
UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
UNIDADE ACADÊMICA DE ENGENHARIA ELÉTRICA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Relatório de Estágio

# Uso de Amostras Não-Marcadas para a Melhoria de Desempenho na Classificação de Texto

**André Dieb Martins**

**Prof. Dr. Bruno Barbosa Albert**

Professor Orientador

Campina Grande, Agosto de 2012

**Universidade Federal de Campina Grande**

André Dieb Martins

# Uso de Amostras Não-Marcadas para a Melhoria de Desempenho na Classificação de Texto

Relatório de estágio supervisionado submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para obtenção do título de Engenheiro Eletricista.

**Área de Concentração:** Telecomunicações

Orientador:

Prof. Dr. Bruno Barbosa Albert

Campina Grande

Julho de 2012

**Universidade Federal de Campina Grande**

André Dieb Martins

# Uso de Amostras Não-Marcadas para a Melhoria de Desempenho na Classificação de Texto

Relatório de estágio supervisionado submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para obtenção do título de Engenheiro Eletricista.

**Área de Concentração:** Telecomunicações

Banca Examinadora:

---

Prof. Dr. Bruno Barbosa Albert
(Orientador)
UFCG

---

Professor Convidado
UFCG

Campina Grande
Julho de 2012

# Resumo

Nesse trabalho é apresentado uma técnica de treinamento semi-supervisionado, originalmente introduzida por Kamal Paul Nigam (2001), que utiliza-se de amostras não-marcadas para a melhoria de desempenho do classificador Bayes Ingênuo. Tal metodologia é motivada pela redução de custos na construção de classificadores, visto que amostras marcadas são significativatemente mais dispendiosas que amostras não-marcadas.

Através da utilização do método de Maximização da Expectativa, mostra-se que o método tradicional de treinamento supervisionado pode ser superado ao se introduzir um conjunto de amostras não marcadas, reduzindo a quantidade de amostras marcadas necessárias para atingir certos patamares de desempenho. Experimentos realizados no *corpus* 20Newsgroup-18828 mostraram aumentos de desempenho similares aos obtidos por Nigam, mesmo relaxando algumas das condições impostas. Mais especificamente, ao relaxar a condição cronológica de teste (utilização de documentos mais antigos para o treino e mais recentes para o teste), foi observada uma manutenção das vantagens do método até um máximo de 15 classes.

**Palavras-Chave:** aprendizado de máquina, classificação de texto, aprendizado semi-supervisionado, Bayes ingênuo, maximização da expectativa, modelo mistura, Python, scikit-learn, scipy, numpy

# Abstract

Here we present Kamal Paul Nigam's semi-supervisioned training technique for improving the Naive Bayes classifier's performance through the use of unlabeled samples. This metodology is motivated by the cost reduction achieved when building a classifier using fewer labeled samples (costlier) and more more unlabeled samples.

It's shown that using a combination of the Expectation-Maximization method and Naive bayes learning surpasses the traditional one alone. Through the introduction of unlabeled samples in the learning, it's observed a reduction on the amount of labeled samples needed for achieving several performance levels. The experiments were performed on 20Newsgroups-18828 *corpus* and show similar results to Nigam's, even when relaxing some of the conditions imposed. More specifically, we relaxed the chronologic condition (use older documents for training, newer for testing), resulting in similar positive results while under 15 classes.

**Keywords:** machine learning, text classification, semi-supervised learning, Naive Bayes, Expectation Maximization, Mixture Models, Python, scikit-learn, scipy, numpy

# Lista de Figuras

# Lista de Tabelas

# Sumário

# 1 Introdução

Na atualidade, é notável o crescimento massivo do volume de dados e de seu tráfego na Internet. Essa tendência é alimentada em grande parte pelo uso da rede para aplicações do dia-a-dia, como pesquisas, estudos, comunicações, negócios, redes sociais, dentre inúmeros outros.

Nesse contexto, nos últimos dez anos, a área de Recuperação da Informação (RI) tem adquirido bastante renome por criar sistemas para a gestão e o acesso à documentos digitais. A Categorização de Texto (CT), também conhecida como Classificação de Texto, é uma das tarefas essenciais desses sistemas.

A CT tem uma história na pesquisa desde a década de 60, mas apenas na década de 90 obteve a atenção merecida, com o surgimento de novas aplicações, aliado com a introdução de computadores mais velozes e maior acessibilidade a documentos na forma digital.

## 1.1 A Tarefa de Classificação de Texto

A tarefa de CT é definida como a associação de uma variável Booleana a cada par $(d_i, c_j) \in \mathcal{D} \times \mathcal{C}$, onde $\mathcal{D} = \{d_1, \ldots, d_{|\mathcal{D}|}\}$ é um conjunto de documentos e $\mathcal{C} = \{c_1, \ldots, c_{|\mathcal{C}|}\}$ é um conjunto de categorias nas quais os documentos devem ser classificados [SEBASTIANI 2002]. Associa-se $V$ à $(d_i, c_j)$ se o documento $d_i$ faz parte da classe $c_j$, e $F$ caso contrário. Formalmente, essa associação é definida pela função alvo $\hat{\Phi} : \mathcal{D} \times \mathcal{C} \rightarrow \{V, F\}$, uma função desconhecida que descreve exatamente como os documentos se relacionam com a classe.

A tarefa de CT visa obter uma aproximação de $\hat{\Phi}$, denotada por $\Phi$, usualmente denominada de classificador, modelo ou regra. Note que quanto mais $\Phi$ se aproximar de $\hat{\Phi}$, melhor o classificador descreverá as relações documento-classe.

Os problemas de CT podem ser classificados, dentre outras possibilidades, de acordo com o número de marcações e com o foco do problema. Os problemas de *marcação*

*única* são aqueles em que apenas uma classe é associada a cada documento, enquanto que *marcação múltipla* uma ou mais classes podem estar associadas. No que diz respeito ao foco do problema, na categorização *centrada em documento* deseja-se obter todas as classes $c_j \in \mathcal{C}$ que se apliquem a um dado documento $d_j$, enquanto que na categorização *centrada em categoria* deseja-se listar todos os documentos $d_j \in \mathcal{D}$ que pertençam à classe $c_i$.

Apresentamos a seguir um breve histórico da classificação de texto e do aprendizado de máquina, contextualizado com o problema tratado nesse trabalho.

## 1.2 Breve Histórico da Classificação de Texto

A primeira abordagem prática ao problema de CT se deu com a Engenharia de Conhecimento (do inglês *knowledge engineering*), em que máquinas de CT eram construídas manualmente a partir de um conjunto de regras inflexíveis criadas por operadores humanos. Entretanto, na década de 90 esse paradigma perdeu popularidade para a vertente do Aprendizado de Máquina (AM, do inglês *machine learning*), que fundamenta na probabilidade e estatística a construção automática dessas máquinas.

Em geral, o paradigma AM apresenta como principal vantagem uma performance de classificação comparável a *experts humanos*, entretanto, a um menor custo, visto que não são mais necessários os *engenheiros de conhecimento* ou *experts*. Por outro lado, a introdução do AM não é gratuita, sendo necessárias amostras pré-classificadas para seus processos de aprendizado e avaliação.

Sendo assim, qual é o custo de uma amostra pré-classificada? Bem, pode-se dizer que isso depende da aplicação, do custo de obtenção da amostra e do custo do classificador humano. Por exemplo, suponha que um portal eletrônico de notícias deseja apresentar suas notícias organizadas por categorias e deseja que isso seja feito automaticamente. Para tal, o portal deve inicialmente contratar um ou mais indivíduos capazes de categorizar suas notícias com precisão. Esses, por sua vez, classificarão manualmente uma certa quantidade de notícias de cada categoria (de acordo com algum critério da empresa), formando um conjunto comumente conhecido como *corpus*. O custo, nesse caso, dependerá dos salários dos indivíduos e do tempo total de trabalho que estarão alocados.

Note que, em um portal de notícias, essa tarefa pode se mostrar trivial e consequentemente ter um baixo custo. Entretanto, em áreas como medicina ou processamento de imagens, a tarefa pode adquirir um nível de dificuldade elevado, requisitando julgamento

de profissionais mais experientes e provavelmente mais custosos.

Com isso, no final do século XX e no início do século XXI, surgiu um maior interesse por técnicas que utilizassem menos amostras marcadas para alcançar as mesma marcas de desempenho. Trabalhos como [BLUM & MITCHELL 1998], [JOACHIMS 1998] e [NIGAM et al. 1998] inauguraram esse movimento ao utilizar pequenos conjuntos de amostras marcadas e grandes conjuntos de amostras não-marcadas na melhora de desempenho de classificadores.

No contexto do AM, essa abordagem costuma ser rotulada como *aprendizado semi-supervisionado*. A conotação *supervisionado* vem da necessidade de intervenção humana no aprendizado da máquina, mais especificamente no ato da classificação de amostras, enquanto que o prefixo *semi* indica o caráter parcial dessa intervenção, sinalizando que a máquina tem uma certa autonomia no processo.

## 1.3 Objetivos

Esse trabalho tem como objetivo apresentar o método de treinamento semi-supervisionado do classificador Bayes ingênuo através da técnica Maximização de Expectativa, como proposto por [NIGAM 2001]. Além disso, visa-se demonstrar procedimentos experimentais complementares aos experimentos realizados por Nigam, verificando a funcionalidade do método inclusive sob condições mais gerais de *corpus*.

### 1.3.1 Organização

No Capítulo 2, é apresentada a formulação teórica do modelo generativo de documentos e do classificador Bayes ingênuo, assim como do método de Maximização da Expectativa.

Em seguida, no Capítulo 3, são apresentadas as métricas utilizadas na avaliação de sistemas de CT, necessárias para os testes subsequentes.

Para a verificação do método, foram realizados experimentos sob uma série de condições. Esses são descritos no Capítulo 4, da preparação do experimento à análise dos resultados.

Finalmente, no Capítulo 5 são feitas as considerações finais e apresentadas as sugestões de trabalhos futuros.

## 1.4 O Laboratório

O estágio foi realizado no Laboratório de Automação e Processamento de Sinais (LAPS), que é um laboratório de ensino e pesquisa da Unidade Acadêmica de Engenharia Elétrica (UAEE) da Universidade Federal de Campina Grande (UFCG). O laboratório se situa no bloco CJ do campus I e conta com a participação de alunos de graduação, mestrado e doutorado em suas atividades, atuando em algumas subáreas do processamento digital de sinais com maior concentração nos temas:

- Rádio Definido por Software (SDR);
- Processadores Digitais de Sinais (DSPs);
- Aprendizado de Máquina e Sistemas de Classificação.

