regression, SVM, and KNN, we use a bag-of-words model.

4.1. Feature Extraction

We chose to run each of our models (other than RNN) using both unigram (bag-of-words) and bigram features (as opposed to parts of speech) due to previous research demonstrating that using variants of word-related features yielded the most accurate results (see **Related Works** for more details).

By increasing the value of n in an n-grams approach, we increase the complexity of each feature. Theoretically, this gives us more information about the usage of the words in order; however, the features may be too specific to each individual datapoint, since we are only using 100 words per datapoint. To find the balance between complexity and specificity, we tried both uni-gram and bi-gram approaches.

For RNN, we used the sequence of words, as each hidden layer would then process the next word.

4.2. Baselines

4.2.1 Random Guessing

Our first baseline for our system us randomly guessing the original language for the short-story, which achieved an (expected) accuracy of 16.7% (1/6). This provides a very basic lower bound for our performance, since any classifier we create should perform better than random choice.

4.2.2 KNN

Our second baseline is a k-nearest neighbor (KNN) model. For the KNN model, we store the features and label for each training data point. When making a prediction for a new text, we find the K points which are closest to the new testing point in the feature space, then choose the label that the plurality of the points have. To tune the k hyperparameter, we test using $k = 3$, 5, 7, and 9, using both uni-gram and bi-gram features. KNN model achieved the highest overall accuracy of 46.167% using bi-gram features and $k = 9$, providing a second lower bound for the performance of our three primary models.

4.3. Oracle

4.3.1 Human Classification

As an oracle, we used a human who was provided a list of ten common names for each language and a list of ten major cities in the country from which the language originates.

For example, the human oracle was given:

```
French:

[Cities: 'Paris', 'Bordeaux',
'Cannes', 'Lille', 'Lyon', 'Marseille',
'Montpellier', 'Nantes', 'Avignon']

[Names: 'Louise', 'Alice', 'Chloe',
'Ines', 'Jeanne', 'Gabriel', 'Adam',
'Raphael', 'Paul', 'Louis']
```

in addition to full-length French-to-English translated short stories as "extra information" to aid in classifying French short stories.

Instead of 100-word sequences, we gave the oracle the full texts of 54 short stories, ranging from 5,000 words to 14,000 words. There were 9 texts for each of the six languages (Spanish, French, Korean, Portuguese, Russian, and English), with approximately the same number of total words per language. This additional information helps set an upper bound to compare our model to.

The human read different texts and predicted which of the six languages was the source language for each of the texts. The oracle achieved an accuracy of 90.7%, classifying 49/54 texts correctly.

4.4. Primary Models

4.4.1 Multi-class Logistic Regression

The first of our primary models is multi-class logistic regression. We chose multi-class logistic regression as one of our models because it is one of the computationally least expensive to run while also giving accurate results; its training time for this dataset is significantly shorter than for linear SVM. We use multinomial logistic regression, which is a type of logistic regression generalizable to categorical data.

Similar to binary logistic regression, this model creates a linear predictor function $f(k, i)$ to predict the probability of observation i having label k, based on seen datapoints during training. In this case, k is an integer in the range 0 to 5 inclusive, representing each of the six languages, and i is a feature vector representing word counts (either with uni-gram or bi-gram). In the following multinomial logistic

regression equation, let $\beta_{m,k}$ be the regression coefficient assigned to the mth feature and kth label:

$$f(k,i) = \beta_{0,k} + \beta_{1,k}x_{1,i} + \beta_{2,k}x_{2,i} + \cdots + \beta_{M,k}x_{M,i}$$

