Visualising Top Features in Linear SVM with Scikit Learn and Matplotlib

Aneesha Bakharia Follow
Feb 1, 2016 · 3 min read

A classifier need not be a black-box! I know we are often dealing with very high dimensional data but particularly with a linear Support Vector Machine, it is possible to take a peek at the top features. This blog post is not about using feature selection or ranking using information gain metrics—I'll cover that in a follow-up blog post. In this post I am going to cover how to visualise the top feature coefficients after an SVM model has been created in Scikit Learn. I have found the technique to be very useful especially when dealing with binary text classification.

> *A linear SVM model is not a complete Black Box!*

I'll start by sharing a good tip for working with Scikit Learn. Don't just stick with the known public API—you need to explore the _methods and attributes_. After applying a CountVectorizer you can for example get a vocabulary count using .vocabulary_.

Once a linear SVM is fit to data (e.g., svm.fit(features, labels)), the coefficients can be accessed with svm.coef_. Recall that a linear SVM creates a hyperplane that uses support vectors to maximise the distance between the two classes. The weights obtained from svm.coef_ represent the vector coordinates which are orthogonal to the hyperplane and their direction indicates the predicted class. The absolute size of the coefficients in relation to each other can then be used to determine feature importance for the data separation task.

I'll share a method that takes the fitted linear SVM model, feature names (obtained with .get_feature_names()) and the number of top features to use as parameters and produces a bar chart using Matplotlib.

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.svm import LinearSVC
import matplotlib.pyplot as plt
```

```
def plot_coefficients(classifier, feature_names,
top_features=20):
 coef = classifier.coef_.ravel()
 top_positive_coefficients = np.argsort(coef)[-
top_features:]
 top_negative_coefficients = np.argsort(coef)[:top_features]
 top_coefficients = np.hstack([top_negative_coefficients,
top_positive_coefficients])
 # create plot
 plt.figure(figsize=(15, 5))
 colors = [‘red’ if c < 0 else ‘blue’ for c in
coef[top_coefficients]]
 plt.bar(np.arange(2 * top_features),
coef[top_coefficients], color=colors)
 feature_names = np.array(feature_names)
 plt.xticks(np.arange(1, 1 + 2 * top_features),
feature_names[top_coefficients], rotation=60, ha=’right’)
 plt.show()
```

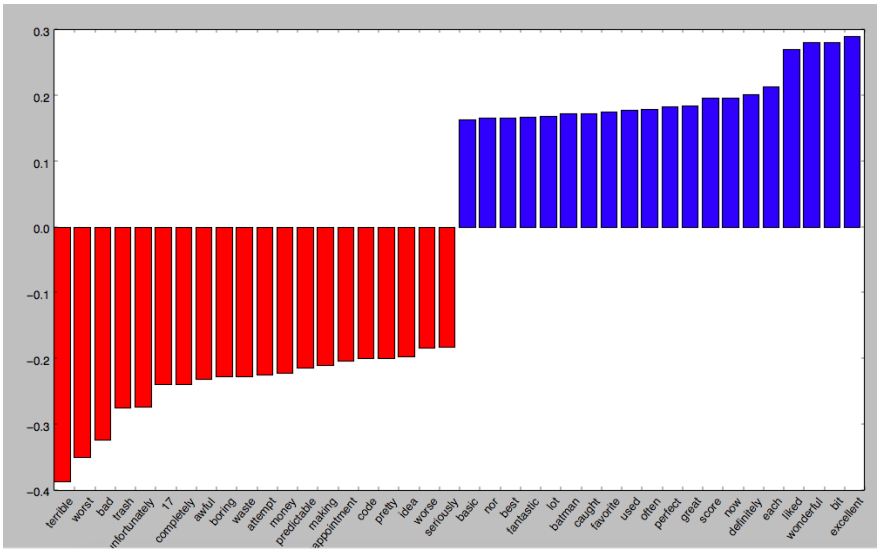
```
cv = CountVectorizer()
cv.fit(data)
print len(cv.vocabulary_)
print cv.get_feature_names()
```

```
X_train = cv.transform(data)

svm = LinearSVC()
svm.fit(X_train, target)


plot_coefficients(svm, cv.get_feature_names())
```

As you can see below, the plot provides useful insight into what features (words) are being used by the model to make the positive and negative classifications from a sentiment dataset. Some features seem reasonable while others such as ‘17’ and ‘bit’ are not. In any case visualising the top feature coefficients makes the mystery behind what the linear SVM classifier has learnt more transparent.



An example plot of the top SVM coefficients plot from a small sentiment dataset.

I'll conclude with a link to a good paper on SVM feature selection. In the paper the square of the coefficients are used as a ranking metric for deciding the relevance of a particular feature. In a future blog post I'll cover feature ranking and selection using Information Gain.