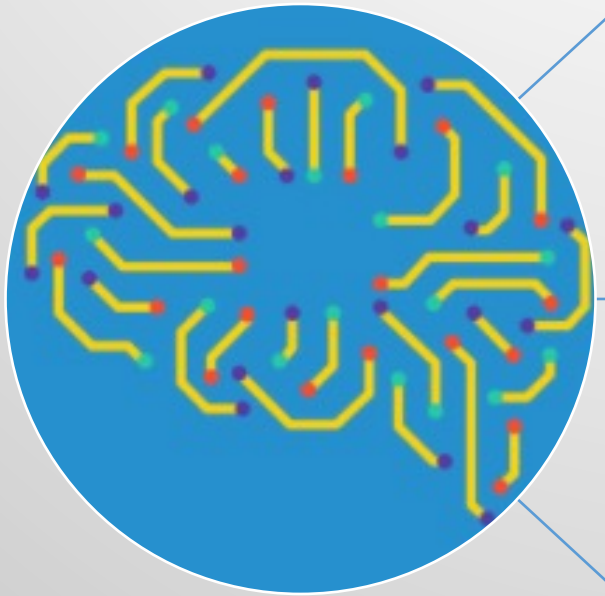
Natural Language Processing

Dr. Karen Mazidi

Part Five:
Machine Learning



Topics

- Naïve Bayes
- Classification metrics

Quizzes

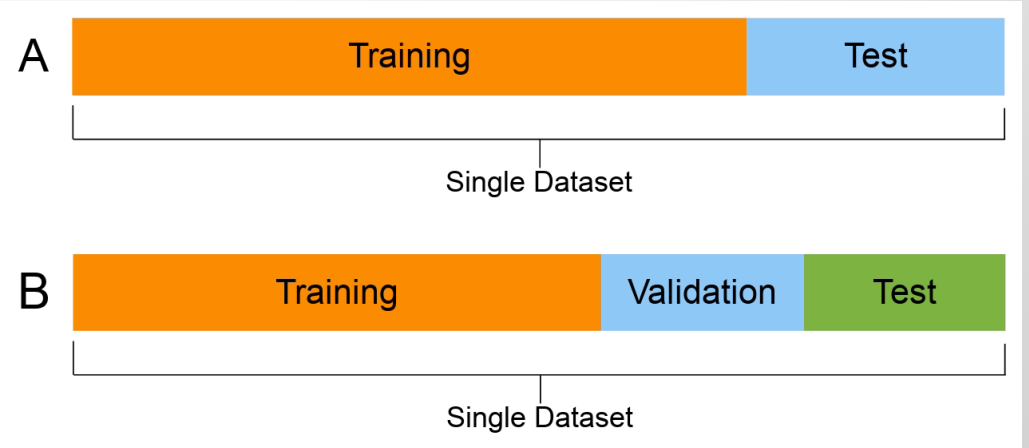
- Q: Naïve Bayes and Logistic Regression

Homework

- Homework: not yet

Workflow for text classification

- Import text data
- Convert to numeric data
- Divide train/test
- Train on training data
- Predict on test data
- Metrics on test data
- If results not great, try another approach



Naïve Bayes

- Performs well on small data sets
- Often used as a baseline for other algorithms
- Suitable for:
  - Binary classification problems: spam/not-spam
  - Multinomial classification problems: positive/negative/neutral sentiment

Bayes' Theorem

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \quad aka: \quad \textbf{posterior} = \frac{\textbf{likelihood} \times \textbf{prior}}{\textbf{marginal}}$$