By writing the regression coefficients assigned to label k as $\boldsymbol{\beta}_k$ and the features associated with input i as $\boldsymbol{x}_i$, we can condense the above equation to:

$$f(k,i) = \boldsymbol{\beta}_k \cdot \boldsymbol{x}_i$$

To estimate the values of $\boldsymbol{\beta}_k$, a log-linear model is used: we take the log of the probability that we see a particular output label using our current linear predictor, in addition to a normalization factor. Then, we can apply the softmax function to normalize the outputs to probabilities that sum to 1, which is a key part of modifying binary logistic regression to support multiclass classification. The softmax function takes in an $N-$dimensional vector of real numbers and normalizes it to a probability distribution with N probabilities:

$$p_i = \frac{e^{a_i}}{\sum_{k=1}^{6} e_k^a}$$

Let l be a given label in the range $[0, 5]$, and Y_i as the model's output classification given the feature vector i. Based on our logistic regression equation, we can therefore write the probability that $Y_i = l$ as:

$$Pr(Y_i = l) = \frac{e^{\boldsymbol{\beta}_l \cdot \boldsymbol{x}_i}}{\sum_{k=1}^{6} \boldsymbol{\beta}_k \cdot \boldsymbol{x}_i}$$

We will therefore receive 6 different probabilities, and the output of this model (i.e. the predicted label) will be the value of l for which $Pr(Y_i = l)$ is the highest.

We also tuned the regularization parameter C, which reduces overfitting by penalizing large weight coefficients. This allows us to optimize our models.

We used ScikitLearn's LogisticRegression model with the `multi_class` parameter set to `'multinomial'`. L2 regularization is applied by default via sklearn, but we parameter-tuned with L1 and L2 regularization and without regularization (see **Results** for a comparison of regularization parameter results).

4.4.2 SVM

The second of our primary models is SVM. We chose to use SVM because it is the most common approach taken in simpler versions of solving this problem (see **Related Work** for more details). SVM has generally been the method of choice in solving the language classification problem (and classification problems in general), because the use of the hinge loss function generally makes the model less sensitive to outlier points when training, compared to logistic regression. This is because logistic loss diverges faster than hinge loss. See comparison of logistic loss versus hinge loss in Figure 2.

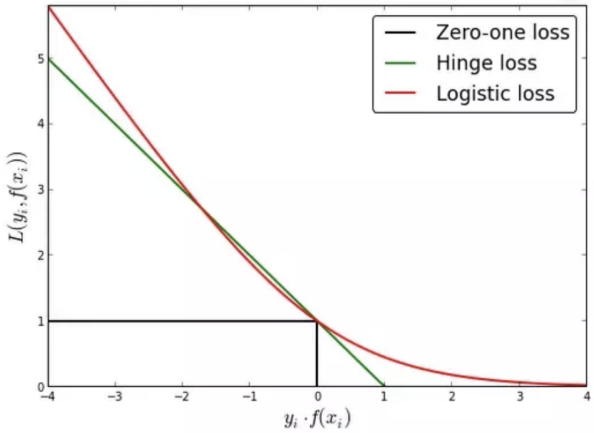

Figure 2. A comparison of logistic versus hinge loss [14].

Another minor difference, which is unlikely to have a large effect in accuracy, is that logistic loss will never reach zero, no matter how confidently a point is classified, while the hinge function will. This could result in small differences in accuracy, but only marginally [**?**]. See comparison of logistic loss versus hinge loss in .

In order to support multiclass classification, we break the 6-class classification problem into several binary classification problems via a one-versus-all (OVA) approach. In an OVA approach, we consider each possible class l (i.e. language) as a binary classification: i.e. determining whether a given input classifies as l or not. When using SVMs specifically, the class l that maximizes the margin is selected as the output.

We have 6 languages and therefore 6 different classes, so we build 6 different binary SVM classifiers as explained above to accomplish multiclass classification. For each of these classifiers, the optimal hyperplane is found, such that it separates the data points between the two categories for each l: original language is likely l or not.

We tuned two parameters for our SVM model: the kernel type and the regularization parameter.

For the first hyperparameter, we trained using linear, polynomial, and radial basis function (RBF) kernels. A linear kernel works best for linearly separable data (i.e. data that can be separated into classes simply by drawing a line), while polynomial kernels divide non-linearly separable data by comparing combinations of input features for similarity. In a binary classifier with a linear kernel, an SVM determines two hyperplanes:

$$w \cdot x - b = 1$$

where anything above or on this boundary is labeled 1, and

$$w \cdot x - b = -1$$

where anything below or on this boundary is labeled -1.