O laboratório conta com placas de desenvolvimento USRP (Universal Software Radio Peripheral) para criação de aplicações de SDR e, placas de desenvolvimento equipadas com DSPs da Texas Instruments®, tendo ambos equipamentos utilizados tanto em atividades de pesquisa como de ensino nas disciplinas da graduação de engenharia elétrica.

Em termos de infra-estrutura, o ambiente conta com um sistema de refrigeração de ar-condicionado, computadores, acesso de banda larga à Internet e bancadas de trabalho apropriadas para o desenvolvimento de trabalhos. Essas condições viabilizaram a execução do presente trabalho sem maiores dificuldades.

## 1.5 Aspectos Gerais do Estágio

O estágio realizado se posiciona na categoria de estágio supervisionado, realizado em um período de 120h. Esse período foi dividido nas seguintes etapas:

- 40h foram dedicadas à revisão bibliográfica;
- 20h foram dedicadas ao estudo do arcabouço *scikit-learn* e das bibliotecas que vieram a ser utilizadas no trabalho;
- 60h foram dedicas à experimentação e avaliação dos resultados

# 2 Algoritmo de Nigam: Bayes ingênuo, Maximização de Expectativa

Neste capítulo apresentamos um algoritmo de treinamento semi-supervisionado proposto por Kamal Paul Nigam (2001), que combina o classificador clássico Bayes ingênuo com a técnica *Maximização de Expectativa*. Tal arranjo busca um aumento no desempenho de CT através de aproveitamento de informação proveniente de documentos sem marcação.

## 2.1 Introdução

Em sua tese de doutorado, Kamal Paul Nigam (2001) propôs uma combinação do classificador Bayes ingênuo (NB) com a técnica de maximização de verossimilhança conhecida como *Maximização de Expectativa* (EM). Além da combinação NB+EM, Nigam também investigou outras técnicas para a solução do mesmo problema. Entretanto, dado o período de execução desse trabalho, mantemos foco apenas no método NB+EM.

A principal motivação por trás dessa abordagem é lidar com o problema de classificação quando se tem poucos documentos marcados disponíveis. Esse é um problema comum, visto que documentos marcados são dispendiosos e sua criação costuma necessitar de experts com domínio nas categorias alvo. Outra possível ocorrência desse problema é a existência de limitações técnicas (e.g. aplicações reais em que marcações são obtidas aos poucos).

Em seu trabalho, Nigam mostrou que a combinação NB+EM pode reduzir o erro de classificação. Para tanto, ele inicialmente apresentou um modelo para a geração de documentos de texto. A partir desse modelo, foram derivados os conceitos do classificador NB, levando em conta suas devidas suposições de simplificação. Em seguida, Nigam utilizou a técnica EM para maximização de verossimilhança, sob a suposição de correlação direta entre o modelo mistura e acurácia de classificação, ou seja, ele conjecturou que a maximização resulta em uma melhora nos índices de acurácia.

## 2.2 Modelo Generativo para os Documentos

Essa seção diz respeito ao modelo probabilístico supostamente utilizado na geração dos documentos de texto. Esse modelo se baseia em três princípios:

- os textos são produzidos por um modelo mistura ($\theta$);
- existe uma correspondência um para um entre componentes da mistura e classes;
- as componentes da mistura são distribuições multinomiais de palavras individuais, isto é, as palavras de um documento são independentes umas das outras.

Por exemplo, dado um conjunto de classes $\mathcal{C}$ e um vocabulário $\mathcal{V}$, podemos gerar um documento da seguinte forma:

1. Jogue um dado enviesado[1] de $|\mathcal{C}|$ lados para determinar a classe $c_j$ do documento;
2. Escolha o dado enviesado de $|\mathcal{V}|$ lados correspondente a classe resultante $c_j$;
3. Jogue o dado $|d|$ vezes e anote as palavras resultantes;

Formalmente, o modelo de mistura $\theta$ é composto pelas componentes $c_j \in \mathcal{C} = \{c_1, \ldots, c_{|\mathcal{C}|}\}$. A criação de um documento $d_i$ se dá inicialmente pela escolha de uma das componentes da mistura, de acordo com suas probabilidades de evento $P(c_j|\theta)$. Em seguida, o documento é gerado de acordo com sua própria probabilidade de evento $P(d_i|c_j;\theta)$. Dessa forma, a verossimilhança relacionada ao documento $d_i$ pode ser escrita como

$$P(d_i|\theta) = \sum_{c_j \in \mathcal{C}} P(c_j|\theta) P(d_i|c_j;\theta). \tag{2.1}$$

O documento $d_i$ é composto por um conjunto ordenado de palavras $< w_{d_{i,1}}, w_{d_{i,2}}, \cdots >$, em que $w_{d_{i,k}}$ é a $k$-ésima palavra do documento $d_i$. Supondo que o comprimento do documento $|d_i|$ é escolhido independentemente da classe e das palavras sorteadas, podemos escrever

[1] Pode-se entender um dado enviesado como sendo aquele cuja geometria apresente viés de ocorrência de seus lados. Por exemplo, pode-se lapidar um dado de 6 faces de maneira que a face 1 ocorra com maior frequência do que as demais.

$$
\begin{aligned}
P(d_i|c_j;\theta) &= P(< w_{d_{i,1}}, \ldots, w_{d_{i,|d_i|}} > |c_j;\theta) \\
&\overset{(a)}{=} P(|d_i|)\prod_{k=1}^{|d_i|} P(w_{d_{i,k}}|c_k;\theta;w_{d_{i,q}}, q<k) \\
&\overset{(b)}{=} P(|d_i|)\prod_{k=1}^{|d_i|} P(w_{d_{i,k}}|c_k;\theta), \qquad (2.2)
\end{aligned}
$$

em que $(a)$ segue da suposição de que a geração de palavras é condicionada nas palavras anteriores. Como simplificação, relaxa-se esse condicionamento em $(b)$ e portanto temos um processo independente de geração de palavras.

Nos capítulos e seções a seguir, utilizamos a seguinte notação para se referir ao modelo definido e aos parâmetros relacionados:

- Conjunto de classes: $\mathcal{C} = \{c_1, c_2, \ldots, c_{|\mathcal{C}|}\}$, vocabulário: $\mathcal{V} = \{w_1, w_2, \ldots, w_{|\mathcal{V}|}\}$; Corpus: $\mathcal{D} = \{d_1, \ldots, d_{|\mathcal{D}|}\}$;
- Parâmetros do modelo:
  - Probabilidades de classe: $\theta_{c_j} \triangleq P(c_j|\theta)$;
  - Probabilidades de palavras: $\theta_{w_t|c_j} \triangleq P(w_t|c_j;\theta)$;
- Modelo completo: $\theta = \{\theta_{w_t|c_j} : w_t \in \mathcal{V}; \theta_{c_j} : c_j \in \mathcal{C}\}$

Nas seções seguintes, são apresentados os resultados de estimativa de máxima verossimilhança de probabilidades de classe e palavra, de acordo com a conjectura apresentada anteriormente.

## 2.3 Treinamento do Classificador Bayes Ingênuo

O treinamento do classificador Bayes Ingênuo (NB, do inglês *Naive Bayes*) consiste em estimar os parâmetros do modelo generativo $\theta$ a partir de um conjunto de documentos marcados $\mathcal{D}$, sendo a estimativa denotada por $\hat{\theta}$. O classificador NB utiliza uma estimativa baseada na maximização a posteriori de probabilidade (MAP), isto é, obtem-se $\arg\max_\theta P(\theta|\mathcal{D})$, isto é, o valor de $\theta$ mais provável dada a evidência empírica e uma distribuição a priori.

Nigam (2001) escolheu uma distribuição a priori formada pelo produto de distribuições de Dirichlet. A fórmula de estimativa dos parâmetros resultante da maximização, dada a evidência e a distribuição a priori, é simplesmente uma razão de contagens. Sendo assim, a estimativa da probabilidade de classe (ou peso da componente de mistura) é dada pela razão entre a quantidade de documentos pertencentes àquela classe pela quantidade total de documentos,

$$\begin{aligned} \hat{\theta}_{c_j} &\triangleq P(c_j|\hat{\theta}) \\ &= \frac{1+\sum_{i=1}^{|\mathcal{D}|} z_{ij}}{|\mathcal{C}|+|\mathcal{D}|}, \end{aligned} \tag{2.3}$$

em que $\mathcal{D}$ é o conjunto de documentos, $\mathcal{C}$ é o conjunto de classes, $z_{ij}$ é a função indicadora de pertinência do documento $d_i$ à classe $c_j$ (i.e. $z_{ij} = 1$ para $y_i = c_j$, 0 caso contrário) e ambos numerador e denominador são expandidos com constantes a fim de evitar probabilidades nulas (*Laplace smoothing*). Similarmente, a distribuição de probabilidades de palavra é dada pela razão

$$\begin{aligned} \hat{\theta}_{w_t|c_j} &\triangleq P(w_t|c_j;\hat{\theta}) \\ &= \frac{1+\sum_{i=1}^{|\mathcal{D}|} z_{ij} N(w_t,d_i)}{|V|+\sum_{s=1}^{|\mathcal{V}|}\sum_{i=1}^{|\mathcal{D}|} z_{ij} N(w_s,d_i)}, \end{aligned} \tag{2.4}$$

em que $N(w_s, d_i)$ é contagem de aparições da palavra $w_s$ no documento $d_i$ e $z_{ij}$ é a função indicadora de classe, definida anteriormente.

## 2.4 Classificação de Documentos com Classificador Bayes Ingênuo

Pela regra de Bayes em conjunto com as equações (2.1) e (2.2), podemos escrever a probabilidade de ocorrência da classe $c_j$ dado o documento $d_i$, sob a ótica do modelo estimado $\hat{\theta}$, como

$$
\begin{aligned}
P(y_i = c_j|d_i;\hat{\theta}) &= \frac{P(c_j|\hat{\theta})P(d_i|c_j;\hat{\theta})}{P(d_i|\hat{\theta})} \\
&= \frac{P(c_j|\hat{\theta})\prod_{k=1}^{|d_i|}P(w_{d_{i,k}}|c_j;\hat{\theta})}{\sum_{r=1}^{|\mathcal{C}|}P(c_r|\hat{\theta})\prod_{k=1}^{|d_i|}P(w_{d_{i,k}}|c_r;\hat{\theta})},
\end{aligned} \tag{2.5}
$$

Dado o modelo $\hat{\theta} = \{\hat{\theta}_{w_t|c_j} : w_t \in \mathcal{V}; \hat{\theta}_{c_j} : c_j \in \mathcal{C}\}$, podemos reescrever a equação (2.5) como

$$
P(y_i = c_j|d_i;\hat{\theta}) = \frac{\hat{\theta}_{c_j}\prod_{k=1}^{|d_i|}\hat{\theta}_{w_{d_{i,k}}|c_j}}{\sum_{r=1}^{|\mathcal{C}|}\hat{\theta}_{c_r}\prod_{k=1}^{|d_i|}\hat{\theta}_{w_{d_{i,k}}|c_r}}, \tag{2.6}
$$

Sendo a classe real do documento $d_i$ denotada por $c_j$, a tarefa de CT para o classificador NB consiste em escolher a classe melhor posicionada em um *ranking* das probabilidades $P(y_i = c_j|d_i;\hat{\theta}), c_j \in \mathcal{C}$. Em outras palavras, essa tarefa consiste em determinar, dado o modelo probabilístico e o documento $d_i$, qual classe é mais provável. Matematicamente, essa escolha se traduz em

$$
\hat{y}_i = \arg\max_{c_j \in \mathcal{C}} P(y_i = c_j|d_i;\hat{\theta}), \tag{2.7}
$$

em que $P(y_i = c_j|d_i;\hat{\theta})$ é dada pela equação (2.6), e $\hat{y}_i$ é a classe estimada para o documento $d_i$.

