Character Set Encoding Detection using a Support Vector Machine

Sean R. Sullivan
seans@google.com

Abstract

I propose a method to automatically detect the character set encoding for an HTML text document using a Support Vector Machine. Using only the stream of bytes within the document, this algorithm learns the patterns of byte n-gram frequencies to classify the document. The algorithm will currently detect the following popular character set encodings: US-ASCII, UTF-8, UTF-16, ISO-8859-1, ISO-8859-2, ISO-8859-5, Big5, GB2312, GB18030, Shift-JIS, EUC-JP, EUC-KR, KOI8-R.

1 Introduction (The Problem)

Have you ever received a web page containing strange question mark characters, or other garbled text? Unable to decode bytes into text characters, the browser substitutes these question marks. Or believing erroneously it can decode the bytes into text, it outputs gibberish instead. Without knowledge of the scheme used to encode text into bytes, the character set encoding, it is not possible to make sense of a stream of bytes. These encodings come in several different flavors: single-byte, multi-byte, and variable-length. Additionally, some encodings are proper supersets of other encodings (e.g. UTF-8 is a superset of US-ASCII).

Table 1: Character Set Encodings

Encoding	Common Languages Encoded	Num Bytes
US-ASCII	English	1
UTF-8	All	1 - 4
ISO-8859-1 (latin 1)	Spanish, French, German	1
GB18030	Chinese, Japanese, Korean	1, 2, or 4
GB2312	Simplified Chinese	2
Shift-JIS	Japanese	1 - 2

The web's Hypertext Transfer Protocol (HTTP) [1] specifies the character set encoding for a web page should be declared in the `Content-Type` response header. Example:

```
Content-Type:  text/html; charset=ISO-8859-1
```

But if this `Content-Type` header is wrong or missing, how can we decode the web page? Is it possible to detect the character set encoding of a text document without this metadata? Using features generated from the raw bytes of HTML text documents, this experiment attempts to use a Support Vector Machine (SVM) to determine the document's character set encoding.

1.1 Digresion: Unicode

After the initial proliferation of incompatible character set encodings, the Unicode consortium was created to unify the representation of all languages. Toward that end, the consortium has attempted to define a standard for the characters of all the worlds languages. Separately, it has defined methods to encode these characters. Several of these encodings include UTF-8, UTF-16, and UTF-32. Because the variable-length UTF-8 encoding is compatible with US-ASCII, and because it is a compact representation of some the web's most common languages, it is fast becoming the current standard for encoding text documents.

2 Features (Byte N-gram Frequency)

Since almost all character set encodings require between one to four sequential bytes to encode a character, I define a **byte n-gram** as a sequential concatenation of *n* bytes. For example, a byte 3-gram (trigram) is the concatenation of three adjacent bytes within a document, while a 1-gram (unigram) is simply a single byte. Consequently, there are 2^{8n} possible values for each type of n-gram. These n-grams are generated by scanning the document sequentially, shifting the n-gram window by one byte at each iteration. The frequency (by percentage) within the document of each of the n-gram values defines a feature. For example, a document encoded in US-ASCII could have a frequency of 0.0137 for the unigram value 100 (which is character 'd' in this encoding), while a Chinese language document encoded in GB18030 could have a significantly different frequency for this unigram value. If we include all n-grams up to 4, the feature space size is:

$$2^8 + 2^{16} + 2^{24} + 2^{32} = 4{,}311{,}810{,}304$$

2.1 Feature Selection

Since this feature space is so large, this experiment only calculates features for unigrams, bigrams, and trigrams. But since even this space is too large (approximately 16 million), we need a method to choose good features from this large set. In order to find features which discriminate well between the positive and negative class, this experiment uses the Kullback-Leibler (KL) divergence:

$$KL(p\|q) = \int_x p(x) \log \frac{p(x)}{q(x)}$$

Assuming the distribution of the byte n-gram frequency is Gaussian for each encoding, this experiment calculates the frequency mean and variance for the set of positive and negative examples for each feature. Since the KL divergence is an asymmetric measure of the distance between two probability distributions, we can measure how much a feature differs between the positive and negative examples. A high KL divergence indicates a feature which discriminates well between the positive and negative examples, because it indicates the positive label has high probability and the negative label has low probability. In order to have some confidence in the Gaussian parameter estimation, this experiment only includes a feature if a minimum threshold of number of frequencies was present for both the positive and negative examples. Not all features correspond to a character in the positive label encoding, but the following table illustrates examples when the chosen feature is a character.

Table 2: Feature KL Divergence Examples

Encoding	Language	Byte Value	Num Bytes	KL Divergence
UTF-8	Simplified Chinese	35280	2	210.97
UTF-8	Korean	47569	2	179.39
UTF-8	Traditional Chinese	40372	2	127.29
ISO-8859-5	Russian	213	1	10.91
ISO-8859-1	Portuguese	195	1	2.71

3 Experiments

3.1 Data Set

In order to generate HTML documents with different character set encodings, a set of web pages from Wikipedia [2] was downloaded for some of the world's most common languages. These web pages were all encoded in UTF-8. These pages were then converted into encodings common to these languages, using the CharsetEncoder [3] class included in the *Standard Edition 6* of the Java Development Kit. Additionally, every example was encoded into UTF-16:

Table 3: Languages and their Character Set Encodings

Language	Character Set Encodings
English	US-ASCII
Spanish	ISO-8859-1
French	ISO-8859-1
German	ISO-8859-1
Italian	ISO-8859-1
Portuguese	ISO-8859-1
Polish	ISO-8859-2
Czech	ISO-8859-2
Hungarian	ISO-8859-2
Russian	ISO-8859-5, KOI8-R
Chinese	Big5, GB2312, GB18030
Japanese	Shift-JIS, EUC-JP
Korean	EUC-KR

3.2 Support Vector Machine

This experiment used Thorsten Joachim's `SVM-light` software [4] which implements a Support Vector Machine. The software defines four built-in kernels: linear, polynomial, radial basis function, and sigmoid hyperbolic tangent. Having found no difference between the standard kernels, the linear kernel was used in all of the experiments.