- in a spam-detection system:

$$P(spam|data) = \frac{P(data|spam)P(spam)}{P(data)}$$

The algorithm

- Naive Bayes makes a simplifying assumption that all predictors are independent:

$$p(X_1, X_2, \ldots X_D|Y) = \prod_{i=1}^{D} p(X_i|Y)$$

- One pass over the data is sufficient to calculate the parameters
- Estimate for class (the prior):

$$MLE_c = \frac{|Y = y_c|}{|N|}$$

The parameters

- MLE for a discrete predictor: $\hat{\theta}_{ic} = \frac{|X_{ic}|}{|N_c|}$
- Smoothing ensures that counts are not 0: $\hat{\theta}_{ic} = \frac{|X_{ic}| + l}{|N_c| + |V|}$
- MLE for a continuous predictor: $\hat{\theta}_{ic} = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

Example: spam detection

- 5574 SMS messages: 0 not spam; 1 spam

```
0    Go until jurong point, crazy.. Available only ...
1                        Ok lar... Joking wif u oni...
2    Free entry in 2 a wkly comp to win FA Cup fina...
3    U dun say so early hor... U c already then say...
4    Nah I don't think he goes to usf, he lives aro...
Name: text, dtype: object
```

Preprocess text

- Set up vectorizer
- Fit vectorizer to train only, then apply to train and test

Code 20.1.1 — Text Preprocessing. Set up Vectorizer

```
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer

stopwords = set(stopwords.words('english'))
vectorizer = TfidfVectorizer(stop_words=stopwords)
```

Code 20.1.2 — Text Preprocessing. Apply Vectorizer

```
# apply tfidf vectorizer
X_train = vectorizer.fit_transform(X_train)  # fit and transform train
X_test = vectorizer.transform(X_test)        # transform only on test
```

Train the model

- Import algorithm
- Create an instance of the algorithm
- Fit the algorithm

Code 20.1.3 — Naive Bayes. Training

```
from sklearn.naive_bayes import MultinomialNB

naive_bayes = MultinomialNB()
naive_bayes.fit(X_train, y_train)

#output
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

Choice of algorithms

- MultinomialNB works well with discrete data such as word counts, but can also handle tfidf data

Default settings were used in the MultinomialNB() algorithm. These settings are:

- alpha: additive (Laplace) smoothing (0 for no smoothing); the default shown in the output above is 1 for add-one smoothing
- class_prior: lets you specify class priors
- fit_prior: if True, learn priors from data; if false, use a uniform prior

Learning

- The NB algorithm learns:
  - the prior probability of spam/not-spam

```
# priors
prior_p = sum(y_train == 1)/len(y_train)
print('prior spam:', prior_p, 'log of prior:', math.log(prior_p))
# output the prior the model learned
naive_bayes.class_log_prior_[1]

# output:
prior spam: 0.13388472473507365 log of prior: -2.01077611244103
-2.0107761124410306
```

Learning

- The NB algorithm also learns:
  - the likelihood of each word in a spam email

```
naive_bayes.feature_log_prob_
```

```
#output:
array([[-9.643029   , -9.67373923, -9.47714135, ..., -6.31041976],
       [-8.23356461, -7.60523447, -9.19154209, ..., -9.19154209]])
```

Predict and evaluate

- Confusion matrix: want most observations on upper left to lower right diagonal

Code 20.1.4 — Naive Bayes. Evaluate on Test Dta

```
# make predictions on the test data
pred = naive_bayes.predict(X_test)

