Text Analytics & Business Application

Text Classification 1

Qinglai He

Department of Operations and Information Management

Wisconsin School of Business

WISCONSIN SCHOOL OF BUSINESS

IC3 Discussions – Pros and Cons of Using More Grams

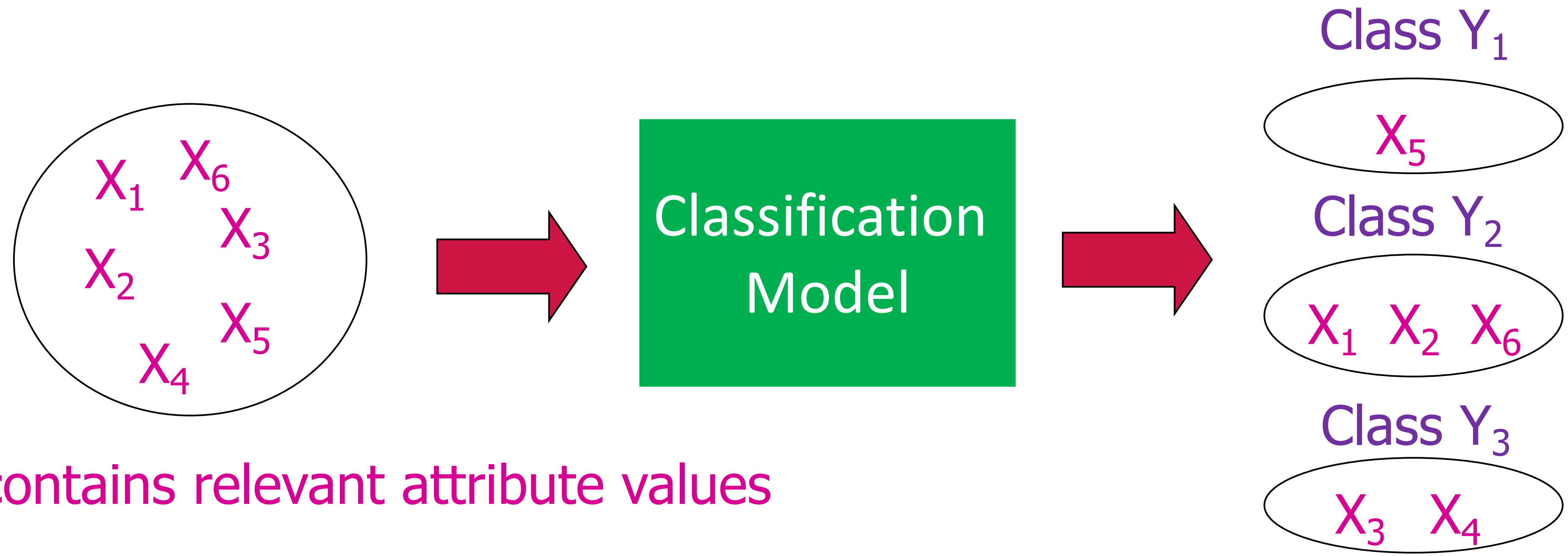
- Some methods to improve the TF-IDF output: remove more uninformative words, perform stemming or lemmatization to normalize tokens ...
- Pros and cons of using more grams
  - **Pros**: capture more contextual information
  - **Cons**:
    - May capture some meaningless combinations of terms, bring more noise to the representation
    - Increase the dimensionality of the representation vectors (features),
    - Lead to more sparse representation vectors
    - High-dimensional and sparse representations lead to inefficient storage and may further hurt downstream tasks (e.g., classification) performance.

Outline of Today's Class

- Hand-coded rules / rule-based classifier
- One pipeline, many classifiers
  - Naïve bayes
  - Logistic regression
  - Support vector machine

Recap Classification

What is Classification?



X_i: contains relevant attribute values

Classes Y_1, Y_2, and Y_3 are pre-determined

Text Classification

- In ML, **classification** is the problem of categorizing a data instance into one or more known classes.
- **Text classification** is a special instance of the classification problem.
  - **Input**: text
  - **Goal**: categorize the piece of text into one or more buckets (called a class) from a set of pre-defined buckets (classes).
  - The "text" can be of arbitrary length: a character, a word, a sentence, a paragraph, or a full document.

