

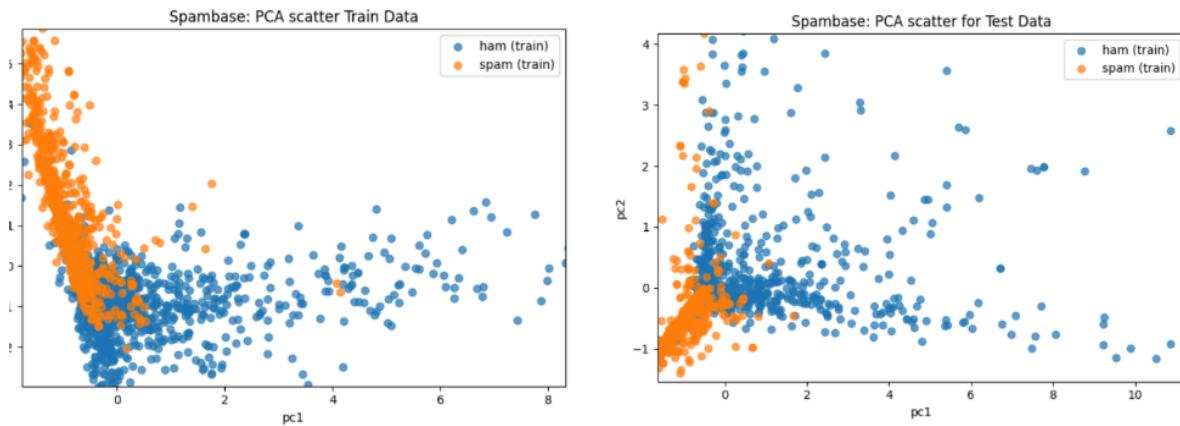
Using Linear Models For Language Identification and Spam Detection

This report examines linear classification models; Perceptron, Logistic Regression, and Support Vector Machines (SVM) and how they're applied to two binary classification problems: spam detection using the UCI Spambase dataset and language identification between English and Dutch texts. The goal is to evaluate model performance and explore the effects of feature engineering on accuracy when doing classification.

Spam Detection

The features for the Spambase were scaled using the Standard Scaler which essentially lowers the feature's variance. The data is modeled through Sklearn's Principal Component Analysis algorithm.

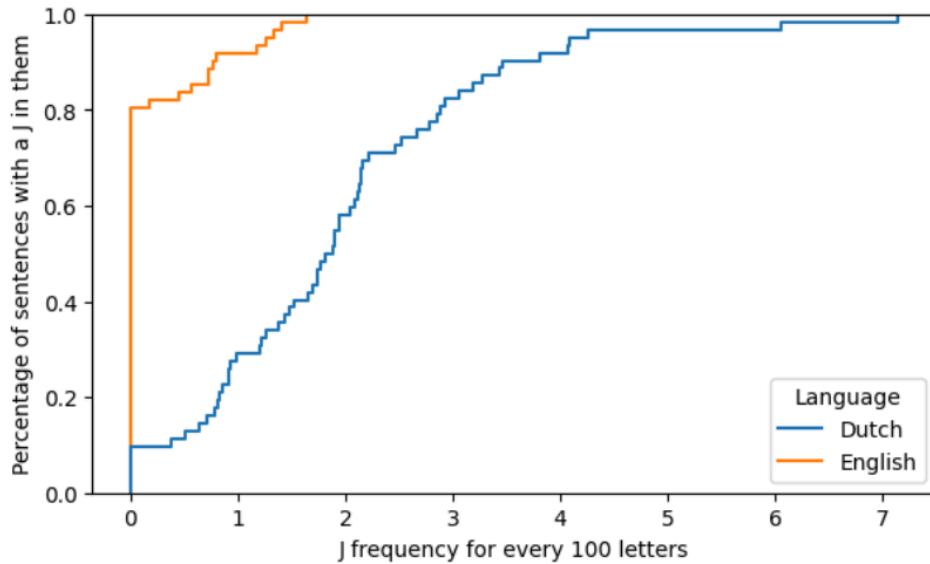
Three linear models were trained on the UC Irvine Spambase dataset. As seen by the scatter plot below the data is not quite linearly separable for both the training and testing data. Given that there isn't a clear separation between the two categories, the perceptron converges on a poor hyperplane. The PCA plots show that in both train and test the data forms essentially a wedge. Granted that a straight line cannot separate them well which is why the perceptron had an accuracy score of 39%. Meanwhile the SVM and the Logistic Regression's scored 91.8% and 91.6%; as their boundary curved to follow the shape of the data. Additionally, The dataset was split into 80% training, 10% development, and 10% testing sets using the Scikit Learn Train Test Split function. All models were trained using the UCI training set.