## 2.5 Maximização da Expectativa

Dado um conjunto de documentos de treinamento $\mathcal{D}$, nosso objetivo é construir um classificador NB como descrito nas seções anteriores, cujo processo é ilustrado na Figura 1. Tradicionalmente, são conhecidas todas as classes $y_i \in \mathcal{C}$ dos documentos de treinamento $d_i \in \mathcal{D}$ e portanto todo o conjunto pode ser utilizado no treinamento indutivo. Posteriormente à construção, o desempenho do classificador é testado utilizando outro conjunto de documentos de teste $\mathcal{D}_t$.


***Figura 1*** *– Esquema de Treinamento Supervisionado*

Por outro lado, suponha o caso em que que se conhece apenas uma parte das classes $y_i$ dos documentos $d_i \in \mathcal{D}$, isto é, $\mathcal{D}$ é formado por duas partições disjuntas, $\mathcal{D}_l$ de documentos de classe conhecida e $\mathcal{D}_u$ de documentos de classe desconhecida, arranjo este apresentado na Figura 2). Nessas condições, espera-se que o treinamento tradicional sob o conjunto $\mathcal{D}_l$ gere um desempenho inferior ao caso em que $\mathcal{D}_l = \mathcal{D}$, visto que menos documentos marcados estão disponíveis.

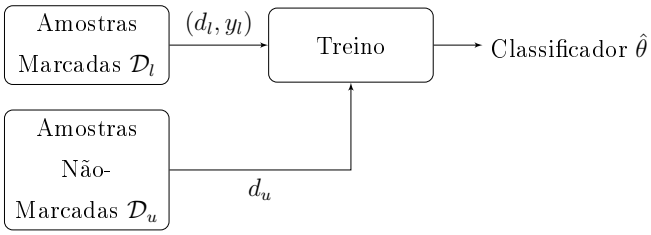


***Figura 2*** *– Esquema de Treinamento Semi-Supervisionado*

Como descrito na seção 2.3, o treinamento NB é abordado como uma maximização a priori (MAP) de $\theta$, isto é, o cálculo de $\arg\max_\theta P(\theta|\mathcal{D})$. Como o arg max do logarítmo do operando corresponde ao máximo do operando, podemos escrever

$$\begin{aligned} \arg\max_\theta P(\theta|\mathcal{D}) &= \arg\max_\theta \log P(\theta|\mathcal{D}) && (2.8) \\ &= \arg\max_\theta \log P(\mathcal{D}|\theta)P(\theta) \\ &= \arg\max_\theta \left[\log P(\theta) + \log P(\mathcal{D}|\theta)\right]. && (2.9) \end{aligned}$$

O segundo termo da equação (2.9), $\log P(\mathcal{D}|\theta)$, é a probabilidade de todos os documentos do dado espaço de documentos. Dado que no modelo suposto os documentos são independentes entre si, essa probabilidade é dada pelo produto das probabilidades de cada documento. A probabilidade de um documento marcado é $P(y_i = c_j|\theta)P(d_i|y_i = c_j;\theta)$, enquanto que para um documento não-marcado é $\sum_{j=1}^{\mathcal{C}} P(c_j|\theta)P(d_i|c_j;\theta)$. Sendo assim, o segundo termo da eq. (2.9) pode ser escrito como

$$\begin{aligned}\log P(\mathcal{D}|\theta) &= \log\left[\prod_{d_i\in\mathcal{D}_u}\sum_{j=1}^{|\mathcal{C}|}P(c_j|\theta)P(d_i|c_j;\theta)\times\prod_{d_i\in\mathcal{D}_l}P(y_i=c_j|\theta)P(d_i|y_i=c_j;\theta)\right]\\ &= \sum_{d_i\in\mathcal{D}_u}\log\sum_{j=1}^{|\mathcal{C}|}P(c_j|\theta)P(d_i|c_j;\theta)+\sum_{d_i\in\mathcal{D}_l}\log P(y_i=c_j|\theta)P(d_i|y_i=c_j;\theta)\\ &.\end{aligned}\tag{2.10}$$

Chamamos essa expressão de probabilidade incompleta e, visto que desconhecemos as classes dos documentos não-marcados, seu cálculo se torna complicado ou até impossível. Por outro lado, se soubéssemos a priori os valores da função indicadora de classe $z_{ij}$, poderíamos calcular $\log P(\mathcal{D}|\theta)$ como:

$$\log P(\mathcal{D}|\theta)=\sum_{i=1}^{|\mathcal{D}|}\sum_{j=1}^{|\mathcal{C}|}z_{ij}\log\left(P(c_j|\theta)P(d_i|c_j;\theta)\right).\tag{2.11}$$

Note que, ao substituirmos $z_{ij}$ pelo seu valor esperado, para documentos não-marcados, temos que a equação (2.11) limita inferiormente a equação (2.10), sendo esta uma aplicação da desigualdade de Jensen (e.g. $\mathrm{E}[\log X]\geq\log\mathrm{E}[X]$). Como resultado, pode-se localizar um máximo local $\hat{\theta}$ através de um procedimento de escalada de colina [1]. Essa técnica é formalmente estudada sob o nome *Maximização da Expectativa* (EM) e foi provada por [DEMPSTER, LAIRD & RUBIN 1977].

Note que $z_{ij}$ é desconhecido para documentos não-marcados e conhecido para documentos marcados. Para documentos não-marcados, substituiremos $z_{ij}$ pelo seu valor esperado, dado o modelo estimado atual ($P(c_j|d_i,\hat{\theta})$).

Considere $\hat{\mathbf{z}}^{(k)}$ e $\hat{\theta}^{(k)}$ as estimativas de $\mathbf{z}=[z_{ij}]$ e $\theta$ na $k$-iteração. O algoritmo EM localiza um máximo local da equação (2.11) através da iteração entre os seguintes passos:

- Passo E: ajuste $\hat{\mathbf{z}}^{(k+1)}=\mathrm{E}[\mathbf{z}|\mathcal{D};\hat{\theta}^{(k)}]$
- Passo M: ajuste $\hat{\theta}^{(k+1)}=\arg\max_\theta P(\theta|\mathcal{D};\hat{\mathbf{z}}^{(k+1)})$

Na prática, o passo E trata do cálculo das probabilidades $P(c_j|d_i;\hat{\theta})$ para os documentos não-marcados utilizando o estado atual do classificador ($\hat{\theta}$) na equação (2.6). O

[1] A escalada de colina é um procedimento otimização que faz parte da familia de métodos de busca local. Trata-se de um algoritmo iterativo que busca máximos locais de uma dada função através da modificação de um de seus parâmetros até que não hajam mais variações significativas no valor da mesma [RUSSELL & NORVIG 2003].

passo M corresponde ao cálculo da nova estimativa de máximo a posteriori dos parâmetros do classificador $\hat{\theta}$ utilizando as estimativas atuais $P(c_j|d_i;\hat{\theta})$ nas equações (2.3) e (2.4). Esse laço pode ser representado pelo sistema apresentado na figura 3, em que o treino é realimentado com o conjunto de amostras sem marcação e suas estimativas de classe.

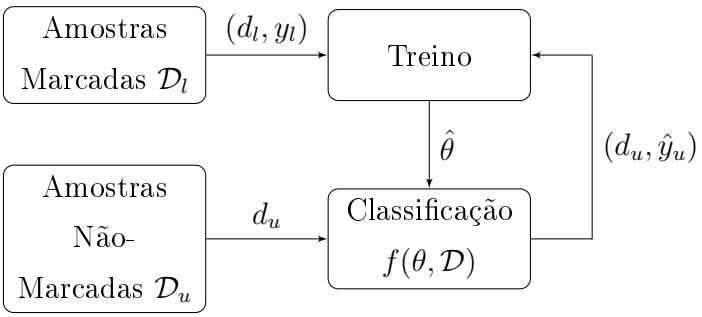


***Figura 3*** *– Esquema de Treinamento Semi-Supervisionado com Laço EM*

Na verificação do método, o classificador será submetido a testes e avaliado seguindo alguns critérios, apresentados no capítulo a seguir.

# 3 Avaliação de Desempenho

Na avaliação de classificadores, uma prática comum é a criação de um experimento baseado em um conjunto de documentos $\mathcal{D}$ (também conhecido como *corpus*) e posterior avaliação do classificador utilizando métricas de desempenho. Em geral, particiona-se $\mathcal{D}$ em dois conjuntos disjuntos, $\mathcal{D}_{treino}$ e $\mathcal{D}_{teste}$, com propósito de averiguar o comportamento do classificador quando submetido à novos documentos. No processo de teste, os documentos $d_i \in \mathcal{D}_{teste}$ são submetidos ao classificador, resultando numa estimativa de classe $\hat{y}_i$. Finalmente, compara-se $\hat{y}_i$ com a classe verdadeira $y_i$ a fim de averiguar acertos e erros. Esse levantamento é feito utilizando as ferramentas apresentadas a seguir.

## 3.1 Matriz de Confusão

Define-se a *matriz de confusão* como sendo uma matriz composta de contagens de erros e acertos de classificação. Mais especificamente, o elemento $a_{ij}$ da matriz corresponde ao número de predições realizadas que resultaram na classe $c_i$ cuja classe verdadeira era $c_j$, isto é, as linhas correspondem às classes verdadeiras e as colunas às classes preditas. Note que para $i = j$, $a_{ij}$ representa o número de predições corretas da classe $c_i$.