# print confusion matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, pred)

# output:
array([[848,   0],
       [ 32,  88]])
```

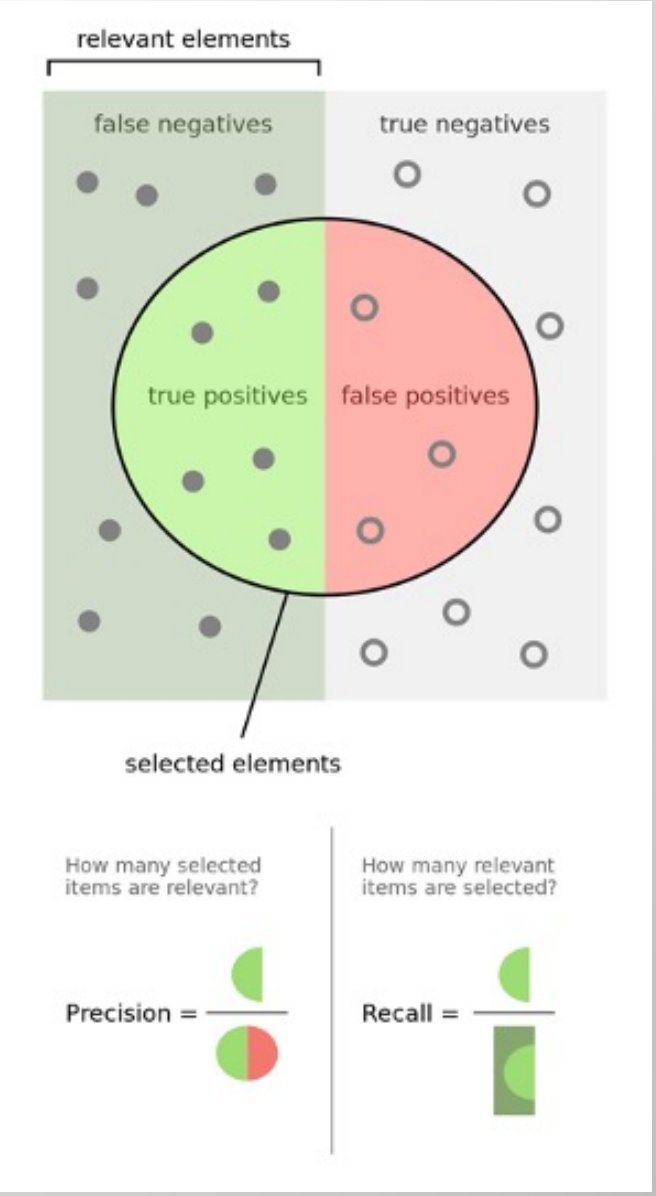
Classification metrics

- Accuracy – percentage of correctly classified examples in the test set
- Precision measures how many observations that were classified as P, really are P

  TP / (TP + FP)

- Recall measures how many true P observations were found

  TP / (TP + FN)



Classification report

- F1 is the geometric mean of precision and recall
- macro average is simple average; weighted avg weights by number of observations in each class

Code 20.1.5 — Naive Bayes. Evaluate Test-set Predictions

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))

# output
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       848
           1       1.00      0.73      0.85       120

    accuracy                           0.97       968
   macro avg       0.98      0.87      0.91       968
weighted avg       0.97      0.97      0.96       968
```

Evaluation

- Is 96.6% accuracy good?
- In the test data, 87.6% of the observations were not spam.
- This is a highly unbalanced data set, so if the classifier guesses not-spam, it will be right 87.6% of the time
- Still, 96.6% is higher, so the classifier learned something

Try again

- The confusion matrix showed 32 messages that were spam, but classified as non-spam
- Visually inspecting these 32 showed they had a lot of numbers, !, CAPS, which the tokenizer removed
- Try different preprocessing (see online notebook)
- The second classifier still had 100% precision, but the recall improved to 85%, bringing the overall accuracy up to 98%

Metrics summary

- Confusion matrix

pred	T	F
T	TP	FP
F	FN	TN

848	0
32	88

- Accuracy: $$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- Recall and precision: $$recall = \frac{TP}{TP + FN} \qquad precision = \frac{TP}{TP + FP}$$

- Accuracy, recall and precision range from [0, 1]