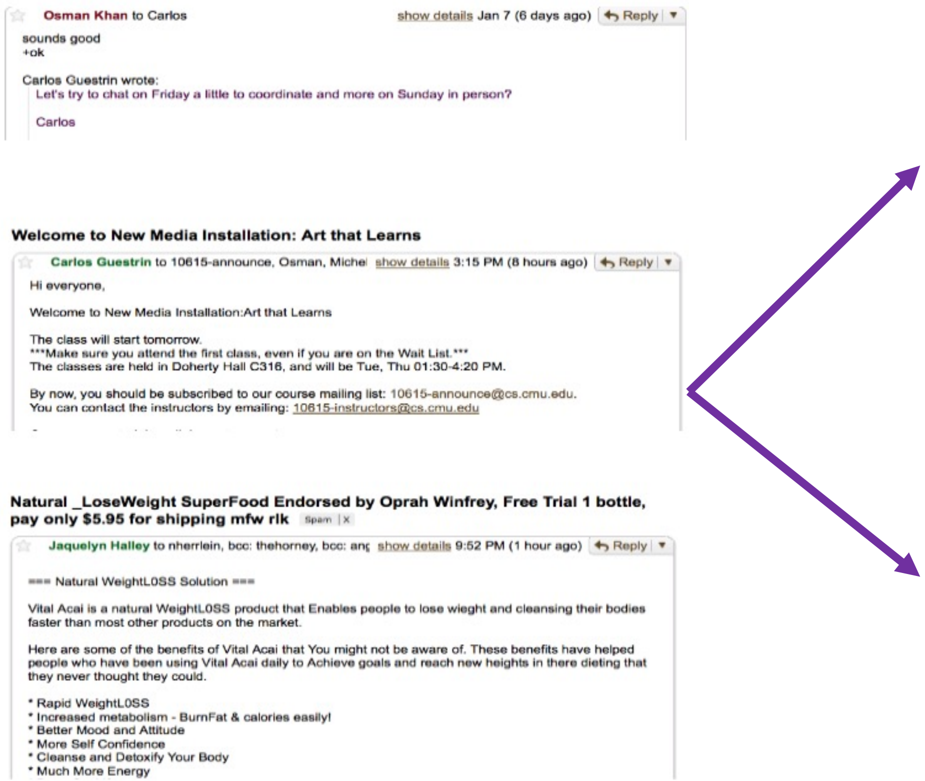
Three Types of Classification

Any supervised classification approach can be further distinguished into three types based on the number of categories involved:

- If the number of classes is two, it's called **binary classification**.
- If the number of classes is more than two, it's referred to as **multiclass classification**.
- In **multilabel classification**, a document can have one or more labels/classes attached to it.

Binary Classifier – Spam Filter

- Example: an email is either spam or not spam.



Not Spam

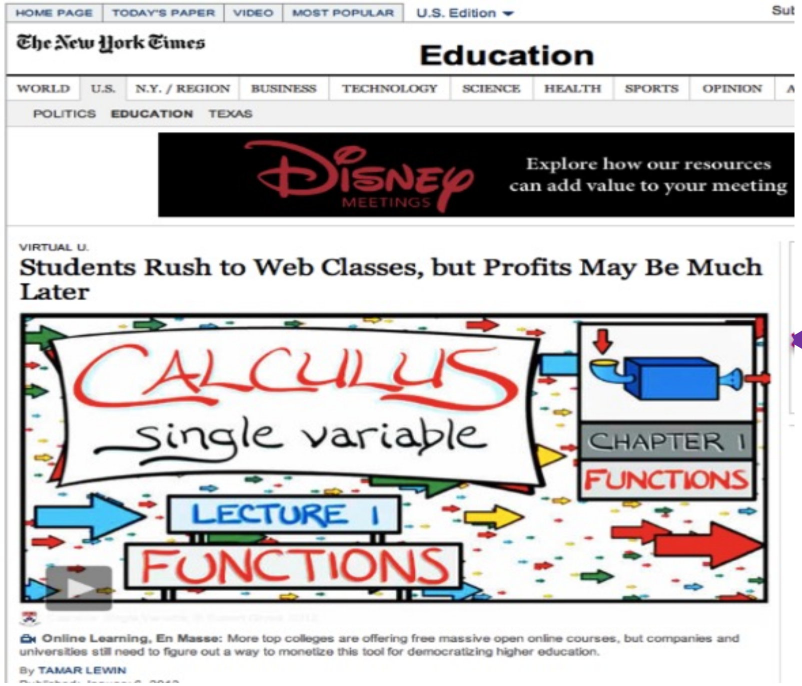
Spam

Input: x
Text of email, sender, IP, …

Output: y

Multiclass Classifier – News Article Tags

- Example: a news article is labeled as education, finance, **or** technology-related.



Input: x
Webpage

Education

Finance

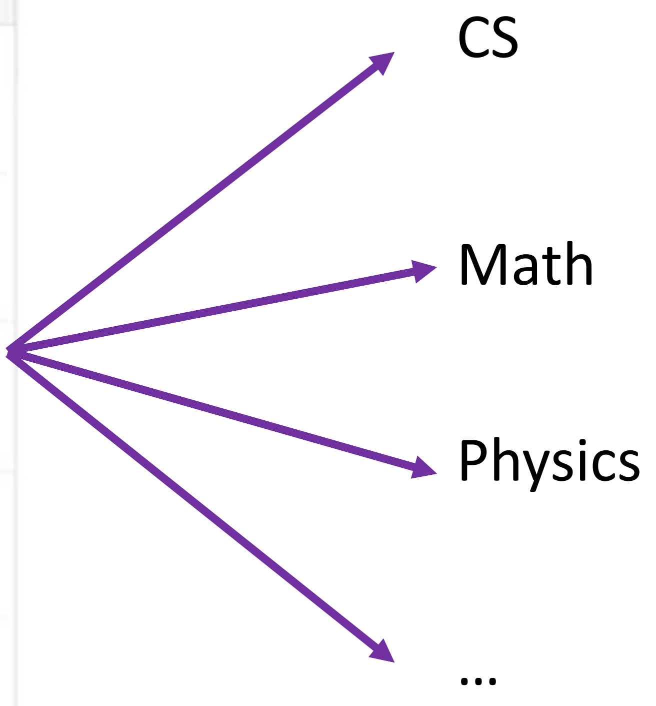
Technology

Output: y
Article
categories

Multilabel Classifier – Paper Classification

- Example: a paper can be CS, math, **and** physics-related.