To determine the placements of these hyperplanes, we maximize the absolute distance between the two, constrained so that no points fall into this margin. Samples lying along the margin are referred to as support vectors.

A polynomial kernel uses both the given feature vector of an input sample and combinations of these feature vectors, and is typically used for non-linearly separable data.

The polynomial kernel is defined by:

$$K(x, y) = (x^\intercal y + c)^d)$$

where d is the degree of the polynomial, x and y are feature vectors in the input space, and c is a free parameter that determines the relative importance of higher versus lower order terms in the polynomial.

The RBF kernel is defined by the equation:

$$K(\boldsymbol{x}, \boldsymbol{x'}) = exp(-\frac{||\boldsymbol{x} - \boldsymbol{x'}||^2}{2\sigma^2})$$

where $\boldsymbol{x}, \boldsymbol{x'}$ are two different feature vectors. The expression $||\boldsymbol{x} - \boldsymbol{x'}||^2$ is the squared Euclidean distance between two feature vectors. Therefore, an RBF kernel can be conceptualized as measuring the similarity between two features $\boldsymbol{x}$ and $\boldsymbol{x'}$.

A linear or polynomial kernel with degree 2 is typically used in solving NLP problems, but RBF kernels are more common with SVMs in general. Therefore, we tuned our SVM model by trying all three kernels to determine which would give the highest accuracy [7].

We also tuned the regularization parameter, referred to as C in ScikitLearn's model. C is a measure of how much a misclassification is penalized in training. As C grows larger, the smaller the model's hyperplane will be, which may result in overfitting. As C becomes smaller, the model's hyperplane becomes larger, which means that the training accuracy may become lower, but the model is more generalizable. Typically, varying C will increase or decrease the amount of overfitting that occurs, and C is usually changed on an order of magnitude.

4.4.3 RNN

Our third primary model is an RNN. We chose to employ an RNN as this model has a "memory" of what has been calculated before. The output for every element depends on the output for the previous elements, which more closely mimics the way in which the meaning of one word depends on the meaning of previous words and may allow us to differences in grammar patterns between different languages[1].

More specifically, the RNN consists of a chain of a repeating modules as shown in Figure 3. The i^{th} module takes in x_i, a vector of shape [V,1] (where there are V words in our vocabulary) which represents the word embedding for the i^{th} word in the text. x_i has all of its values set to 0 except for a 1 at the position corresponding to the i^{th} word of the text. The module also takes in the previous hidden state h_{i-1}, which is a [V, 1] vector, so that we are capturing the maximum amount of information. For the first module, the hidden state is set to $\vec{0}$.

The module then computes the current hidden state h_i. To do so, we calculate

$$h_i = f_W(h_{i-1}, x_i) \quad (1)$$
$$= softmax(W_{hh}h_{i-1} + W_{xh}x_i). \quad (2)$$

W_{ih} and W_{xh} are [V, V] matrices which we will learn through the training process. The next hidden state represents the new strength of our knowledge that the text came from each of the 3 possible original languages, using the new word and old state. For the final state, we end up with three values that represent the confidence of our knowledge that the given text is each of the three original languages. The output for the final layer is a [3, 1] vector representing the likelihood that the input text is in each of the three categories[3].