Por exemplo, considere uma tarefa de classificação com três classes cujos testes resultaram na matriz de confusão

$$A = \begin{bmatrix} 5 & 1 & 2 \\ 1 & 8 & 1 \\ 2 & 3 & 7 \end{bmatrix}. \tag{3.1}$$

Dessa matriz, podemos constatar 5, 8 e 7 predições corretas das três classes, observadas na diagonal principal. Note que foi feita uma predição errônea da $2^a$ classe quando a correta seria a $1^a$ (elemento $a_{12}$) e outra da $3^a$ classe quando a correta seria a $2^a$ (elemento $a_{23}$). As demais constatações de erros são sugeridas como exercício ao leitor.

## 3.2 Tabela de Confusão

Como visto na seção anterior, a matriz de confusão trata-se de uma compilação de dados com respeito à todas as classes da tarefa de CT. Na definição das métricas, entretanto, torna-se conveniente definir a *tabela de confusão*, que diz respeito aos erros e acertos de uma determinada classe.

Mostramos a seguir a estrutura de uma tabela de confusão na tabela 1, descrevendo as definições logo em seguida.

| *Classe* $c_j$ | Classes Preditas | |
|---|---|---|
| Classes Verdadeiras | Verdadeiro Positivo ($VP_j$)<br>Falso Positivo ($FP_j$) | Falso Negativo ($FN_j$)<br>Verdadeiro Negativo ($VN_j$) |

***Tabela 1*** *– Tabela de Confusão da classe* $c_j$

*Verdadeiro Positivo* é definido como o total de predições da classe $c_j$ que de fato se tratava da classe $c_j$. Em complementar, *Falso Positivo* é o total de predições que marcaram documentos como da classe $c_j$ quando na verdade pertenciam a outra.

Na segunda coluna da tabela 1, observamos inicialmente o *Falso Negativo*, que corresponde ao total de predições que marcaram documentos da classe $c_j$ como de outras classes. Finalmente, temos os *Verdadeiros Negativos*, que compreendem as predições de documentos de outras classes que foram classificados corretamente como não pertencentes à classe $c_j$.

A tabela de confusão é um instrumento bastante utilizado, por exemplo, na medicina. Suponha que um médico dá diagnósticos de pacientes como *doentes* ou *saudáveis*. Sendo assim, as definições de VP, FP, VN e FN podem ser interpretadas como

- Verdadeiro positivo: um paciente doente foi diagnosticado como doente;
- Falso positivo: um paciente saudável foi diagnosticado como doente;
- Verdadeiro negativo: um paciente saudável foi diagnosticado como saudável;
- Falso negativo: um paciente doente foi diagnosticado como saudável.

## 3.3 Métricas de Desempenho

Na avaliação de sistemas,é comum se buscar medidas numéricas que valorizem seu desempenho. Nos sistemas de CT e RI, em especial, uma série de medidas foram criadas com este propósito. As mais usuais são a acurácia, precisão, cobertura (ou recall) e F1, apresentadas a seguir.

### 3.3.1 Acurácia

A acurácia é definida pela razão entre o número de predições corretas realizadas e o total de predições realizadas,

$$\text{Acurácia} = \frac{\#\text{ predições corretas}}{\#\text{ de predições}}. \tag{3.2}$$

Geralmente não se utiliza a acurácia como uma métrica confiável, pois é completamente suscetível ao viés de classes no conjunto de treinamento. Por exemplo, dada uma tarefa de CT e um conjunto de treinamento composto por 90 documentos de uma certa classe A e 10 documentos de uma outra classe B, é possível construir um classificador que marque qualquer documento como A, sem ao menos analisar seu conteúdo. Note que esse classificador poderá alcançar 90% de acurácia, mesmo errando na predição de todos os documentos da classe B. Vale notar que esse problema pode ser amenizado ao se utilizar um conjunto de treinamento balanceado, onde se encontra presente um número igual de documentos por classe.

Em termos da tabela de confusão (Tabela 1), a acurácia em uma dada classe $c_i$ pode ser calculada através da fórmula

$$Acc_i = \frac{VP_i + VN_i}{VP_i + FP_i + FN_i + VN_i}. \tag{3.3}$$

O complemento da acurácia é o erro de classificação, dado por

$$Err_i = \frac{FP_i + FN_i}{VP_i + FP_i + FN_i + VN_i}. \tag{3.4}$$

### 3.3.2 Precisão

A precisão é uma métrica derivada da tabela de confusão e indica a proporção de predições positivas verdadeiras (VPs) realizadas dentre todas as predições positivas presentes. No campo da recuperação da informação, essa definição é interpretada como a probabilidade de que um documento recuperado seja relevante. Em ambos casos, a precisão é dada pela razão

$$\text{Precisão} = \frac{\#\text{ de predições positivas verdadeiras}}{\#\text{ de predições positivas}}.$$

Em termos das variáveis da tabela de confusão, pode-se escrever a precisão $P_i$ da classe $c_i$ como

$$P_i = \frac{VP_i}{VP_i + FP_i}. \tag{3.5}$$

Vale notar que a precisão $P_i$ assume valores no intervalo $[0, 1]$. Note que a precisão é máxima ($P_i = 1.0$) na ausência de falsos positivos ($FP_i = 0.0$), ou seja, todos as predições de documentos como da classe $c_i$ estavam corretas. De outro ponto de vista, pode-se afirmar que nenhum documento de outra classe foi predito errôneamente como sendo da classe $c_i$.

A precisão alcança um mínimo ($P_i = 0.0$) na ausência de VPs ($VP_i = 0.0$) ou quando a fração de VPs for insignificante quando comparada com os FPs ($FP_i \to \infty$). Sendo assim, a precisão é nula quando todas as predições de documentos que resultam na classe $c_i$ são errôneas, isto é, os documentos pertencem a outras classes.

### 3.3.3 Cobertura (Recall)

A cobertura, também conhecida como *recall*, é uma métrica derivada da tabela de confusão e indica a proporção de predições positivas verdadeiras (VPs) realizadas dentre todos os documentos positivos. No campo da recuperação da informação, costuma-se interpretar essa definição como a probabilidade de se recuperar um documento relevante. Essa razão é dada pela expressão

$$\text{Cobertura} = \frac{\#\text{ de predições positivas verdadeiras}}{\#\text{ de documentos positivos}}. \tag{3.6}$$

Em termos das variáveis da tabela de confusão, pode-se escrever a cobertura $C_i$ da classe $c_i$ como

$$C_i = \frac{VP_i}{VP_i + FN_i}. \tag{3.7}$$

Assim como a precisão, a cobertura assume valores no intervalo $[0, 1]$. A cobertura máxima ($C_i = 1.0$) é alcançada para $FN_i = 0.0$, isto é, não há presença de falsos negativos e portanto, não houveram predições de documentos da classe $c_i$ marcados como de outra classe. O mínimo é dado para $VP_i = 0.0$ ou $FN_i \to \infty$, isto é, nenhuma predição de documentos da classe $c_i$ foi correta.

### 3.3.4 F1

As métricas E e F foram introduzidas por Van Rijsbergen [RIJSBERGEN 1979] e avaliam o desempenho de sistemas de recuperação da informação. A métrica F1, caso particular da métrica F, indica o quanto a precisão e a cobertura se sobrepõem, sendo confiável na avaliação de desempenho de classificadores. O valor de F1 para uma dada classe $c_i$ é dado pela média harmônica

$$F1_i = \frac{2P_iR_i}{P_i + R_i}, \tag{3.8}$$

em que $P_i$ e $R_i$ são respectivamente a precisão e a cobertura (ou *recall*) da classe $c_i$.

# 4 Experimentos

Nesse capítulo são descritos os experimentos realizados nesse trabalho com intuito de se verificar o método NB+EM proposto. Nigam (2001) realizou experimentos na verificação do método utilizando três diferentes *corpora*: coleção de discussões de grupos da UseNet (20Newsgroups), coleção de páginas web (WebKB) e coleção de notícias (Reuters). Dentre seus resultados, ele observou que o uso de documentos sem marcação melhora a acurácia do classificador NB. Além disso, observou uma significativa melhoria para o caso de um baixo número de documentos marcados no treinamento, alcançando redução de até 30% no erro de classificação para 300 documentos de treinamento e 10000 documentos sem marcação.

Em seus experimentos, Nigam utilizou o método de extração de características de vetorização por contagem (do inglês *Count Vectorizer*). A vetorização inicia-se com a compilação de um vocabulário de palavras presentes nos documentos de treinamento. Em seguida, é feita uma contagem dessas palavras para cada documento dos conjuntos de treinamento e teste, resultando nos respectivos vetores de características.

Nigam optou por não utilizar um método de seleção de características, visto que seu uso reduziria a dimensionalidade dos documentos sem marcação, acarretando uma potencial perda de informação e prejudicando o método proposto.

Na preparação do experimento, ele selecionou para teste os documentos mais recentes (20% do total), sob o argumento de dependência temporal entre os documentos (i.e. documentos mais novos tem conteúdo dependente dos documentos mais antigos). Além disso, ele efetuou uma distribuição uniforme de classes nos conjuntos, ou seja, cada conjunto (treinamento, teste, não-marcados) tem o mesmo número de documentos por classe. 10000 documentos foram utilizados para formar o conjunto de documentos sem marcação e 6000 foram utilizados para formar o conjunto de documentos marcados para treinamento.

Nesse trabalho, em complemento aos experimentos de Nigam, buscamos mostrar a efetividade do método NB+EM em *corpus* mais escassos de documentos. Além disso,

ignorou-se a dependência temporal entre os documentos. Os conjuntos (treinamento, teste e não-marcados) foram escolhidos aleatóriamente do *corpus*.

O *corpus* utilizado neste trabalho foi o "20Newsgroups-18828- uma coleção de 18828 mensagens de grupos de notícias da rede UseNet. Essa coleção é derivada da coleção original "20Newsgroups-19997- coletada por Ken Lang - em que foram removidas mensagens duplicadas e campos de metadados das mensagens. Foram mantidos apenas o texto das mensagens e os endereços eletrônicos dos remetentes. Essa coleção encontra-se disponível tanto no portal MLcomp.org quanto no arcabouço scikit-learn.

## 4.1 Implementação

A implementação do algoritmo NB+EM utilizou-se do arcabouço **scikit-learn** [PEDREGOSA et al. 2011]. Esse arcabouço foi escrito na linguagem de programação Python, e utiliza-se de bibliotecas científicas de grande renome na comunidade Python (numpy, scipy, matplotlib [JONES et al. 2001–]).