Kappa

- Tries to account for correct prediction by chance

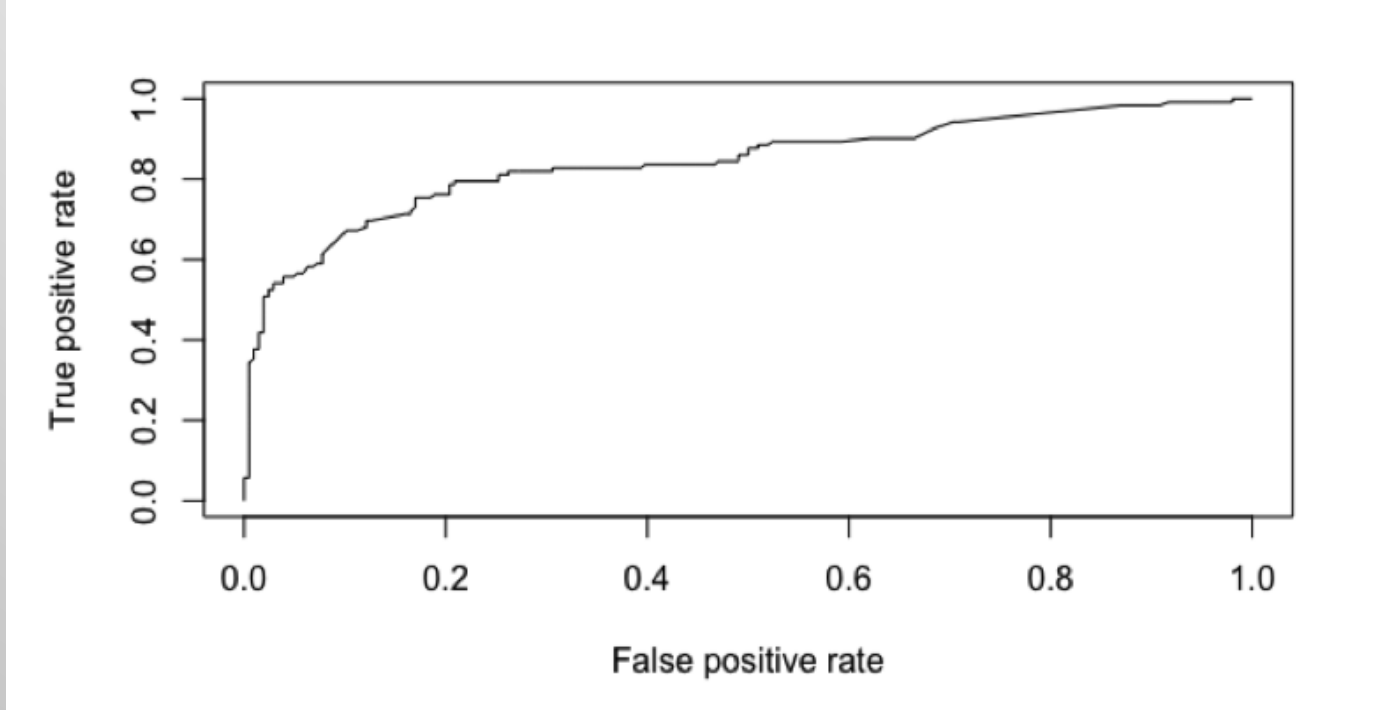
$$\kappa = \frac{Pr(a) - Pr(e)}{1 - Pr(e)}$$

where Pr(a) is the actual agreement and Pr(e) is the expected agreement based on the distribution of the classes. The following interpretation of Kappa is often used but there is not universal agreement on this. Kappa scores:

- <0.2 poor agreement
- 0.2 to 0.4 fair agreement
- 0.4 to 0.6 moderate agreement
- 0.6 to 0.8 good agreement
- 0.8 to 1.0 very good agreement

ROC curve and AUC

- ROC (Receiver Operating Characteristics) was developed in WW2 to distinguish true/false sonar signals
- We want the curve to shoot up to upper left
- AUC is area under curve: 0.5 random guess; 1.0 perfect



Code Examples

- Part 5 Chapter 20 Naïve Bayes notebook

Essential points to note

- Works well on small data
- Fast
- Often a baseline for more complex algorithms
- See sklearn documentation

To Do

- Quiz on Naïve Bayes and Logistic Regression
- Homework: soon