K-Nearest Neighbor Classifier

- Also called “Instance-based learning” or “lazy learning”
  - low/no cost in “training”, high cost in online prediction
- Commonly used in pattern recognition (5 decades)
- Theoretical error bound analyzed by Duda & Hart (1957)
- Applied to text categorization in 1990’s
- Among top-performing text categorization methods
K-Nearest Neighbor Classifier

From all training examples:
- Find $k$ examples that are most similar to the new document
  - “neighbor” documents
- Assign the category that is most common in these neighbor documents
  - neighbors “vote” for the category
- Can also consider the distance of a neighbor
  - a closer neighbor has more weight/influence

• Idea: find your language by what language your neighbors speak
  
1-NN: Red; 5-NN: Brown; 10-NN: ?; Weighted 10-NN: Brown
K Nearest Neighbor: Technical Elements

- Document representation
- Document distance measure: closer documents should have similar labels; neighbors speak the same language
- Number of nearest neighbors (value of K)
- Decision threshold

K Nearest Neighbor: Framework

Training data: \( D = \{(x_i, y_i)\}, \quad x_i \in \mathbb{R}^M, \text{docs}, \quad y_i \in \{0,1\} \)

Test data: \( x \in \mathbb{R}^M \)

The neighborhood is \( D_k \in D \)

Scoring Function: \( \hat{y}(x) = \frac{1}{k} \sum_{x_i \in D_k(x)} \text{sim}(x, x_i)y_i \)

Classification:
\[
\begin{cases} 
1 & \text{if } \hat{y}(x) - t > 0 \\
0 & \text{otherwise}
\end{cases}
\]

Document Representation: \( X_i \) uses tf.idf weighting for each dimension
Choices of Similarity Functions

**Euclidean distance**
\[ d(x_1, x_2) = \sqrt{\sum_v (x_{1v} - x_{2v})^2} \]

**Kullback Leibler distance**
\[ d(x_1, x_2) = \sum_v x_{1v} \log \frac{x_{1v}}{x_{2v}} \]

**Dot product**
\[ x_1 \cdot x_2 = \sum_v x_{1v} \cdot x_{2v} \]

**Cosine Similarity**
\[ \cos(x_1, x_2) = \frac{\sum_v x_{1v} \cdot x_{2v}}{\sqrt{\sum_v x_{1v}^2} \sqrt{\sum_v x_{2v}^2}} \]

**Kernel functions**
\[ k(x_1, x_2) = e^{-d(x_1, x_2)^2/2\sigma^2} \] (Gaussian Kernel)

Automatic learning of the metrics

---

Choices of Number of Neighbors (K)

![Micro-F1 vs Number of Neighbors](attachment:image.png)

Trade off between small number of neighbors and large number of neighbors
Choices of Number of Neighbors (K)

- Find desired number of neighbors by cross validation
  - Choose a subset of available data as training data, the rest as validation data
  - Find the desired number of neighbors on the validation data
  - The procedure can be repeated for different splits; find the consistent good number for the splits

Characteristics of KNN

Pros
- Simple and intuitive, based on local-continuity assumption
- Widely used and provide strong baseline in TC Evaluation
- No training needed, low training cost
- Easy to implement; can use standard IR techniques (e.g., tf.idf)

Cons
- Heuristic approach, no explicit objective function
- Difficult to determine the number of neighbors
- High online cost in testing; find nearest neighbors has high time complexity
Problem: Weighting of Terms

• K-NN treats all terms equally
  – Frequent but unimportant terms may dominate
• Which terms are more important?
  – TF.IDF?
  – ...
• Solution – machine learning
  – We have training data

Naïve Bayes Classification

• Naïve Bayes (NB) Classification
  – Generative Model: Model both the input data (i.e., document contents) and output data (i.e., class labels)
  – Make strong assumption of the probabilistic modeling approach
• Methodology
  – Similar with the idea of language modeling approaches for information retrieval
  – Train a language model for all the documents in one category
Naïve Bayes Classification

