

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



FEUP

Sugestões de Rotas Turísticas Baseadas no Perfil do Utilizador

João André Correia Rodrigues

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Doutor António Fernando Vasconcelos Cunha Castro Coelho (Professor
Auxiliar)

17 de Junho de 2011

Sugestões de Rotas Turísticas Baseadas no Perfil do Utilizador

João André Correia Rodrigues

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo júri:

Presidente: Doutor Jorge Alves Silva (Professor Auxiliar)

Vogal Externo: Doutor João Paulo Moura (Professor Auxiliar)

Orientador: Doutor António Fernando Vasconcelos Cunha Castro Coelho (Professor Auxiliar)

17 de Junho de 2011

Resumo

A evolução dos sites turísticos está a convergir para um conjunto de funcionalidades e melhores práticas que se estão a tornar norma, de tal forma que começa a ser viável o desenvolvimento de uma plataforma genérica adaptável a qualquer região turística. Um dos objectivos mais importantes deste género de sites é mostrar ao turista sugestões daquilo que pode encontrar no destino, idealmente de forma personalizada. Este pode ser considerado um caso particular da selecção de dados relevantes para um utilizador em situações de excesso de informação.

Contudo, um problema em implementar esta funcionalidade é a inexistência de informação de utilização que seja adequada à realização de *data mining* quando um novo site é inaugurado. Esta dissertação propõe uma abordagem que conjuga um conjunto de técnicas, aplicadas ao caso de estudo da Região do Douro, que vão desde tirar partido dos dados disponíveis no Flickr, cruzando-os com uma base de dados de pontos de interesse e utilizando a Google Prediction API para gerar sugestões de visita personalizadas, até à análise do itinerário geográfico que o utilizador definiu, para escolha de sugestões com base na proximidade, recorrendo a funcionalidades da área dos sistemas de informação geográficos. Fica demonstrado que é possível construir um modelo que consegue ser mais flexível e adaptável que anteriores abordagens ao problema.

Abstract

The evolution of tourism websites is converging to a set of features and best practices that are becoming standard, in such a way that developing a generic platform, customizable to any tourist region, is becoming feasible. A very important feature in this kind of site is showing the tourist suggestions of what can be seen or visited at the destination, optimally in a personalized way. This can be thought of as a special case of relevant data selection for a user in information overload scenarios.

However, one problem in deploying such functionality is the lack of user information suited to perform data mining when a new site is brought online. This thesis proposes an approach, applied to the Douro Region case study, which combines a set of techniques, ranging from harnessing the information available in Flickr, crossing it with a points of interest database and using Google Prediction API to generate personalized travel suggestions, to the analysis of the itinerary the user defined, in order to come up with geographically nearby suggestions. A more flexible and adaptable model emerges, with further possibilities than previous approaches.

Agradecimentos

Gostava de agradecer a toda a gente que contribuiu para que o esforço de escrever esta tese tenha atingido o seu objectivo, em particular à São, que sempre esteve do meu lado, aos meus pais e aos meus amigos.

Também aos professores desta casa, que ensinaram muito do que sei, mesmo que isso tenha tido lugar há uma década atrás. Excepção feita para o Prof. António Coelho que, *nos dias de hoje*, desempenhou exemplarmente o papel de orientador, além de ter feito a sugestão original da realização desta dissertação.

E adicionalmente aos meus colegas na Equipa Portal Douro no INESC Porto, Leonel Dias, Hélder Nunes, Lino Oliveira, Prof. Mário Jorge Leitão e Prof. José Manuel Oliveira, devido à enorme dedicação que devotam ao projecto; sei que é sempre possível contar com eles!

A todos, muito obrigado. . .