	ID	TITLE	ABSTRACT	Computer Science	Physics	Mathematics	Statistics	Quantitative Biology	Quantitative Finance
0	1	Reconstructing Subject-Specific Effect Maps	Predictive models allow subject-specific inf...	1	0	0	0	0	0
1	2	Rotation Invariance Neural Network	Rotation invariance and translation invarian...	1	0	0	0	0	0
2	3	Spherical polyharmonics and Poisson kernels fo...	We introduce and develop the notion of spher...	0	0	1	0	0	0
3	4	A finite element approximation for the stochas...	The stochastic Landau--Lifshitz--Gilbert (LL...	0	0	1	0	0	0
4	5	Comparative study of Discrete Wavelet Transfor...	Fourier-transform infra-red (FTIR) spectra o...	1	0	0	1	0	0



A Pipeline for Building Text Classification Systems

1. Collect or create a labeled dataset suitable for the task.
2. Transform raw text into feature vectors (i.e., text representation).
3. Split the dataset into two (training and test) or three parts(training, validation and test sets) then decide on evaluation metric(s).
4. Train a classifier using the feature vectors and the corresponding labels from the training set.
5. Using the evaluation metric(s), benchmark the model performance on the test set.
6. Deploy the model to serve the real-world use case and monitor its performance.

Approach 1: Hand-coded rules / rule-based classifier

Example of Hand-coded rules/rule-based Classifier

- Scenario: we're given a corpus of tweets where each tweet is labeled with its corresponding sentiment: negative or positive.
  - "The new James Bond movie is great!" is clearly expressing a <u>positive</u> sentiment,
  - "I would never visit this restaurant again, horrible place!!" has a <u>negative</u> sentiment.
- How to build a classification system that will predict the sentiment of an unseen tweet using only the text of the tweet?
- Solution:
  - Create lists of positive and negative words in English—i.e., words that have a positive or negative sentiment.
  - Compare the usage of positive versus negative words in the input tweet
  - Make a prediction based on this information

Hand-coded rules/rule-based

- Rules based on combinations of words or other features
  - For example, one rule-based approach could be:
    - Spam detector: using blocklist address AND term “dollars”
- Pro: Accuracy can be high
  - If rules carefully refined by expert
- Con: But building and maintaining these rules is expensive

Approach 2: ML-based Classifiers (1)

Naïve Bayas Classifier

Naive Bayes Classifier

- A probabilistic classifier that uses Bayes' theorem to classify texts based on the evidence seen in training data.
- It estimates the conditional probability of each feature of a given text for each class based on the occurrence.
- Finally, it chooses the class with maximum probability

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$$

Naive Bayes: Coding Examples

- We use a Naive Bayes implementation in scikit-learn.
- Once the dataset is loaded, we split the data into train and test data, as shown in the code snippet below:

```
#Step 1: train-test split
X = our_data.text
#the column text contains textual data to extract features from.
y = our_data.relevance
#this is the column we are learning to predict.
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
#split X and y into training and testing sets. By default,
it splits 75% #training and 25% test. random_state=1 for reproducibility.
```

Naive Bayes: Coding Examples (cont.)

- The next step is to pre-process the texts and then convert them into feature vectors.
- The code snippet below shows this pre-processing and converting the train and test data into feature vectors using CountVectorizer in scikit-learn, which is the implementation of the BoW approach we discussed in Week 3.

```
#Step 2-3: Pre-process and Vectorize train and test data
vect = CountVectorizer(preprocessor=clean)
#clean is a function we defined for pre-processing, seen in the notebook.
X_train_dtm = vect.fit_transform(X_train)

X_test_dtm = vect.transform(X_test)
print(X_train_dtm.shape, X_test_dtm.shape)
```

Naive Bayes: Coding Examples (cont.)

- We now have the data in a format we want: **feature vectors (it is also the text representation)**.
- The next step is to train and evaluate a classifier. The code snippet below shows how to do the training and evaluation of a Naive Bayes classifier with the features we extracted above:

```
nb = MultinomialNB() #instantiate a Multinomial Naive Bayes classifier
nb.fit(X_train_dtm, y_train)#train the mode
y_pred_class = nb.predict(X_test_dtm)#make class predictions for test data
```

Question: How to measure our model performance?

Recap: Performance Evaluation – Confusion Matrix

- Performance measures for classification models can be computed from a confusion matrix. Below is a 2-by-2 confusion matrix.