O arcabouço conta com implementações dos principais classificadores conhecidos (e.g. SVM, Bayes ingênuo, kNN), assim como facilidades de acesso à diferentes *corpora*, como por exemplo o 20Newsgroups, dados de doenças oculares, e inclusive acesso as bibliotecas MLComp.org e MLData.org, que possuem um grande acervo de *corpora* para aprendizado de máquina. Em seu acervo, também conta com implementações algoritmos de treinamento semi-supervisionado e não supervisionado, esquemas de validação cruzada e inclusive as métricas de avaliação (precisão, cobertura, recall, etc).

A escolha do arcabouço se deu por maior experiência com a linguagem por parte do autor, assim como pelo *status* do arcabouço, por ter inúmeros contribuidores de prestígio, apoio de grandes empresas como por exemplo o Google Inc. e qualidade constatada em seus resultados.

## 4.2 Corpus e Protocolo de Tratamento

O corpus utilizado foi o 20Newsgroups-18828, contendo um total de 18828 documentos divididos em 20 categorias. Em nossos testes, utilizamos 8 categorias, resultando em uma soma de 7476 documentos cujo vocabulário totalizou 44200 palavras únicas.

Para o treinamento, foram utilizados aproximadamente 30% dos documentos, equiv-

alente a 2242 documentos. Foram reservados 748 para o teste e para documentos não marcados do método NB+EM, foram reservados os 4486 restantes.

O experimento realizado verificou a funcionalidade do método NB+EM para um *corpus* menor do que o utilizado por Nigam (20Newsgroups-19999 com todas as categorias). Além disso, o experimento buscou verificar a validade do método ignorando a suposição de dependência temporal de Nigam, em que ele utilizou apenas os documentos datados mais recentes em seus testes.

Em nossos testes, verificamos que o uso de vetorização TF-IDF prejudicou a performance (como constatado por Nigam). Sendo assim, utilizamos vetorização por contagem na preparação dos documentos, em que os vetores de características são montados a partir de contagens de ocorrências das palavras do vocabulário.

Em seguida, foram avaliados os classificadores NB tradicional e NB+EM sob as métricas definidas na seção 3.3, submetidos a progressivas quantidades de documentos marcados na etapa de treinamento, isto é, foram alimentados no treinamento com $8, 16, \ldots, 2242$ documentos. O propósito desse arranjo foi verificar a melhoria das métricas por parte do NB+EM quando comparado com o NB tradicional.

## 4.3 Resultados

O algoritmo NB+EM trata-se de um processo iterativo de maximização da probabilidade de verossimilhança, como descrito na seção 2.5. Em geral, o processo iterativo tomou uma média de 15 iterações até convergir, utilizando como condição de parada as mudanças no valor da equação (2.11).

Ao realizar o experimento com 8 categorias e preparação descrita como na seção 4.2, obtivemos os resultados apresentados na Figura 4 a seguir.

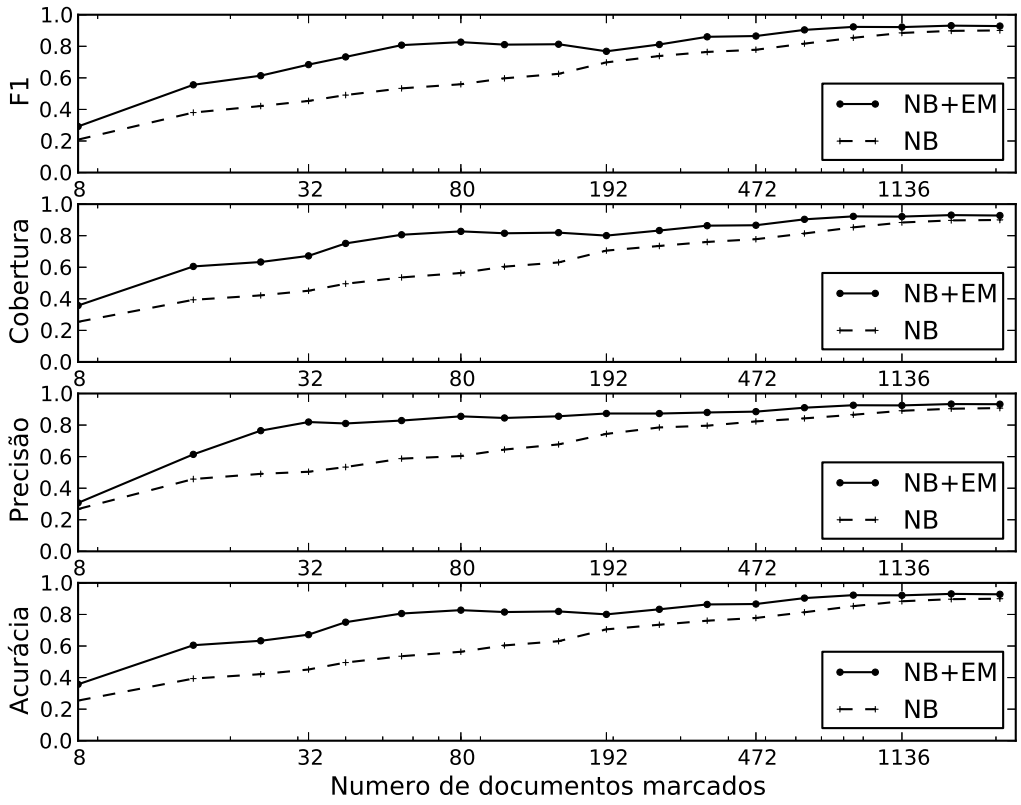


***Figura 4*** *– Métricas F1, Cobertura, Precisão e Acurácia de classificadores NB e NB+EM para 8 classes vs Número de documentos marcados (treinamento)*

Da Figura 4, observamos que o classificador NB+EM apresenta todas as métricas superiores ao NB tradicional.

Em seu experimento, Nigam constatou uma melhora na acurácia com maior significância na região inicial, onde se utilizam poucos documentos marcados. Comparativamente, nossos resultados apresentam que, para menos classes e ignorância da dependencia temporal, o classificador NB+EM apresenta melhorias mais significativas numa região mais central, mais especificamente na faixa $[32, 80]$ do eixo horizontal. Diferentemente de Nigam, optamos por apresentar os gráficos completos de cada uma das métricas, ao invés de apresentar apenas o melhor ponto de balanço entre a precisão e a cobertura (*break-even point*).

Para a acurácia, principal medida utilizada por Nigam em sua análise, observamos uma melhoria mais significativa numa faixa mais larga, $[16, 192]$ do eixo horizontal, como apresentado no gráfico mais detalhado da Figura 5. Com esses resultados, podemos constatar a melhoria proveniente da Maximização de Expectativa e aproveitamento da

informação presente nos documentos não-marcados.

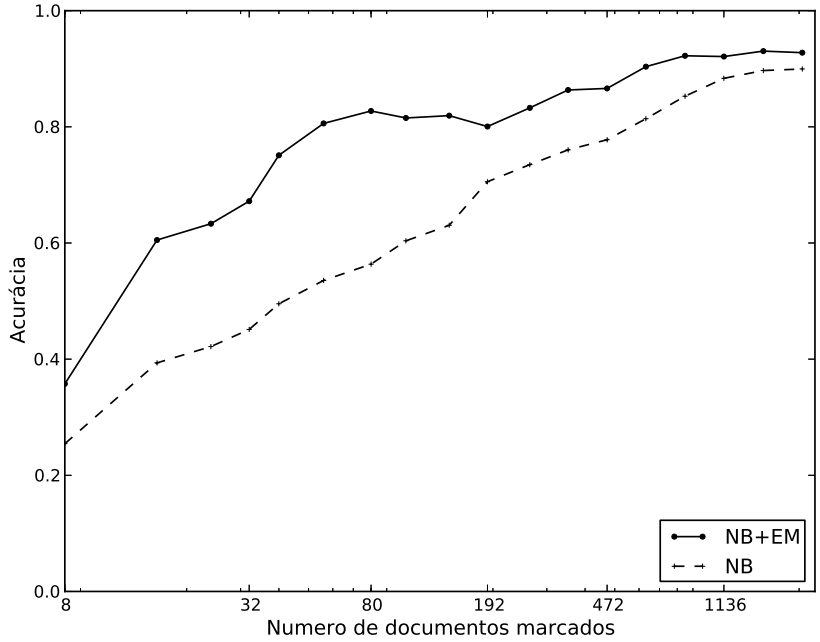


***Figura 5*** *– Acurácia para experimento com 8 classes*

Dos gráficos das figuras 4, 5, pode-se constatar que com o aumento do número de documentos marcados, os classificadores tendem a convergir a um ponto comum. Ao fim do experimento, o classificador NB+EM apresentou em média um mesmo valor de 0.93 para todas as métricas (precisão, cobertura, F1 e acurácia). As medidas por classe são apresentadas na Tabela 2 a título de ilustração.

| Categoria | Precisão | Cobertura | F1 |
|---|---|---|---|
| alt.atheism | 0.95 | 0.96 | 0.96 |
| comp.graphics | 0.78 | 0.90 | 0.83 |
| comp.sys.ibm.pc.hardware | 0.92 | 0.95 | 0.93 |
| comp.windows.x | 0.98 | 0.81 | 0.88 |
| rec.autos | 1.00 | 0.94 | 0.97 |
| sci.med | 0.96 | 0.96 | 0.96 |
| sci.space | 0.96 | 0.96 | 0.96 |
| talk.politics.misc | 0.91 | 0.95 | 0.93 |
| Média | 0.93 | 0.93 | 0.93 |

***Tabela 2*** *– Precisão, cobertura e F1 por classe para classificador NB+EM, experimento com 8 classes*

## 4.4 Testes com Inclusão de Mais Categorias

Para garantir maior abrangência dos experimentos, realizamos experimentos variando o número de classes para 6, 15 e 20 classes. Diferentemente dos experimentos de Nigam, ignoramos uma possível dependência temporal entre os documentos e verificamos qual sua implicação nos resultados.

Reduzindo de 8 para 6 classes, a melhoria resultante do método NB+EM permaneceu presente, como ilustrado no gráfico da Figura 6 a seguir. Dada a similaridade dos gráficos das figuras 6 e 4, as análises realizadas na seção anterior ainda valem. Além disso, é importante observar que a melhoria de desempenho para 6 classes supera a melhoria para 8 classes, indicando uma possível tendência de melhoria em direção à classificação binária ($|\mathcal{C}| = 2$).