João André Correia Rodrigues

Conteúdo

1	Introdução	1
1.1	Enquadramento e Motivação	1
1.2	Descrição do Problema	2
1.3	Objectivos	4
1.4	Trabalho Relacionado	5
1.5	Estrutura do Documento	7
2	Revisão Bibliográfica	9
2.1	Concepção de Sites Turísticos	9
2.2	Ferramentas de <i>Trip Planning</i>	10
2.3	Sistemas de Recomendação	11
2.4	Filtragem Colaborativa	12
2.5	Motores de Sugestão Aplicados ao <i>E-Tourism</i>	13
2.6	Conclusão	13
3	Metodologia	15
3.1	Mecanismos de Sugestão Estáticos e Dinâmicos	15
3.2	Integração no Portal e Interface de Comunicação	16
3.3	Modo de Implementação e Teste	17
3.4	Conclusão	19
4	Mecanismo de Sugestões Estático	21
4.1	Classificação dos POI com Base nas Fotos Disponíveis no Flickr	21
4.2	Google Places	29
4.3	Classificação Editorial	31
4.4	Classificação dos Utilizadores	31
4.5	Conclusão	32
5	Mecanismo de Sugestões Dinâmico	33
5.1	Sugestão Aleatória de Pontos de Interesse	33
5.2	Restrição das Sugestões à Proximidade do Itinerário do Utilizador	35
5.3	Prioritização dos POI Mais Próximos do Itinerário do Utilizador	37
5.4	Consideração do Perfil Explícito do Utilizador	37
5.5	Consideração do Perfil Implícito do Utilizador	39
5.6	Incorporação do Mecanismo de Sugestões Estático	43
5.7	Consideração das Distâncias em Estrada	45
5.8	Conclusão	47

CONTEÚDO

6	Conclusões e Trabalho Futuro	49
	Referências	53
A	Listagens e Tabelas de Apoio	57
A.1	Tabela Flickr	57
A.2	Flickr Dump	57
A.3	Qualidade das Associações POI/Flickr	59
A.4	Parametrizações do sistema	61
A.5	Associações entre fotografias e POI	62
A.6	Google Places Dump	62
A.7	Tabela Classificações	67
A.8	Versão 1: Pontos Aleatórios	67
A.9	Versão 2: Na Proximidade do Itinerário	69
A.10	Versão 3: Distância como Factor de Escolha	71
A.11	Versão 4: Perfil Explicitamente Definido pelo Utilizador	75
A.12	Geração do Ficheiro de Treino	78
A.13	Versão 5: Consideração do Perfil Implícito do Utilizador	80
A.14	Versão 6: Incorporação dos Componentes de Sugestão Estáticos	86
A.15	Tabela <i>Cache</i> das Distâncias em Estradas	91
A.16	Versão 7: Consideração das Distâncias em Estrada	92

Lista de Figuras

1.1	A vista clássica alfanumérica.	3
1.2	A aplicação geográfica.	4
1.3	<i>Mockup</i> de parte da aplicação geográfica onde é possível ver em que contexto deverão surgir as sugestões.	5
2.1	Sugestão personalizada de um percurso na Pinacoteca do Vaticano, utilizando metodologias clássicas.	12
2.2	Sugestão personalizada, envolvendo agora itens serendipiosos.	12
3.1	Casos de uso relevantes para a geração de recomendações.	16
3.2	Parte do modelo de componentes, mostrando o contexto onde o motor de sugestões a desenvolver se enquadra.	17
3.3	Aplicação para testar a DLL com a lógica de recomendação.	18
4.1	Modelo de dados, extraído utilizando o Visio, da base de dados SQL Server utilizada pelo portal, simplificado pela omissão da indicação das colunas das tabelas.	22
4.2	Pormenor do modelo de dados modificado, onde é possível observar a tabela destinada a receber informação do site Flickr.	23
4.3	A acção da <i>query</i> de limpeza apresentada na Listagem 4.1.	23
4.4	Relação entre o acumulado de associações correctas e a distância entre cada foto e o seu POI respectivo.	27
4.5	Pormenor da interface do <i>Google Places</i> , onde é possível classificar um local.	29
4.6	Pormenor do modelo de dados, após receber a tabela destinada a guardar as classificações dos utilizadores.	32
5.1	Visualização de cinco POI escolhidos aleatoriamente.	34
5.2	Visualização do resultado, com a zona criada pelo método <code>STBuffer()</code> , assinalada a cinzento.	36
5.3	Reconstrução de 115 percursos, cada um com uma tonalidade diferente, a partir dos dados disponíveis no site Flickr.	39
5.4	Pormenor do modelo de dados modificado, onde é possível observar a tabela destinada a receber a <i>cache</i> das distâncias em estrada.	46
5.5	Um percurso original (linha sólida) e o desvio (a tracejado) necessário para visitar a sugestão s_1	46