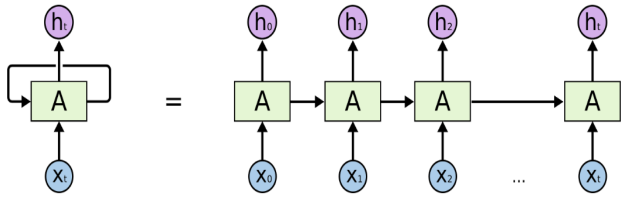

Figure 3. *This figure shows an unraveled RNN [3]*

Figure 1. An "unrolled" RNN demonstrating chained copies of the same network

For our loss function, we will use the softmax function with cross entropy loss. The softmax function takes in an $N-$dimensional vector of real numbers and normalizes it to a probability distribution with N probabilities:

$$p_i = \frac{e^{a_i}}{\sum_{k=1}^{N} e_k^a}$$

The softmax function is used in the final layer of our neural network because it outputs a probability distribution. For the loss function itself we use cross entropy loss, which calculates the distance between the model's belief of the output probability distribution determined by softmax and the actual original distribution:

$$H(y, p) = -\sum_i y_i \log p_i$$

Model	Val Accuracy
Logistic Regression (Uni-gram Features)	0.805
SVM (Uni-gram Features)	0.773
RNN	0.18

Table 3. *This table contains the validation accuracies of the best models using multiclass logistic regression, SVM, and RNN which we ran.*

	Value of C			
Regularization Type	**0.1**	**1**	**10**	**100**
L1	0.42	0.682	0.805	0.682
L2	0.687	0.783	0.805	0.783

Table 4. *This table shows the results for the logistic regression model with unigram features, varying the regularization type and value of regularization parameter C.*

	Value of C			
Regularization Type	**0.1**	**1**	**10**	**100**
L1	0.5066	0.6616	0.5066	0.69
L2	0.6983	0.7183	0.6983	0.725

Table 5. *This table shows the results for the logistic regression model with bigram features, varying the regularization type and value of regularization parameter C.*

5. Results and Analysis

Overall, we saw that the multi-class logistic regression using uni-gram features with C = 10 and L2 regularization performed the best, achieving an accuracy of 80.5% on the validation set and 83% on the testing set. In general, we saw that multi-class logistic regression performed comparatively to SVMs. Our RNN did not perform well, most likely due to the vanishing gradient problem Table 3.

5.1. Multi-class Logistic Regression

For multi-class logistic regression, we trained with both uni-gram and bi-gram features, tuning the regularization (L1 or L2) and the C hyperparameter (C = 0.01, 0.1, 1, and 10).

From a comparison of Table 4 and Table 5, we find that the uni-gram features produce higher accuracies. Furthermore, using C = 1 produces the highest accuracies, and in general, L2 regularization produces better results than L1 regularization.

The optimal model was uni-gram features with C = 1 and L2 regularization, which achieved 80.5% accuracy on the validation set (as shown in Table 3) and 83% on the testing set.

5.2. Support Vector Machine

For SVM with uni-gram features, we tuned the kernel and C hyperparameter, as shown in Table 6. We saw that the linear kernel performed the best, causing us to believe that the data is linearly seperable. RBF was the kernel that produced the second best accuracies. Polynomial degrees 2 and 3 produced the same accuracies. Interestingly, we saw that different values of C did not affect the performance of the linear kernel. However, for RBF, polynomial degree 2 and polynomial degree 3, we see that a value of C = 10 or 100 is optimal.

	Value of C			
Kernel Type	**0.1**	**1**	**10**	**100**
Linear	0.773	0.773	0.773	0.773
Polynomial (degree 2)	0.41	0.6483	0.7116	0.7116
Polynomial (degree 3)	0.41	0.6483	0.7116	0.7116
RBF	0.46	0.686	0.755	0.755

Table 6. *This table shows the results for the SVM model with unigram features, varying the kernel type and value of regularization parameter C.*

For SVM with bigram features with a linear kernel and C = 100, we achieved 77.3% accuracy on the validation set.

The most optimal SVM model was unigram features with kernel = linear and C = 10, which achieved 77.3% on the validation set (Table 3).

5.3. RNN

With a RNN with a hidden layer size of 1000 and a learning rate of 0.05 (a standard learning rate), we achieved a validation accuracy of 18% (Table 3). We did not tune the learning rate, as we believe that this error was caused by the vanishing gradient (see **Error Analysis** section 6.3 for more details) and tuning the step size would not resolve this issue.