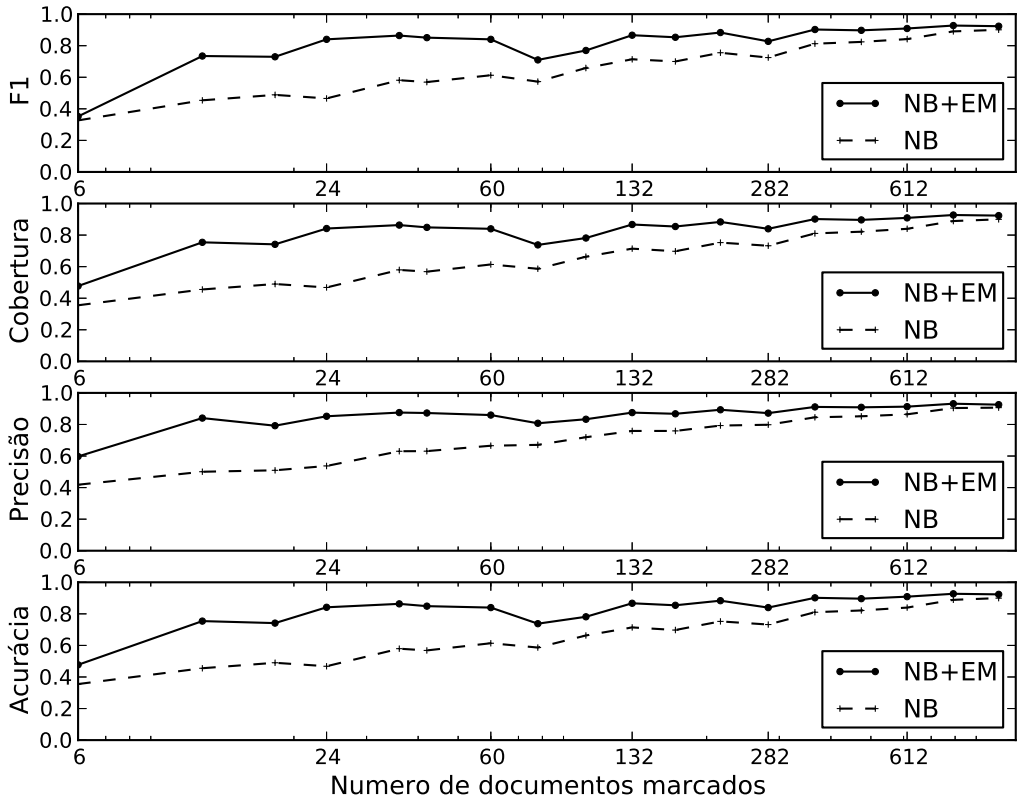


***Figura 6*** *– Métricas F1, Cobertura, Precisão e Acurácia de classificadores NB e NB+EM para 6 classes vs Número de documentos marcados (treinamento)*

Aumentando a quantidade de classes alvo para 15, obtemos os gráficos das métricas apresentados na Figura 7. Ao contrário da melhoria observada na redução de classes,

observamos que o aumento implica numa redução de desempenho quando comparado com o resultado anterior. Por outro lado, quando comparado com o NB tradicional, ainda é possível verificar uma melhora pelo menos na cobertura e na acurácia.

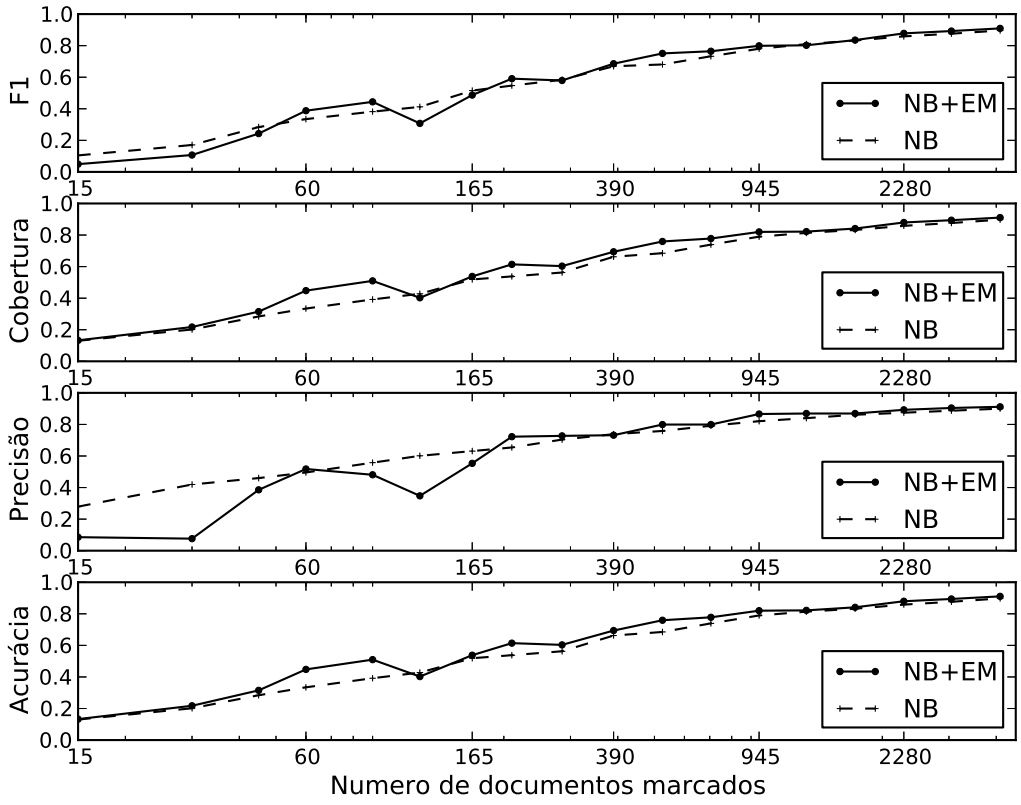


***Figura 7*** *– Métricas F1, Cobertura, Precisão e Acurácia de classificadores NB e NB+EM para 15 classes vs Número de documentos marcados (treinamento)*

Finalmente, para 20 classes, observamos degradações ainda maiores do que o experimento para 15 classes. De fato, 20 classes implica numa degradação completa em relação ao método NB tradicional. O gráfico da Figura 8 apresenta os resultados para o experimento com 20 classes.

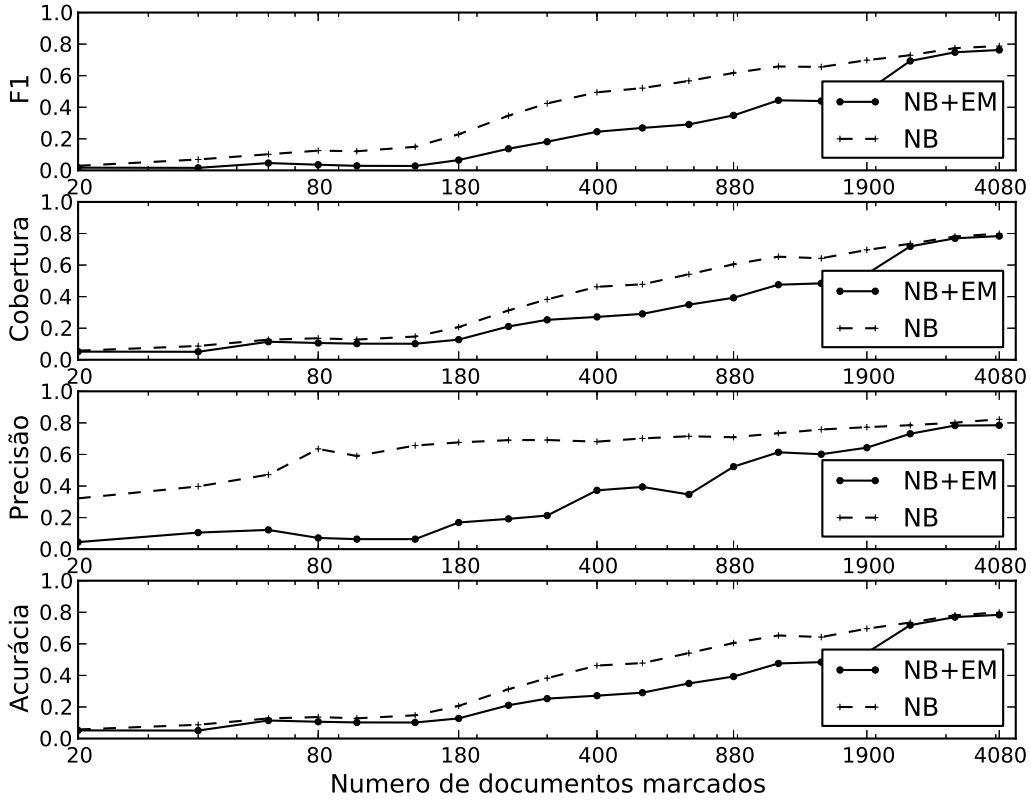


***Figura 8*** *– Métricas F1, Cobertura, Precisão e Acurácia de classificadores NB e NB+EM para 20 classes vs Número de documentos marcados (treinamento)*

# 5 Considerações Finais

Nesse trabalho foi apresentado um método semi-supervisionado de treinamento para classificador Bayes ingênuo que utiliza documentos não-marcados na melhoria de desempenho. Esse método, proposto inicialmente por Nigam, consiste na utilização do treinamento tradicional em conjunto com a técnica de Maximização de Expectativa. Ao longo do trabalho, foram apresentadas as bases probabilísticas nas quais o método se fomenta. Por fim, foram apresentados experimentos verificando a funcionalidade do método sob novas condições, não verificadas por Nigam em sua tese de doutorado.

Dos testes realizados, podemos concluir que o arranjo NB+EM de fato acarreta numa melhoria de classificação sob certas condições. Em seu trabalho, Nigam mostrou quando realizado o teste com documentos mais recentes (e consequentemente o treinamento com documentos mais antigos), o método apresenta melhorias significativas de acurácia e de BEP (*break-even point*), porém, não analisou detalhadamente as métricas de precisão, cobertura e F1.

Em nossos testes evitamos utilizar o critério de ordem cronológica, buscando verificar a efetividade do método sobre um caso mais geral.

No corpus utilizado, o método NB+EM apresentou uma melhoria de performance para uma baixa quantidade de classes alvo, quando comparado com o NB, principalmente quando poucos documentos são utilizados no treinamento. Tal resultado se mostra relevante, visto que o uso de menos documentos de treinamento efetivamente reduz o custo de construção do classificador.

Em nossos testes, verificamos uma superioridade do método NB+EM em relação ao NB para $|\mathcal{C}| \leq 15$. Em particular, foi constatado que quanto menos classes alvo, melhor o desempenho do método NB+EM.

Após o limiar de $|\mathcal{C}| > 15$ classes, o NB+EM apresentou performance menor ou igual ao tradicional NB. Durante a execução dos experimentos para $|\mathcal{C}| > 15$, observou-se empiricamente que algumas das classes tiveram todas as suas métricas anuladas (cobertura,

precisão), sendo esse possivelmente o motivo da redução das métricas globais do classificador.