LISTA DE FIGURAS

Lista de Tabelas

2.1	Os factores que contribuem para a permanência de visitantes, organizados por Beldona e Cai [BC06] em três categorias.	10
2.2	Os factores que contribuem para a permanência de visitantes no caso específico do enoturismo, segundo Hall [Hal00].	10
5.1	Cinco POI escolhidos aleatoriamente.	34
5.2	Um resultado obtido, utilizando o método <code>STBuffer()</code>	36
5.3	Resumo apresentado no final de uma sessão de treino.	41
5.4	Saída de uma predição.	42
5.5	Pesos utilizados para os diversos componentes.	43
A.1	As associações entre fotos e POI com as menores distâncias.	63

LISTA DE TABELAS

Listagens de Código

4.1	A eliminação das fotografias não pertencentes à Região do Douro.	23
4.2	A CTE <code>CoordsQnt</code> descobre as coordenadas e o número de vezes que estas se repetem, caso este valor seja igual ou superior a seis; posteriormente são eliminados os registos relativos às fotos que tenham as coordenadas descobertas.	24
4.3	Vista que exclui os POI sem interesse turístico.	25
4.4	Uma <i>query</i> que associa a cada fotografia o seu POI mais próximo e respectiva distância.	26
4.5	Uma <i>query</i> que prepara a tabela <code>DOUROREGION.Poi</code> para receber a informação relativa à <i>popularidade fotográfica</i>	28
4.6	Uma <i>query</i> que associa a cada POI a proporção de fotos que tem associadas, relativamente ao ponto de interesse com mais fotos associadas. . . .	28
4.7	Segunda versão da vista exclui os POI sem interesse turístico, agora exibindo também a coluna <code>flickr</code>	28
4.8	Terceira versão da vista exclui os POI sem interesse turístico, agora exibindo também a coluna <code>places</code>	30
5.1	Código que obtém aleatoriamente cinco POI.	33
5.2	Introdução de uma zona de restrição à selecção dos POI.	35
5.3	A utilização da distância ao percurso como factor de escolha do POI a sugerir.	37
5.4	A utilização das preferências do utilizador relativamente a pilares como factor de escolha do POI a sugerir.	38
5.5	Extracção dos valores da base de dados.	44
5.6	Consideração dos valores no <i>ranking</i>	44

LISTAGENS DE CÓDIGO

Abreviaturas e Símbolos

API	Application Programming Interface
CCDR-N	Comissão de Coordenação e Desenvolvimento Regional do Norte
CITMAD	Centro de Inovação de Trás-os-Montes e Alto Douro
CMS	Content Management System
CSV	Comma Separated Values
CTE	Common Table Expression
DLL	Dynamic Link Library
FEP	Faculdade de Economia da Universidade do Porto
FEUP	Faculdade de Engenharia da Universidade do Porto
GPS	Global Positioning System
IIS	Internet Information Services
IGESPAR	Instituto de Gestão do Património Arquitectónico e Arqueológico
INESC	Instituto de Engenharia de Sistemas e Computadores
IPA	Instituto Português de Arqueologia
IPPAR	Instituto Português do Património Arquitectónico
JSON	JavaScript Object Notation
NIPC	Número de Identificação de Pessoa Colectiva
OCG	Open Geospatial Consortium
PHP	PHP: Hypertext Preprocessor
PIB	Produto Interno Bruto
POI	Point of Interest
REST	Representational State Transfer
RFP	Request for Proposal
SQL	Structured Query Language
UML	Unified Modeling Language
UNESCO	United Nations Educational, Scientific and Cultural Organization
URI	Uniform Resource Identifier
UTAD	Universidade de Trás-os-Montes e Alto Douro
WPF	Windows Presentation Foundation

ABREVIATURAS E SÍMBOLOS

Capítulo 1

Introdução

1.1 Enquadramento e Motivação

Os websites turísticos começam a diferenciar-se dos demais pelas funcionalidades que oferecem ao turista. Informação sobre a forma de chegar, o que fazer, sugestões sobre o que ver, onde comer, onde ficar, mapas do local, fotografias, vídeos, planeamento de uma ida, ... começam a tornar-se norma para todos os sites turísticos. No entanto, ao contrário de certas áreas, que contam com plataformas já desenvolvidas (a educativa com o Moodle, a gestão de projectos com o Redmine, ...) e que somente necessitam de alguma personalização, o mesmo não acontece com o turismo. Cada região precisa de implementar de raiz o mesmo conjunto de funcionalidades, não havendo reutilização de componentes, e não tirando partido dos testes, resoluções de problemas e inovações que têm lugar nos sites de outras regiões. Seria possível utilizar CMS como Drupal, Joomla ou Plone, mas tratam-se de sistemas demasiado genéricos, sobre os quais há necessidade de realizar extensos desenvolvimentos; há em particular a ausência de componente geográfica associada aos conteúdos nestas plataformas.