• Methodology
  – Train a language model for all the documents in one category
    Category 1: \(\vec{d}_{1,1}, \vec{d}_{1,2}, \ldots, \vec{d}_{1,n_1}\) \(\rightarrow\) Language model \(\theta_1\)
    Category 2: \(\vec{d}_{2,1}, \vec{d}_{2,2}, \ldots, \vec{d}_{2,n_2}\) \(\rightarrow\) Language model \(\theta_2\)
    ...... \(\rightarrow\)
    Category C: \(\vec{d}_{C,1}, \vec{d}_{C,2}, \ldots, \vec{d}_{C,n_C}\) \(\rightarrow\) Language model \(\theta_C\)

  – What is the language model? (Multinomial distribution)
  – How to estimate the language model for all the documents in one category?

Naïve Bayes Classification

• Representation
  – Each document is a “bag of words” with weights (e.g., TF.IDF)
  – Each category is a super “bag of words”, which is composed of all words in all the documents associated with the category
  – For all the words in a specific category c, it is modeled by a multinomial distribution as
    \[ p(\vec{d}_{c,1}, \ldots, \vec{d}_{c,n_c} | \theta_c) \]

  – Each category (c) has a prior distribution \(P(c)\), which is the probably of choosing category c BEFORE observing the content of a document
Naïve Bayes Classification

Maximum Likelihood Estimation:

- Find model parameters for a category that maximizes generation likelihood:
  \[ \theta^*_c = \arg \max_{\theta_c} p(\vec{d}_{c1}, \ldots, \vec{d}_{cn} | \theta_c) \]

There are K words in vocabulary, \( w_1 \ldots w_K \)

Data: documents \( \vec{d}_{c1}, \ldots, \vec{d}_{cn} \)

For \( \vec{d}_c \) with counts \( c_i(w_1), \ldots, c_i(w_K) \), and length \( |\vec{d}_c| \)

Model: multinomial M with parameters \( \{p(w_k)\} \)

Likelihood: \( \Pr(\vec{d}_{c1}, \ldots, \vec{d}_{cn} | \theta) \)

\[ \theta^*_c = \arg \max_{\theta_c} p(\vec{d}_{c1}, \ldots, \vec{d}_{cn} | \theta_c) \]

Maximum Likelihood Estimation (MLE)

\[ p(\vec{d}_{c1}, \ldots, \vec{d}_{cn} | \theta) = \prod_{i=1}^{n_c} \left( \frac{|\vec{d}_{ci}|}{p_{k_i}(w_i)} \right) \prod_{k=1}^{K} p_{c_i}(w_k) = \prod_{i=1}^{n_c} \prod_{k=1}^{K} p_{c_i}(w_k) \]

\[ l(\vec{d}_{c1}, \ldots, \vec{d}_{cn} | \theta) = \log p(\vec{d}_{c1}, \ldots, \vec{d}_{cn} | \theta) = \sum_{i=1}^{n_c} \sum_{k} c_i(w_k) \log p_k \]

\[ l(\vec{d}_{c1}, \ldots, \vec{d}_{cn} | \theta) = \sum_{i=1}^{n_c} \sum_{k} c_i(w_k) \log \theta_k + \lambda \left( \sum_k p_k - 1 \right) \]

\[ \frac{\partial l}{\partial p_k} = \frac{\sum_{i=1}^{n_c} c_i(w_k)}{p_k} + \lambda = 0 \quad \Rightarrow \quad p_k = -\frac{\sum_{i=1}^{n_c} c_i(w_k)}{\lambda} \]

Use Lagrange multiplier approach

Set partial derivatives to zero

Get maximum likelihood estimate

Since \( \sum_k p_k = 1 \), \( \lambda = -\sum_k \sum_{i=1}^{n_c} c_i(w_k) = -\sum_{i=1}^{n_c} |\vec{d}_{ci}| \)

So, \( p_k = p(w_k) = \frac{\sum_{i=1}^{n_c} c_i(w_k)}{\sum_{i=1}^{n_c} |\vec{d}_{ci}|} \)
Naïve Bayes Classification