Concluímos que o método NB+EM pode ser um método viável para problemas de CT de poucas classes, oferencendo um potencial aumento de desempenho. Sugere-se, entretanto, uma comparação prévia com pelo menos um classificador tradicional (e.g. NB, SVM) sob condições controladas, visto que os testes realizados no presente trabalho se restringiram a apenas um *corpus*.

Esse estágio apresentou imensuráveis ganhos para o autor na condição de aluno de graduação, visto que o colocou em contato com tecnologia de ponta empregada por toda a indústra ao redor do mundo e, além disso, tratou de um tópico de bastante interesse na comunidade acadêmica. Dada a gama de aplicações no mundo real, é possível afirmar que os conhecimentos adquiridos ajudarão em abrir muitas portas, sejam em pesquisa e desenvolvimento ou no mercado de trabalho.

## 5.1 Trabalhos Futuros

Mesmo tendo um breve período de execução, esse estágio permitiu o levantamento de novas ideias e possíveis temas para trabalhos futuros.

O leitor pode observar que as análises apresentadas nesse trabalho se restringiram ao classificador NB treinado com o método EM. Sendo assim, torna-se interessante um estudo da efetividade do método a outros classificadores (e.g. SVM, TSVM, kNN). Deseja-se também efetuar medições do método quando submetido a outros *corpora*, afim de assegurar uma certa segurança nos resultados.

Além disso, em sua tese de doutorado, Nigam apresentou extensões do método aqui apresentado, obtendo resultados surpreendentes. Um estudo detalhado dessas extensões pode constituir um interessante trabalho futuro, em que além do crescimento pessoal, existe a possibilidade de alterações e experimentação com o mesmo.

Ainda no âmbito do NB+EM, o autor gostaria de investigar sua efetividade em *corpora* de formas variadas, e.g. *corpus* com documentos predominantemente extensos ou curtos (e.g. mensagens do Twitter), *corpus* exclusivamente numéricos (e.g. diagnósticos de doenças), dentre outros. Uma outra possível variação seria a utilização de um algoritmo de seleção de características incorporado ao processo do EM, seja na estimativa de pertinência do conjunto de amostras não-marcadas ou no conjunto de amostras marcadas.

Uma outra sugestão é a introdução de um filtro na realimentação de estimativas do conjunto de documentos não-marcados, com intuito de balancear as distribuições de classes entre as predições para os documentos não-marcados, evitando que certas classes alcancem uma probabilidade a priori ($\hat{\theta}_{c_j}$) dominante.

# Referências


BLUM, A.; MITCHELL, T. Combining labeled and unlabeled data with co-training. In: **Proceedings of the eleventh annual conference on Computational learning theory**. New York, NY, USA: ACM, 1998. (COLT' 98), p. 92–100. ISBN 1-58113-057-0. Disponível em: <http://doi.acm.org/10.1145/279943.279962>.

DEMPSTER, A. P.; LAIRD, N. M.; RUBIN, D. B. Maximum likelihood from incomplete data via the em algorithm. **JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B**, v. 39, n. 1, p. 1–38, 1977.

JOACHIMS, T. **Text Categorization with Support Vector Machines: Learning with Many Relevant Features**. 1998.

JONES, E. et al. **SciPy: Open source scientific tools for Python**. 2001–. Disponível em: <http://www.scipy.org/>.

NIGAM, K. et al. Learning to classify text from labeled and unlabeled documents. In: **Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence**. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1998. (AAAI '98/IAAI '98), p. 792–799. ISBN 0-262-51098-7. Disponível em: <http://dl.acm.org/citation.cfm?id=295240.295806>.

NIGAM, K. P. **Using unlabeled data to improve text classification**. Tese (Doutorado), Pittsburgh, PA, USA, 2001. AAI3040487.

PEDREGOSA, F. et al. Scikit-learn: Machine Learning in Python . **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

RIJSBERGEN, C. J. V. **Information Retrieval**. 2nd. ed. Newton, MA, USA: Butterworth-Heinemann, 1979. ISBN 0408709294.

RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 2. ed. [S.l.]: Pearson Education, 2003. ISBN 0137903952.

SEBASTIANI, F. Machine learning in automated text categorization. **ACM Comput. Surv.**, ACM, New York, NY, USA, v. 34, n. 1, p. 1–47, mar. 2002. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/505282.505283>.