Actual Class	Predicted Class 1	Predicted Class 0
Class 1	No. of true positives (TP)	No. of false negatives (FN)
Class 0	No. of false positives (FP)	No. of true negatives (TN)

- Assume the success or target class is class 1
- True positive: class 1 classified as class 1
- True negative: class 0 classified as class 0
- False positive (Type I error): class 0 incorrectly classified as class 1
- False negative (Type II error): class 1 incorrectly classified as class 0

Recap: Recall, Precision, and F1 Score

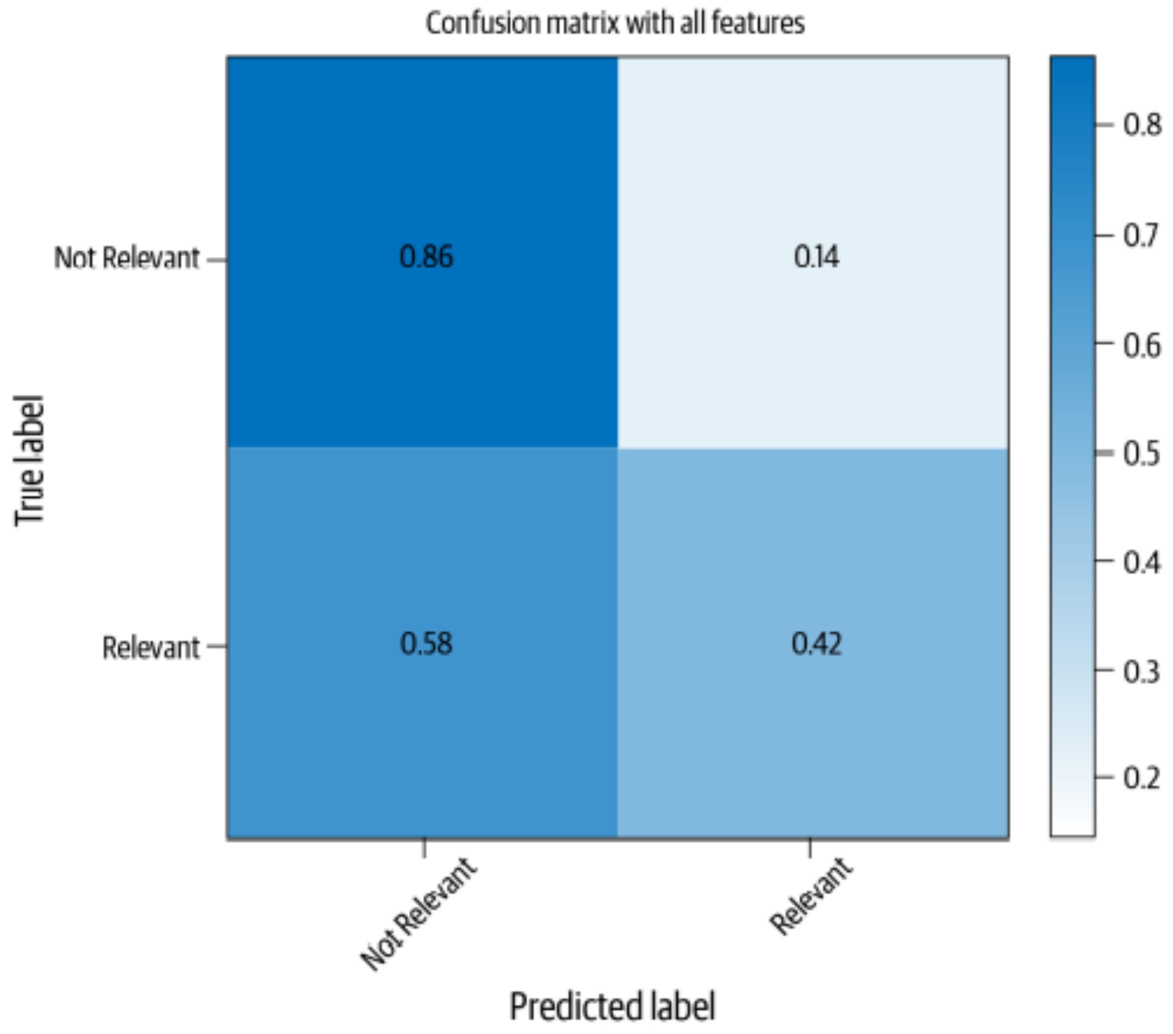
Actual Class	Predicted Class 1	Predicted Class 0
Class 1	No. of true positives (TP)	No. of false negatives (FN)
Class 0	No. of false positives (FP)	No. of true negatives (TN)

- **Misclassification rate**: error rate, proportion of observations that are misclassified, $\frac{FP+FN}{TP+TN+FP+FN}$
- **Accuracy rate**: proportion of observations that are classified correctly, $\frac{TP+TN}{TP+TN+FP+FN}$
- **Recall (sensitivity/True Positive Rate)**: proportion of target class cases that are classified correctly, $\frac{TP}{TP+FN}$
- **Precision**: positive predicted value, proportion of the predicted target classes that belong to the target class, $\frac{TP}{TP+FP}$
- Specificity: proportion of nontarget class cases that are classified correctly, $\frac{TN}{TN+FP}$
- **F1 score**: A combined measure that assesses the Precision/Recall tradeoff is F measure (weighted harmonic mean). People usually use balanced F1.