O ideal seria assim a existência de uma plataforma específica para portais turísticos, mas suficientemente genérica para poder ser adaptada a qualquer região.

Concretamente em Portugal, o turismo constitui um dos maiores sectores da economia; de acordo com o *World Travel & Tourism Council* [Wor10], em 2010 contribuiu directa e indirectamente para 14,4% do PIB e gerou emprego para 18,8% da população activa. Num estudo da Brandia Central [Bra09], o caso do Douro é analisado em particular, através de sondagens com vista a determinar a percepção que o cidadão comum tem da região. Os resultados apresentados revelam que, embora o seu índice de *atractividade* seja superior à média nacional e se qualifique em primeiro lugar na componente *paisagem natural*, acaba por ficar aquém destes valores no que diz respeito ao *património histórico*

(4º lugar) e à *oferta cultural e social* (6º lugar), entre outros. É neste contexto e perante a inexistência de um site turístico oficial dedicado à Região do Douro, que a CCDDR-N, através da “ON.2 – O Novo Norte” (Programa Operacional Regional do Norte), decide reunir um conjunto de parceiros (INESC Porto, FEUP, FEP, UTAD e CITMAD) com vista à sua implementação, surgindo assim a oportunidade de iniciar um processo de concepção e desenvolvimento com o objectivo de criar uma plataforma turística genérica, com um caso de estudo concreto ao qual possa ser aplicada.

Após uma análise exaustiva do estado da arte na área dos portais turísticos, começaram a ser definidos os objectivos para o projecto, apontando para uma forte componente multi-média, que vá mais longe que um simples conjunto de textos e fotos, passando ainda pelo envolvimento dos utilizadores, câmaras municipais, museus e quintas da região, procurando tirar partido das tecnologias web mais recentes (HTML5, CSS3, AJAX, jQuery, ...) e suportando integração com redes sociais; por último, mas não menos importante, uma total georreferenciação dos conteúdos, proporcionando contextualização geográfica e planificação de viagens, incorporando um mecanismo de geração de sugestões.

Relativamente à estrutura do site, prevê o cruzamento de dois eixos, pilares e actividades, a partir dos quais é possível explorar todos os POI, eventos, notícias, fotos, vídeos, ... disponibilizados. No caso de estudo os primeiros são “Turismo”, “Paisagem”, “Vinho” e “Cultura & Património”; no entanto estas escolhas dependem da região para a qual a plataforma é personalizada; numa zona litoral com ondas, poderia haver um pilar “Surf”, por exemplo. Por outro lado as actividades são “Explorar”, “Viver”, “Viajar” e “Ficar & Degustar”; mais uma vez pode haver necessidade de alterar estas escolhas noutra região.

Como ponto de partida, foi seleccionado um CMS extensamente utilizado por sites autárquicos, uma finalidade relativamente próxima da turística. Mais detalhes sobre o processo de concepção e desenvolvimento podem ser encontrados em [ORN⁺11].

Ainda no âmbito deste projecto existe o Mobidouro, que pretende estender o portal para dispositivos móveis e consequentemente conseguir tirar partido das capacidades GPS para seleccionar publicidade de estabelecimentos nas proximidades do utilizador e que se adequem ao seu perfil [CD11].

1.2 Descrição do Problema

A organização planeada para a plataforma passa pela existência de dois modos de funcionamento: uma vista clássica alfanumérica e uma inovadora aplicação geográfica. Mesmo durante a navegação no modo clássico, um pequeno mapa está sempre presente, proporcionando contextualização geográfica do item que estiver a ser visualizado, como pode ser visto na Figura 1.1(a).

Introdução



Figura 1.1: A vista clássica alfanumérica.