# APÊNDICE A – Código Fonte

```
# -*- coding: utf-8 -*-
# Author: Andre Dieb Martins <andre.dieb@gmail.com>

import sys

from time import time
from operator import itemgetter
from collections import Counter
from pprint import pprint

import cPickle as pickle

import numpy as np
import pylab as pl

from scipy.sparse import vstack, csr_matrix

from sklearn.datasets import fetch_20newsgroups, load_mlcomp
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import f1_score, recall_score, precision_score,
    classification_report, zero_one_score



categories = [
    'misc.forsale',
    'comp.graphics',
    'comp.os.ms-windows.misc',
    'comp.sys.ibm.pc.hardware',
    'comp.sys.mac.hardware',
    'comp.windows.x',
    'alt.atheism',
    'soc.religion.christian',
```

```
    'talk.religion.misc',
    'talk.politics.misc',
    'talk.politics.guns',
    'talk.politics.mideast',
    'rec.autos',
    'rec.motorcycles',
    'rec.sport.baseball',
    'rec.sport.hockey',
    'sci.crypt',
    'sci.electronics',
    'sci.med',
    'sci.space',
]
n_cat = len(categories)
datafile = 'nigam_nb_em.%d.out' % n_cat

# r1: percentage of labeled documents to use on training
# e.g. if r1=0.3, 30% of the documents will be used for training
r1 = 0.3

# r2-r1: percentage of unlabeled documents to use on NB+EM method
#  1-r2: percentage of labeled documents for testing
# e.g. if r1=0.3, r2=0.9, then 60% will be used as unlabeled for NB+EM
# and 10% will be used for testing.
r2 = 0.9


def benchmark_create(x_train, y_train, x_test, y_test, scores):
    """ Create a benchmark for the given train and test dataset. Store
        results
    on given "scores" dict.

    Returns a function fit_predict(estimator, name) that fits the train,
        predicts the test
    and stores the scores (f1, recall, precision and accuracy) under the
        given name.
    """
    def fit_predict(clf, name):
        clf.fit(x_train, y_train)
        prediction = clf.predict(x_test)
        scores['f1'].setdefault(name, []).append(f1_score(y_test,
            prediction))
```

```
71          scores['rec'].setdefault(name, []).append(recall_score(y_test,
                prediction))
            scores['prec'].setdefault(name, []).append(precision_score(y_test,
                prediction))
            scores['acc'].setdefault(name, []).append(zero_one_score(y_test,
                prediction))
            return prediction
        return fit_predict
76
    def score_avg(scoreset):
        """ Calculate average scores of f1, recall and precision over
        a set of scores, possibly output from iterations. Useful for
        cross-validation purposes.
81      """
        keys = scoreset[0]['f1'].keys()
        avg = {'f1': dict.fromkeys(keys, []),
            'prec': dict.fromkeys(keys, []),
            'rec': dict.fromkeys(keys, []),
86          'acc': dict.fromkeys(keys, [])}

        n_scores = float(len(scoreset))

        for clf in keys:
91          avg['f1'][clf] = sum([np.array(s['f1'][clf]) for s in scoreset])/
                n_scores
            avg['prec'][clf] = sum([np.array(s['prec'][clf]) for s in scoreset
                ])/n_scores
            avg['rec'][clf] = sum([np.array(s['rec'][clf]) for s in scoreset])/
                n_scores
            avg['acc'][clf] = sum([np.array(s['acc'][clf]) for s in scoreset])/
                n_scores

96      return avg


    def f1_rec_prec_acc(y_test, prediction):
        """ Computes (f1, recall, precision, accuracy) given the known labels
            and
101     the prediction.
        """
        return (f1_score(y_test, prediction),
                recall_score(y_test, prediction),
                precision_score(y_test, prediction),
```

```
106             zero_one_score(y_test, prediction))


    def is_score_better(score1, score2):
        """ Returns True if score1 has a higher average value over score2.
111     """
        diff = np.array(score1) - np.array(score2)
        return np.mean(diff) > 0.0


116 def normalize_class_freqs(num_docs, categories, x_train, y_train):
        """ Returns a training tuple (x_train, y_train) with an
        uniform distribution of classes (or categories). E.g. given
        (x_train, y_train), num_docs=100 and categories=['a', 'b','c', 'd'],
        returns a new (x_train_, y_train_) with at least 25 documents
121     of each class (100 docs/4 categories = 25 docs/cat).
        """
        n_cat = len(categories)
        docs_per_cat = np.floor(num_docs/float(n_cat))
        x_train_, y_train_ = [], []
126     added = 0

        print 'To train: %d documents, %d categories, %d docs/cat' % (num_docs,
            n_cat, docs_per_cat)
        print 'Sorting and selecting documents for training...'

131     for idx in range(x_train.shape[0]):
            stats = Counter(y_train_)

            x = x_train[idx]
            y = y_train[idx]
136
            if stats[y] >= docs_per_cat:
                continue

            x_train_.append(x.toarray()[0])
141         y_train_.append(y)

            # Only add up to num_docs
            added += 1
            if added == num_docs:
146             break
```

```
    print "Transforming matrix"
    y_train_ = np.array(y_train_)
    x_train_ = csr_matrix(x_train_)

    return x_train_, y_train_


class NigamNB(MultinomialNB):

    def complete_log_measure(self, x, y):
        """ Calculate complete log measure using the joint log likelihoods
        of each of the components.
        """
        total = 0.0

        clss = self.classes_.tolist()
        for cls in clss:
            jll = self._joint_log_likelihood(x)
            for idx in xrange(x.shape[0]):
                if y[idx] == cls:
                    total += jll[idx][clss.index(cls)]

        return total


# Output formatting

def h1():
    print '=' * 80
def h2():
    print '-' * 80
def h3():
    print '.' * 80


max_labeled = []


def procedure():
    global categories, r1, r2, max_labeled

    t0 = time()
    scores = {'f1': {}, 'prec': {}, 'rec': {}, 'acc': {}}
```

```
191
    # Load 20newsgroup-18828 from mlcomp
    data = load_mlcomp('20news-18828', categories=categories, shuffle=True)
    categories = data.target_names[:]

196 # Calculate proportions to split dataset
    div1 = int(np.floor(r1 * len(data.data)))
    div2 = int(np.floor(r2 * len(data.data)))

    print 'Spliting %d dataset into train=%d, unlabeled=%d, test=%d' % (len
        (data.data), div1, div2-div1, len(data.data) - div2)
201
    t1 = time()
    y_train = data.target[:div1]
    x_train_filenames = data.filenames[:div1]
    print 'Created training (%0.3fs)' % (time() - t1)
206
    t1 = time()
    y_unlabeled = data.target[div1:div2]
    x_unlabeled_filenames = data.filenames[div1:div2]
    print 'Created unlabeled (%0.3fs)' % (time() - t1)
211
    t1 = time()
    y_test = data.target[div2:-1]
    x_test_filenames = data.filenames[div2:-1]
    print 'Created test (%0.3fs)' % (time() - t1)
216
    print 'Finished separating data (%0.3fs)' % (time() - t0)

    print 'Categories: ', categories

221 print 'Vectorizing'
    t1 = time()
    vectorizer = CountVectorizer(stop_words='english', strip_accents='
        unicode',
         charset_error='replace')
    x_train = vectorizer.fit_transform((open(f).read() for f in
        x_train_filenames)).tocsr()
226 print 'Vectorized x_train'
    x_test = vectorizer.transform((open(f).read() for f in x_test_filenames
        )).tocsr()
    print 'Vectorized x_test'
```

```
    x_unlabeled = vectorizer.transform((open(f).read() for f in
        x_unlabeled_filenames)).tocsr()
    print 'Vectorized x_unlabeled'
    print 'Done (%0.3fs)' % (time() - t1)
    print 'Unique words:', len(vectorizer.vocabulary_)

    del x_train_filenames, x_test_filenames, x_unlabeled_filenames

    # Create a list of labeled counts to use on training
    indices = [int(np.floor(c)) for c in np.logspace(np.ceil(np.log2(n_cat)
        ), np.floor(np.log2(x_train.shape[0])), 20, base=2)]
    max_labeled = np.sort(list(set(indices)))

    # Ease late normalisation by making all elements in counts list
        multiple of the categories number
    for i in range(len(max_labeled)):
        while max_labeled[i] % n_cat != 0:
            max_labeled[i] -= 1
    # Remove duplicates and sort it ascending
    max_labeled = np.sort(list(set(max_labeled)))

    print 'Training support:', max_labeled

    for max in max_labeled:
        t2 = time()
        # Get a balanced training set with an equal amout of docs per
            category
        x_train_, y_train_ = normalize_class_freqs(max, categories, x_train
            , y_train)
        print 'Done (%0.3fs)' % (time() - t2)

        t2 = time()
        docs_per_cat = np.floor(max/float(len(categories)))
        # Make sure it's balanced
        print 'Statistics for training labels:', 'OK' if all([y ==
            docs_per_cat for y in Counter(y_train_).values()]) else 'Bad'

        benchmark = benchmark_create(x_train_, y_train_, x_test, y_test,
            scores)

        h1()
        print 'NB train=%d test=%d' % (x_train_.shape[0], x_test.shape[0])
        h2()
```

```
266         clf_nb = MultinomialNB()
            y_pred = benchmark(clf_nb, 'nb')

            print classification_report(y_test, y_pred, target_names=categories
                )
            print 'Accuracy=%.3f' % zero_one_score(y_test, y_pred)
271         h2()

            # Uncomment me if you want to compare with One-vs-Rest
                classification approach.
            # Tip: it will probably be better than usual NB but worse than NB+
                EM.
            # h1()
276         # print 'NB OVR train=%d test=%d' % (x_train_.shape[0], x_test.
                shape[0])
            # h2()

            # clf_nb_ovr = OneVsRestClassifier(MultinomialNB())
            # y_pred = benchmark(clf_nb_ovr, 'nb+ovr')
281
            # print classification_report(y_test, y_pred, target_names=
                categories)
            # print 'Accuracy=%.3f' % zero_one_score(y_test, y_pred)
            # h2()

286         h1()
            print 'NB+EM train=%d unlabeled=%d test=%d' % (x_train_.shape[0],
                x_unlabeled.shape[0], x_test.shape[0])
            h2()

            clf_nb_em = NigamNB()
291
            print 'Initializing NigamNB with %d labeled docs ...' % (x_train_.
                shape[0])

            # Initialize with labeled documents
            clf_nb_em.fit(x_train_, y_train_)
296
            # Calculate initial breaking condition
            last_ = clf_nb_em.complete_log_measure(x_train_, y_train_)
            print 'Starting goodness: %.3f' % last_
```

```
301         it = 0
            best_score = None

            # Append the unlabeled documents to the labeled ones
            x_train_new = csr_matrix(vstack([x_train_, x_unlabeled]))
306
            while True:
                y_unlabeled_pred = clf_nb_em.predict(x_unlabeled)

                h3()
311             print 'Loop iteration=%d' % it

                # Append the unlabeled predictions to the known labels
                y_train_new = y_train_.tolist()
                y_train_new.extend(y_unlabeled_pred.tolist())
316             y_train_new = np.array(y_train_new)

                print 'Fitting new classifier with joint set Labeled+Unlabeled
                    ..'
                clf_nb_em_test = NigamNB()
                clf_nb_em_test.fit(x_train_new, y_train_new)
321
                print 'Calculating goodness..'
                cur_score = f1_rec_prec_acc(y_test, clf_nb_em_test.predict(
                    x_test))
                new_ = clf_nb_em_test.complete_log_measure(x_train_new,
                    y_train_new)
                change = new_ - last_
326
                print 'Goodness scores: acc=%0.3f f1=%.3f rec=%.3f prec=%.3f' %
                     cur_score
                print 'JLL score: %s (change=%.3f)' % ('GOOD' if change >= 0.0
                    else 'BAD', change)

                if best_score is None:
331                 best_score = cur_score
                elif not is_score_better(cur_score, best_score):
                    print 'New score is worse than last..'
                    # FIXME: empirically we discovered that bailing on the
                        first bad score
                    # is a good approach, since the maximization of the log
                        probability
```

```
336                 # not always lead to the maximization of the metrics (Rec.,
                        Prec., Acc.).

                    # This means that bailing on a bad score is a good aproach
                        for achieving
                    # greater performance. Uncomment the following line for
                        that.
                    #break
341
                clf_nb_em = clf_nb_em_test

                # Bail when change on log prob. is small enough. Tweaking this
                    threshold may
                # lead to better results.
346             if np.abs(change) < 0.1:
                    print 'Change on JLL is enough, bye'
                    break

                last_ = new_
351             it += 1

            scores['f1'].setdefault('nb+em', []).append(cur_score[0])
            scores['rec'].setdefault('nb+em', []).append(cur_score[1])
            scores['prec'].setdefault('nb+em', []).append(cur_score[2])
356         scores['acc'].setdefault('nb+em', []).append(cur_score[3])

            print
            print 'Final goodness score: acc=%0.3f f1=%.3f rec=%.3f prec=%.3f'
                % cur_score
            print classification_report(y_test, clf_nb_em.predict(x_test),
                target_names=categories)
361
            h2()

            print 'Tests for k=%d took %0.3fs' % (max, time() - t2)

366     print 'All tests took %0.3fs' % (time() - t0)
        return scores


    if len(sys.argv) != 2:
371     # Repeat the procedure 5 times (5-fold)
        scores = [procedure() for i in range(5)]
```

```
        # Compute the scores' average
        scores = score_avg(scores)

        print 'Saving to file %s' % datafile
        pickle.dump({'scores': scores, 'features': max_labeled}, open(datafile,
            'w+'))
    elif sys.argv[1] == '--last':
        data = pickle.load(open(datafile, 'r'))
        scores = data['scores']
        max_labeled = data['features']


    font = {'size': 12}
    pl.rc('font', **font)
    pl.figure()
    pl.grid(True)
    pl.title('F1, Recall & Precision')

    f1 = pl.subplot(411)
    rec = pl.subplot(412)
    prec = pl.subplot(413)
    acc = pl.subplot(414)


    def setup_graph(g):
        g.legend(loc='lower right', prop=font)
        g.set_xlabel('Numero de documentos marcados')
        g.tick_params(axis='both', which='major', labelsize=10)
        g.tick_params(axis='both', which='minor', labelsize=7)
        g.xaxis.set_major_formatter(pl.FuncFormatter(lambda x, pos: '%1i' % x))
        g.set_xticks([m for m in max_labeled if max_labeled.tolist().index(m) %
            3 == 0])
        g.set_xlim(xmin=0, xmax=max(max_labeled)*1.1)
        g.set_ylim(ymin=0.0, ymax=1.0)

    plotargs = {'markersize': 3}

    f1.set_ylabel('F1')
    f1.semilogx(max_labeled, scores['f1']['nb+em'], '-ok', label='NB+EM', **
        plotargs)
    f1.semilogx(max_labeled, scores['f1']['nb'], '--+k', label='NB', **plotargs
        )
```

```
    #f1.grid(True)

    rec.set_ylabel('Cobertura')
    rec.semilogx(max_labeled, scores['rec']['nb+em'], '-ok', label='NB+EM', **
        plotargs)
    rec.semilogx(max_labeled, scores['rec']['nb'], '--+k', label='NB', **
        plotargs)
    #rec.grid(True)

    prec.set_ylabel(u'Precisao')
    prec.semilogx(max_labeled, scores['prec']['nb+em'], '-ok', label='NB+EM',
        **plotargs)
    prec.semilogx(max_labeled, scores['prec']['nb'], '--+k', label='NB', **
        plotargs)
    #prec.grid(True)

    acc.set_ylabel(u'Acuracia')
    acc.semilogx(max_labeled, scores['acc']['nb+em'], '-ok', label='NB+EM', **
        plotargs)
    acc.semilogx(max_labeled, scores['acc']['nb'], '--+k', label='NB', **
        plotargs)
    #acc.grid(True)

    setup_graph(f1)
    setup_graph(rec)
    setup_graph(prec)
    setup_graph(acc)

    pl.show()

    pl.figure()
    pl.xlabel('Numero de documentos marcados')
    pl.ylabel(u'Acuracia')
    pl.semilogx(max_labeled, scores['acc']['nb+em'], '-ok', label='NB+EM', **
        plotargs)
    pl.semilogx(max_labeled, scores['acc']['nb'], '--+k', label='NB', **
        plotargs)
    pl.legend(loc='lower right', prop=font)
    pl.tick_params(axis='both', which='major', labelsize=10)
    pl.tick_params(axis='both', which='minor', labelsize=7)

    pl.gca().xaxis.set_major_formatter(pl.FuncFormatter(lambda x, pos: '%1i' %
        x))
```

```
446 pl.xticks([m for m in max_labeled if max_labeled.tolist().index(m) % 3 ==
        0])
    pl.xlim(xmin=0, xmax=max(max_labeled)*1.1)
    pl.ylim(ymin=0.0, ymax=1.0)

    pl.show()
```