- **MLE Estimator:** Normalization by simple counting
  - Train a language model for all the documents in one category
  \[
  p(w | \theta^*_c) = \frac{\sum_{i=1}^{n_c} c_{ci}(w)}{\sum_{i=1}^{n_c} |d_{ci}|}
  \]
  \[
  p(c) = \frac{n_c}{\sum_{c'} n_{c'}}
  \]

- **Category Prior:**
  - Number of documents in the category divided by the total number of documents

---

Naïve Bayes Classification

- **Smoothed Estimator:**
  - Laplace Smoothing
  \[
  p(w | \tilde{\theta}^*_c) = \frac{1 + \sum_{i=1}^{n_c} c_{ci}(w)}{K + \sum_{i=1}^{n_c} |d_{ci}|}
  \]

  - Hierarchical Smoothing
  \[
  p(w | \tilde{\theta}^*_c) = \lambda_1 P(w | \tilde{\theta}^*_c) + \lambda_2 P(w | \tilde{\theta}^*_{c_m}) + \ldots + \lambda_m P(w | \tilde{\theta}^*_{c_n})
  \]

  - Dirichlet Smoothing
Naïve Bayes Classification

- Prediction:

\[ c^* = \arg \max_c p(c \mid d_i) \]

\[ = \arg \max_c \left\{ \frac{p(c) p(d_i \mid c)}{p(d_i)} \right\} \]

\[ = \arg \max_c \left\{ p(c) p(d_i \mid c) \right\} \quad \text{(Bayes Rule)} \]

\[ = \arg \max_c \left\{ p(c) \prod_k p(w_k \mid c)^{c_i(w_k)} \right\} \quad \text{(Multinomial Dist)} \]

\[ = \arg \max_c \left\{ \log(p(c)) + \sum_k c_i(w_k) \log(p(w_k \mid c)) \right\} \]

Plug in the estimator

- Example of Binary Classification

Two classes

\[ c^* = \arg \max_{l \in \{-, +\}} p(c_l \mid d_i) \rightarrow \frac{p(c_+ \mid d_i)}{p(c_- \mid d_i)} \]

\[ p(c_+ \mid d_i) \propto \prod_k p(w_k \mid c_+)^{c_i(w_k)} \frac{n_+}{n_+ + n_-} \]

\[ p(c_- \mid d_i) \propto \prod_k p(w_k \mid c_-)^{c_i(w_k)} \frac{n_-}{n_+ + n_-} \]
Example of Binary Classification

\[ c^* = \arg \max_{c \in \{-1, +1\}} p(c \mid \vec{d}) \rightarrow \frac{p(c_+ \mid \vec{d})}{p(c_- \mid \vec{d})} \]

\[
\log \frac{p(c_+ \mid \vec{d})}{p(c_- \mid \vec{d})} = \log \left( \frac{\prod_k [p(w_k \mid c_+)]^{c_i(w_k)} n_+}{\prod_k [p(w_k \mid c_-)]^{c_i(w_k)} n_-} \right) \\
= \log \left( \frac{n_+}{n_-} \right) + \sum_k c_i(w_k) \log \left( \frac{p(w_k \mid c_+)}{p(w_k \mid c_-)} \right) \\
\log \frac{p(c_+ \mid \vec{d})}{p(c_- \mid \vec{d})} \approx b_0 + \sum_k c_i(w_k) \times \text{weight}(w_k)
\]

Naïve Bayes = Linear Classifier

**denotes +1**

**denotes -1**
Naïve Bayes Classification

• Summary
  – Utilize multinomial distribution for modeling categories and documents
  – Use posterior distribution (posterior of category given document) to predict optimal category
• Pros
  – Solid probabilistic foundation
  – Fast online response, linear classifier for binary classification
• Cons
  – Empirical performance not very strong
  – Probabilistic model for each category is estimated to maximize the data likelihood for documents in the category (generative), not for purpose of distinguishing documents in different categories (discriminative)