$$\boxed{\text{F1-Score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}}$$

Naive Bayes: Model Performance

- The classifier is doing fairly well with identifying the non-relevant articles correctly, only making errors 14% of the time.
- However, it does not perform well in comparison to the second category: relevance. The category is identified correctly only 42% of the time



Potential Reasons for Poor Classifier Performance

1. Since we extracted all possible features, we ended up in a **large**, **sparse** feature vector, where most features are too rare and end up being noise. A sparse feature set also makes training hard.
2. There are very few examples of relevant articles (~20%) compared to the non-relevant articles (~80%) in the dataset. This class **imbalance** makes the learning process skewed toward the non-relevant articles category, as there are very few examples of "relevant" articles
3. Perhaps we need a better learning **algorithm**
4. Perhaps we need a better **pre-processing** and **feature extraction** mechanism
5. Perhaps we should look to tuning the classifier's **parameters** and hyperparameters.

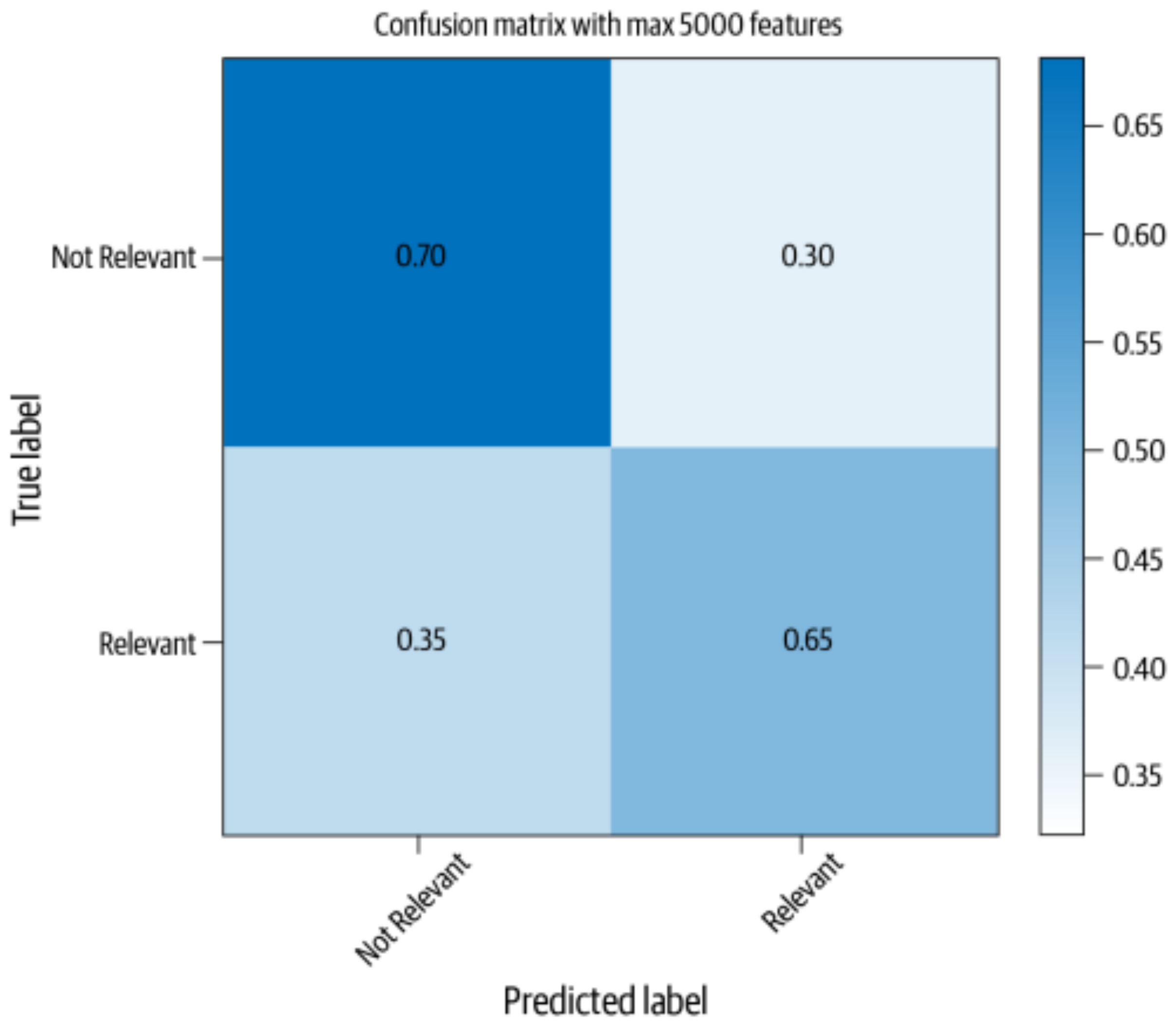
Reason 1 (Large, sparse representation): Solutions