The Logistic Regression and SVM both used regularization, which resulted in them generalizing the non linearly separable data well. As shown in class, the SVM in particular, maximizes the margin between classes which is why it could model this wedge shaped data better than the Perceptron. While the Perceptron is suitable for linearly separable data, its performance on the Spambase dataset was quite poor. The linear classifiers like the SVM and Logistic Regression outperformed it significantly. This makes them better choices for spam detection tasks with overlapping features.

Language Identification

The data given are several text files in English and in Dutch. Upon reviewing the files, the most apparent pattern was the frequency of different characters. The frequency of the letter ‘J’ was significantly higher in Dutch in comparison to English. This can also be observed on the charts below that compare the frequency of the letter J in a sentence in English in comparison to Dutch. The data was split by sentences and was “vectorized” by counting the frequency of each letter in a sentence.



The SVM’s performance is near perfect as English and Dutch are linearly separable in this 26 letter feature space. As discussed in lecture, the performance of the Perceptron was relatively (with an accuracy score of 78%) as it does not find the ideal hyperplane, it converges to any hyperplane which results in its accuracy decreasing. The Logistic Regression had a low recall when classifying English which indicates that the model is biased towards predicting Dutch. This shows that an unbiased model is not directly correlated to balanced data.

SVM:	precision	recall	f1-score	Logistic Regression:	precision	recall	f1-score
English(1)	0.972222	1.000000	0.985915	English(1)	0.857	0.571	0.686
Dutch(-1)	1.000000	0.964286	0.981818	Dutch(-1)	0.679	0.905	0.776
Accuracy: 0.9841				Accuracy: 0.8095			
Perceptron:	precision	recall	f1-score				
English(1)	0.952	0.714	0.816				
Dutch(-1)	0.714	0.952	0.816				
Accuracy: 0.7857							

Given these results, the ideal model to use for this is the Support Vector Machine, as it has the highest overall scores when compared to the Perceptron and the Logistic Regression. Though the data was quite balanced, the Logistic Regression seemed to have a bias for selecting Dutch, which is why the f1-score could actually be used as an indicator for how well the model is performing. The SVM does well here because the two languages are almost linearly separable in the 26 letter

frequency space, and as discussed in class, the algorithm's ability to find the maximum margin hyperplane results in high accuracy.

While the letter frequency based feature set worked relatively well in this experiment, it is not the most effective method for language identification. Word features like words that only exist in English and in Dutch would have helped create more separable data. Had this been realized earlier this certainly would have been implemented.

The Ideal Model

In both classification tasks, the SVM outperformed the two other linear classifiers, particularly in cases where data was linearly separable or essentially nearly separable. The Perceptron struggled with the non linearly separable data, while logistic regression showed biases depending on the feature set. Feature engineering had a significant impact on performance, especially in the language identification task. Had the model's performance been to be analyzed earlier, creating features for words that only exist in English and in Dutch would certainly have been implemented.

Code Explanation

The code for the three classifiers is essentially identical. The Support Vector Machine classifier first generates feature label pairs (either for English or Dutch or from the UCI Spambase), then concatenates and splits them into training and test sets. The model is trained on the training set and then used to predict the test set labels. Finally, it calculates and prints precision, recall, and F1 scores for both classes (English = 1, Dutch = -1), as well as the overall test accuracy.

The Logistic Regression classifier first generates feature label pairs, then splits the data into training and test sets. The data is standardized using Scikit Learn's StandardScaler such that all features contribute equally to the model when it is training. Logistic Regression is then initialized (from the tutorial linked above the code). The model is trained on the scaled training data and evaluated on the scaled test data using the .score() method to calculate accuracy.

The Perceptron class iteratively updates its weight and bias as it iterates over the data. The dataset is split into training and test sets. After training, the classify() function predicts by dotting the inputs with the weights plus the bias. Runtests() evaluates the model's accuracy.