3.3 The Procedure

For each character set encoding, this experiment generated a random set of HTML documents, where approximately half were drawn from this encoding as the positive examples. The negative examples were drawn from the remaining encodings. This generated set was divided into a training set (70%) and a test (30%) set. Once the training set was created, this experiment ran feature selection on this training set to determine the salient features. After generating the features for the training set, the support vector machine was run against the training set to learn a model:

```
svm_learn -x 1 train-UTF-8.dat model-UTF-8
```

The `-x 1` flag calculates the *leave-one-out-cross-validation* (LOOCV) error. Once the model was created, it was run against the features generated from the test set.

```
svm_classify test-UTF-8.dat model-UTF-8 results-UTF-8.txt
```

3.4 Classifier Comparison

As a final check, we compared the accuracy of the SVM classifier against `jchardet` [5], the java port of Mozilla's charset detector. For three popular encodings, we generated another random set of 1000 documents, and asked the `jchardet` classifier to identify the character set encoding. Using our previously generated SVM model, we attempted to classify these same examples. Since the `jchardet` software returns multiple labels if it is unsure, this comparison is inexact. Whenever `jchardet` returned multiple labels, we considered this a misclassification if the correct label was not among the returned labels.

4 Results

For classifying some encodings, the SVM classifier appears to exceed the accuracy of the `jchardet` classifier. Since the LOOCV error appears similiar to the test set error, it appears we are not overfitting the data. The number of support vectors for each model gives us an idea of how

Table 4: SVM with linear kernel

Positive Label	Num Training	Num Test	Num Support Vectors	LOOCV	Accuracy
UTF-8	2231	1275	575	2.15%	96.16%
UTF-16	2285	1302	129	0.09%	99.92%
US-ASCII	2457	1134	396	1.95%	98.41%
ISO-8859-1	2489	1425	644	1.77%	97.54%
ISO-8859-2	2622	1415	264	0.31%	99.65%
ISO-8859-5	2456	1110	62	0.16%	99.82%
KOI8-R	2418	1157	65	0.04%	99.91%
GB2312	2411	1147	170	0.12%	99.30%
GB18030	2419	1140	425	2.27%	98.16%
Big 5	2427	1141	144	0.16%	100.00%
Shift-JIS	2481	1084	113	0.08%	100.00%
EUC-JP	2474	1111	135	0.04%	99.91%
EUC-KR	2444	1158	112	0.04%	99.83%

Table 5: SVM Classifier Compared to Mozilla jchardet

Encoding	SVM Classifier	jchardet Classifier
UTF-8	94.2%	100.0%
ISO-8859-1	99.5%	83.6%
GB18030	99.0%	72.5%

easily the SVM found a separating hyperplane. Unsurprisingly, the models with the fewest support vectors produced the best accuracy.

5 Conclusions and Future Directions

While this experiment produced relatively good classification accuracy, there are several improvements that can be made. The current one-versus-all classification will not scale well. As examples from more encodings in different languages are added to the negative examples, it will become harder to find a separating hyperplane. A better approach might be similiar to a decision tree. An initial classification could determine the number of bytes for the encoding (single, double, variable), which would subsequently provide a more accurate way to generate the byte n-grams.

The feature selection can be improved on several fronts. We could incorporate the confidence interval for the Gaussian parameters in our KL divergence calculation. We could filter out KL divergence measures that do not meet a confidence threshold. Additionally, we can incorporate features which measure the difference in frequency readings between the positive and negative examples. For example, if there are numerous frequencies for the unigram value 30 within ISO-8859-1 encoded documents, but none for all other encodings, then we should use this disparity to identify ISO-8859-1 encoded documents. Currently, we are only incorporating features for which we can estimate Gaussian parameters for both positive and negative examples. Since we are trying to reduce the number of features by selecting those which discriminate the best, it might be worthwhile to explore PCA. Finally, we can incorporate knowledge of specific encodings into the classification, by adding features particular to that encoding. For example, we could add a feature which defined the percentage of n-grams values with the high bit set. Since US-ASCII is a 7-bit encoding, none of the bytes would have the high bit set.

References

[1] R. Fielding, et al., Hypertext Transfer Protocol–HTTP/1.1, RFC 2616, 1999. ⟨http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html⟩

[2] http://en.wikipedia.org

[3] http://java.sun.com/javase/6/docs/api/java/nio/charset/CharsetEncoder.html

[4] T. Joachims, 11 in: Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning, B. Scholkopf and C. Burges and A. Smola (ed.), MIT Press, 1999.

[5] http://sourceforge.net/projects/jchardet