- One way to approach **Reason 1** is to reduce noise in the feature vectors. The approach in the previous code example had close to 40,000 feature.
- Let's see what happens if we restrict this to 5,000 and rerun the training and evaluation process

```
vect = CountVectorizer(preprocessor=clean, max_features=5000) #Step-1
X_train_dtm = vect.fit_transform(X_train)#combined step 2 and 3
X_test_dtm = vect.transform(X_test)
nb = MultinomialNB() #instantiate a Multinomial Naive Bayes model
%time nb.fit(X_train_dtm, y_train)
#train the model(timing it with an IPython "magic command")
y_pred_class = nb.predict(X_test_dtm)
#make class predictions for X_test_dtm
print("Accuracy: ", metrics.accuracy_score(y_test, y_pred_class))
```

Improve Our Naive Bayes Models

- The correct identification of relevant articles increased by over 20%.



Reason 2 (Imbalanced data): Solutions

- Reason 2 in our list was the problem of skew in data toward the majority class.

Solution:

- Manually sampling same data from each class
- Or apply a Python library called “Imbalanced-Learn” that incorporates some of the sampling methods to address this issue
- Random Undersampling
- SMOTE Oversampling

Reason 3 (Better learning algorithm): Solutions

- Reason 3 in our list was that perhaps we need a better learning **algorithm**.

Solution:

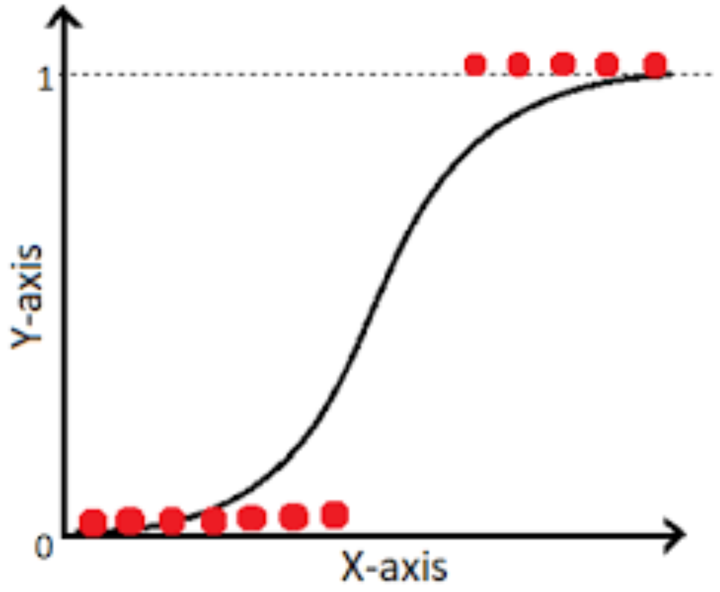
- Alternative ML models
- Deep learning models (RNN, LSTM, etc.)

Approach 2: ML-based Classifiers (2)

Logistics Regression

Logistic Regression

- Logistic regression is an example of a **discriminative classifier** and is commonly used in text classification.
- Logistic regression "learns" the **weights** for individual features based on how important they are to make a classification decision.
- Goal: learn a linear separator between classes in the training data with the aim of maximizing the probability of the data