5.4. Training Accuracies and Bigram Models

Notably, the bi-gram models achieved a training accuracy of 100%, while the uni-gram models achieved training accuracies between 5-15 % above the corresponding testing accuracy (over the validation set), with no uni-gram model's training accuracy reaching 100%. However, the bi-gram models performed significantly worse than the corresponding uni-gram models for logistic regression, with the best bi-gram model for logistic regression achieving 72.5% testing accuracy and the best uni-gram model achieving 80.5% testing accuracy. For SVM, the best bi-gram and uni-gram models performed comparably.

This indicates the possibility of overfitting in the case of logistic regression, especially as C increases (see **Error Analysis** section 6.2 for more details).

5.5. Testing on Best Model

With our validation set, the most optimal model (i.e. model with highest overall accuracy) is logistic regression with L2 regularization and $C = 10$. We ran our test set,

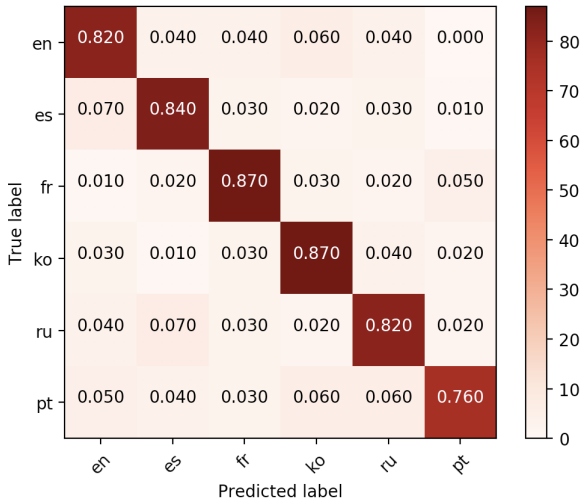

Figure 4. *A confusion matrix detailing accuracies for the most optimal model (as described in Section 5.4)*

which is comprised of 10% of our total dataset, and this optimal model reached an accuracy of 83%. See Figure 4 for a detailed account of accuracies.

We see from Figure 4 that Korean and French texts are most commonly classified to the correct source language, with 87% of texts being classified correctly. For Korean, this result was expected, because it is a non-romance language that is structured very differently than the other languages. We were surprised that French also had a high accuracy, given that it is a romance language with very similar grammar patterns and word usage as Spanish and Portuguese; this may mean that French has unique patterns in word usage that carry over into translated texts that other languages do not have.

The most commonly misclassified language is Portuguese, which had the lowest accuracy at 76%. This was expected due to Portuguese's similiarity to Spanish; both languages were developed in proximity to each other and share a root from Latin. However, the surprising result was that Portuguese was more commonly misclassified as Korean or Russian than Spanish or French, but the difference is very marginal (6% versus 4 % and 3% respectively) and may be accounted for by random chance.

5.6. Analysis

As expected, the most optimally tuned logistic regression model performed similarly to the most optimally tuned SVM model (with logistic regression performing only marginally better with a difference in accuracy of 3.2% on the validation set). While SVM was expected to perform marginally better (see Section 4.4.2), the general consensus is that a linear kernel SVM and logistic regression perform comparably [10]. The small difference in accuracy may be due to features specific to our dataset or differences in how well the parameters are tuned for the models.

The general pattern that we saw for the regularization parameter C is that increasing C up to a certain value results in an increase in accuracy, but once C reaches a certain point, the accuracy either decreases or levels off. For example, changing the regularization parameter C from 0.1 to 1 to 10 increased our accuracy for logistic regression with unigram features, but changing C to 100 decreases the accuracy, as shown in Table 4. This is because higher values of C up to a certain point improves the performance of the model by positioning the hyperplane more optimally, but past that point, the hyperplane's margin becomes too small, resulting in overfitting to the training data. For our most optimal model, we see that $C = 10$ is that optimal value; anything on an order of magnitude higher results in overfitting, and anything below results in a sub-optimal positioning of the hyperplane.