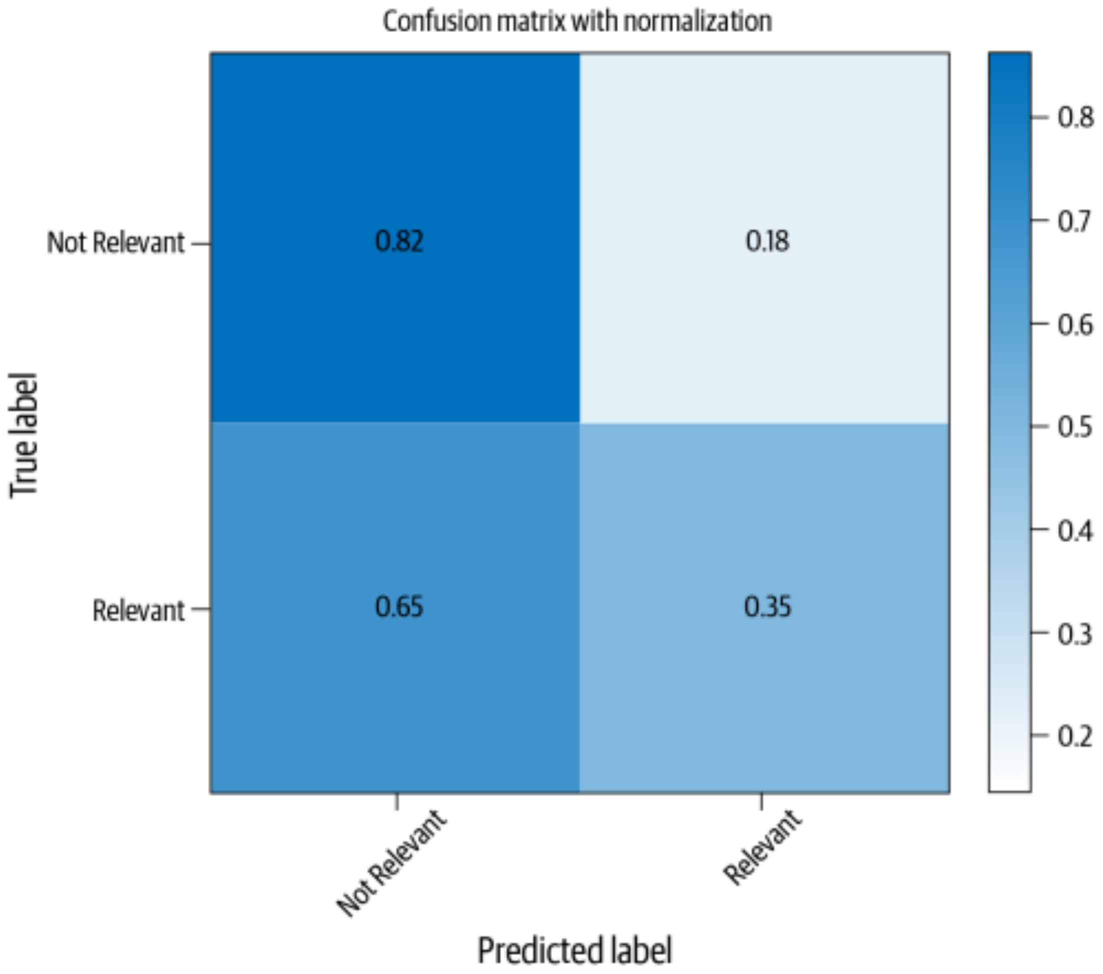
Logistic Regression: Coding Examples

- Let's take the 5,000-dimensional feature vector from the last step of the Naive Bayes example and train a logistic regression classifier instead of Naive Bayes.

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(class_weight="balanced")
logreg.fit(X_train_dtm, y_train)
y_pred_class = logreg.predict(X_test_dtm)
print("Accuracy: ", metrics.accuracy_score(y_test, y_pred_class))
```

Logistic Regression: Model Performance

- This results in a classifier with an accuracy of 73.7%
- The classifier boosted the weights for classes in inverse proportion to the number of samples for that class.
- However, logistic regression seems to perform worse than Naive Bayes for this dataset, because there’s a fall in the bottom-right cell of the confusion matrix.



Reason 3 (Alternative algorithm): Solutions

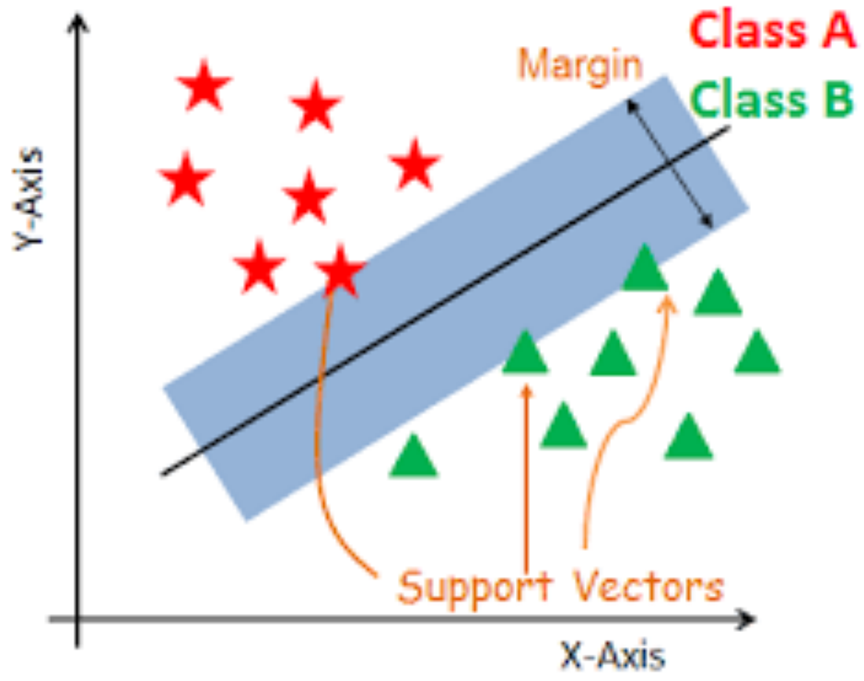
- Reason 3 in our list was: “Perhaps we need a better learning algorithm.” This gives rise to the question: “What is a better learning algorithm?”
- A general rule of thumb when working with ML approaches is that there is <u>no one algorithm that learns well on all datasets</u>.
- A common approach is to experiment with various algorithms and compare them.

Approach 2: ML-based Classifiers (3)

Support Vector Machine

Support Vector Machine

- A support vector machine (SVM) is also a discriminative classifier like logistic regression, but it aims to look for an optimal hyperplane in a higher dimensional space, which can separate the classes in the data by a maximum possible margin.
- SVMs are capable of learning even **non-linear** separations between classes, unlike logistic regression. However, they may also take longer to train.



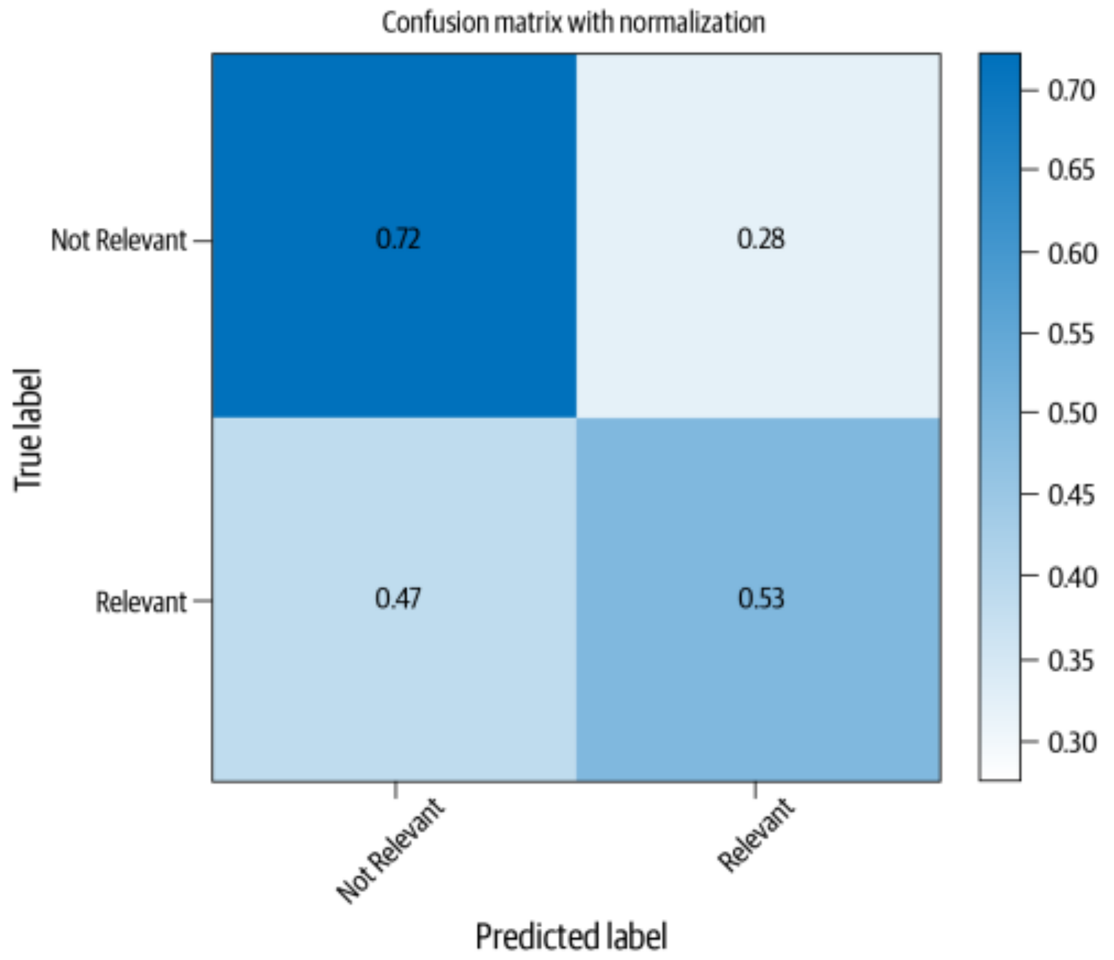
SVMs: Coding Examples

- Let's see how one of them is used by keeping everything else the same and altering maximum features to 1,000 instead of the previous example's 5,000.
- We restrict to 1,000 features, keeping in mind the time an SVM algorithm takes to train.

```
from sklearn.svm import LinearSVC
vect = CountVectorizer(preprocessor=clean, max_features=1000) #Step-1
X_train_dtm = vect.fit_transform(X_train)#combined step 2 and 3
X_test_dtm = vect.transform(X_test)
classifier = LinearSVC(class_weight='balanced') #notice the "balanced" option
classifier.fit(X_train_dtm, y_train) #fit the model with training data
y_pred_class = classifier.predict(X_test_dtm)
print("Accuracy: ", metrics.accuracy_score(y_test, y_pred_class))
```

SVMs: Model Performance

- Compared to logistic regression, SVMs seem to have done better with the relevant articles category.
- Although, among this small set of experiments we did, Naïve Bayes, with the smaller set of features, seems to be the best classifier for this dataset.



Other Popular Classification Algorithms

- K-Nearest Neighbors
- Decision Trees
- Random Forest
- Gradient Boosting

Summary: “One Pipeline, Many Classifiers”

- The three classification methods have different performance in different data.
  - Naive Bayes Classifier
  - Logistic Regression
  - Support Vector Machine
- There is no best classifier.
- Other ways to improve model performance:
  - (Reason 3) Exploring other text classification algorithms (i.e., deep learning model)
  - (Reason 4) Coming up with better pre-processing methods and feature extraction/text representation
  - (Reason 5) Tuning parameters of various classifiers
- Our eventual goal is to build the classifier that best meets our business needs given all the other constraints.

Take 10 minutes break…

Exercises using Google Colab