For regularization types, we saw that L2 (which is the default used by sklearn) performed better than L1. With L1 regularization, we minimize the sum of the absolute differences between the target value and the estimated value:

$$L_1 = \sum_{i=1}^{n} |y_i - f(x_i)|. \tag{3}$$

With L2 regularization, we minimize the square of the differences between the target value and the estimated value:

$$L_2 = \sum_{i=1}^{n} (y_i - f(x_i))^2. \tag{4}$$

Thus, we can see that L_2 penalizes errors more heavily that L_1, taking outliers more into account.

6. Error Analysis

6.1. Corpus Selection

One issue in creating our data set is that we were constrained by the texts that were available for free in a digital format online. This means that many of our datapoints had 100-word texts that were from the same author and/or translator. In addition to the signatures that a source language may leave on its translated text, a particular author or translator may have their own "signatures": the frequent use of a certain word, the tendancy to use proper nouns or pronouns over other parts of speech, etc. [6]

Ideally, each of the 100-word sequences that comprise an input datapoint would have come from a unique text with a unique author and translator. This would reduce the amount of noise in our data and ensure that any differences in word frequencies are truly the result of the source language.

6.2. Feature Extraction

When extracting features, we used a uni-gram and bi-gram model with our 100-word datapoints. Again, we were

still constrained by the amount of professionally translated text in various source languages found online (for example, it was very easy to find original-English language short stories online, but it was more difficult to find Russian or Korean-translated texts).

Because each input datapoint contains only 100 words, it is very likely that bi-gram (and any higher n-gram) features are too complex, i.e. too specific to each datapoint, because each datapoint is too short to contain enough information. This is why our bi-gram models performed worse than our uni-gram models. However, if we were to increase the number of words per datapoint to greater than 100, we may not have enough datapoints to train the model effectively. This is why we settled on 100 words, but if we were to collect more data for each language, we could potentially increase the number of words per datapoint and see better results for bigram and higher-level n-gram models.

6.3. Parameter Tuning

Due to time constraints and the amount of Google Cloud credits available, we were not able to run some of the more complex models when parameter tuning multiple times (for example, SVM with bi-gram features, which takes around 5 hours per model to train, and any RNN model, which takes around 12 hours to train).

Ideally, we would run a full grid-search on all the parameters we wished to tune for both uni-gram and bi-gram features, and we would potentially tune more parameters for SVM (most notably gamma, which is a measure of the influence of a single training point). Because our constraints, we were unable to parameter tune gamma and kept it as the sklearn default value for all of our models, and we trained our bi-gram model with the regularization constant and kernel set to the most optimal values found during our unigram SVM grid search (which were C = 1 and kernel = linear). We also would have tried more hyperparameter values, which would result in our models being better optimized.

6.4. RNN: Vanishing Gradient Problem

We believe that the RNN was only able to achieve 18% accuracy due to vanishing gradients: when we printed out the gradient at each time step, we saw that the weights were not changing (approximately [-1.7546, -1.5328, -1.9873, -1.7684, -1.9384, -1.8358] each time).

In the future we could consider moving from a RNN to LSTM. Using LSTM, a particular type of an RNN, would allow us to better retain information from the beginning of the sequence and would help us resolve the gradient vanishing problem that the vanilla RNN currently faces[1]. For an RNN, we are using the same weights matrices for every module. Thus, when we backpropogate we calculate the loss with respect to the weights using the chain rule, and if the partial derivatives are small (less than 1), then we are multiplying many small numbers together, which causes the gradient to become approximately. If the gradient is approximately 0, the weights do not update. If the partial derivatives are large, then we are multiplying many larger numbers together, which would cause us to have an exploding gradient and have an unstable model.

See Figure 5 and Figure 6 for a visual comparison of an RNN and an LSTM. As shown below, the LSTM contains a cell-state, represented by the horizontal line running through the top of the diagram. Each LSTM module then consists of 4 gates.
The first gate is the forget gate, which decides whether or not we should erase the cell.

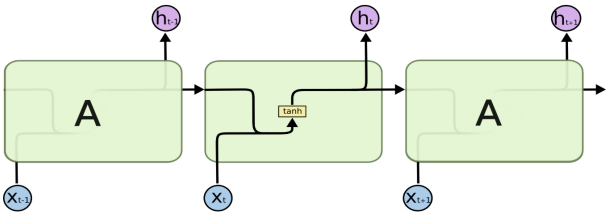

Figure 5. *A recurrent neural network (RNN) with a single-layer repeating module [3].*

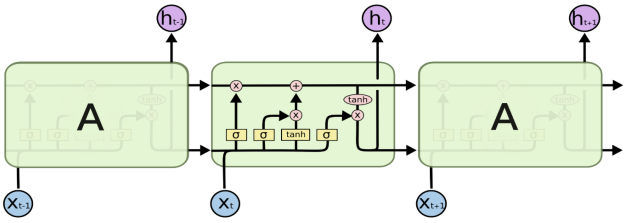

Figure 6. *A long short term memory network (LSTM) containing four interacting layers [3].*

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f). \tag{5}$$

The second gate is the input gate, which decides whether or not we want to write to the cell.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i). \tag{6}$$

The third gate is the gate gate, which decides how much to write to the cell.

$$g_t = \tanh(W_i \cdot [h_{t-1}, x_t] + b_t). \tag{7}$$

The fourth gate is the output gate, which decides what the output will be.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o). \tag{8}$$

We find the next cell state to be

$$c_t = f \bigodot c_{t-1} + i \bigodot g. \tag{9}$$

We then calculate the next hidden state to be

$$h_t = o_t \bigodot tanh(c_t). \tag{10}$$

Thus, each module has a limited interaction with the cell-state, allowing us to better preserve data from the beginning of the sequence and avoid disappearing and exploding gradients.

7. Code Link

We provide a link to our code:
`https://drive.google.com/file/d/1t3J7s3E1Y080lxexIdCzK8nD7-z337xe/view?usp=sharing`

8. Dataset Link

We provide a link to our dataset:
`https://drive.google.com/file/d/1JMFkixFxrxQssn8tidsa--pZUWLvKaea/view?usp=sharing`

References

[1] The atlantic.

[2] Project gutenberg.

[3] Understanding lstm networks.

[4] Korean literature in translation, Jun 1970.

[5] The new york times, Aug 2018.

[6] M. Baroni and S. Bernardini. A New Approach to the Study of Translationese: Machine-learning the Difference between Original and Translated Text. *Literary and Linguistic Computing*, 21(3):259–274, 08 2005.

[7] Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, and C.-J. Lin. Training and testing low-degree polynomial data mappings via linear svm. *J. Mach. Learn. Res.*, 11:1471–1490, Aug. 2010.

[8] O. Craciunescu, C. Gerding-Salas, and S. Stringer-O'Keeffe. Machine translation and computer-assisted translation. *Machine Translation and Computer-Assisted Translation*, 2004.

[9] D. Crystal. *English as a Global Language*. Cambridge University Press, 2003.

[10] G. Drakos. Support vector machine vs logistic regression, Oct 2018.

[11] I. Ilisei, D. Inkpen, G. Corpas Pastor, and R. Mitkov. Identification of translationese: A machine learning approach. volume 6008, pages 503–511, 03 2010.

[12] D. Kurokawa, C. Goutte, and P. Isabelle. Automatic detection of translated text and its impact on machine translation. In *In Proceedings of MT-Summit XII*, pages 81–88, 2009.

[13] G. Lynch and C. Vogel. Towards the automatic detection of the source language of a literary translation. In *Proceedings of COLING 2012: Posters*, pages 775–784, Mumbai, India, Dec. 2012. The COLING 2012 Organizing Committee.

[14] F. Pedregosa. Loss functions for ordinal regression.

[15] V. Volansky, N. Ordan, and S. Wintner. On the features of translationese. *Literary and Linguistic Computing*, 30(1):98–118, 07 2013.