Acontece que o pequeno mapa acaba por ser a aplicação geográfica, mas num modo minimalista e podendo, através de um clique, ser maximizada, como pode ser visto na Figura 1.2. A qualquer altura está também disponível a possibilidade de adicionar, utilizando o botão apresentado na Figura 1.1(b) o item visualizado, seja um castelo, uma quinta ou um restaurante, à colecção “A Minha Viagem”, empregando o *design pattern* que Anders Toxboe [Tox] denomina por *carrinho de compras*; no modo maximizado, este botão é reduzido a um pequeno ícone de uma mala de viagem, à direita no nome de cada POI.

A aplicação geográfica disponibiliza então um módulo de planeamento de viagens no painel “A Minha Viagem” que permite gerir o percurso, sendo possível a reordenação dos pontos de passagem seleccionados, a sua eliminação ou a adição de itinerários pré-concebidos (a Rota do Vinho do Porto ou Rota das Amendoeiras em Flor, por exemplo).

É precisamente neste módulo que está prevista a existência de um painel apresentando sugestões, de acordo com o *mockup* apresentado na Figura 1.3. Surge assim o problema

Introdução

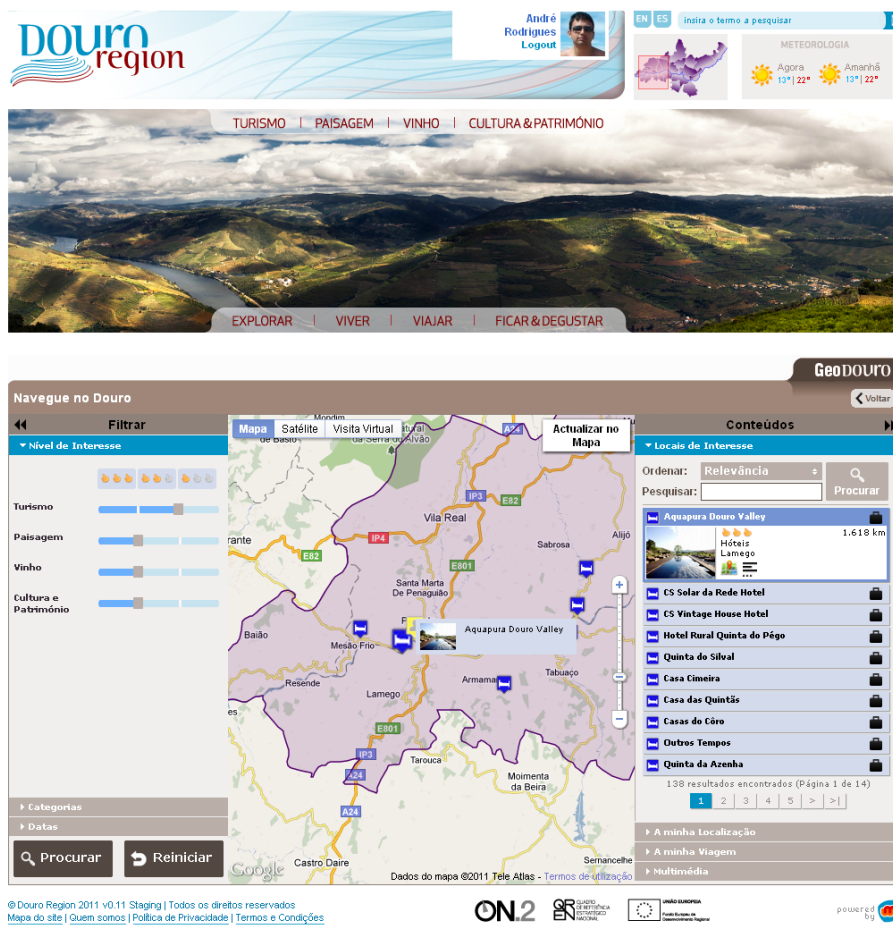


Figura 1.2: A aplicação geográfica.

de como seleccionar sugestões a serem apresentadas, tendo em conta os POI já escolhidos pelo utilizador (as suas categorias e localizações), as preferências declaradas (exprimíveis através dos *sliders* visíveis na Figura 1.2) e a relevância turística das várias possibilidades consideradas.

1.3 Objectivos

Como objectivo geral, pretende-se desenvolver um módulo de geração de sugestões adaptável a qualquer personalização da plataforma genérica a uma região turística. Para tal, será necessário concretizar os seguintes objectivos específicos:

- ter em conta o interesse turístico de cada POI, ou seja, considerar a classificação editorialmente atribuída e dar prioridade àqueles que possuam valores mais elevados;

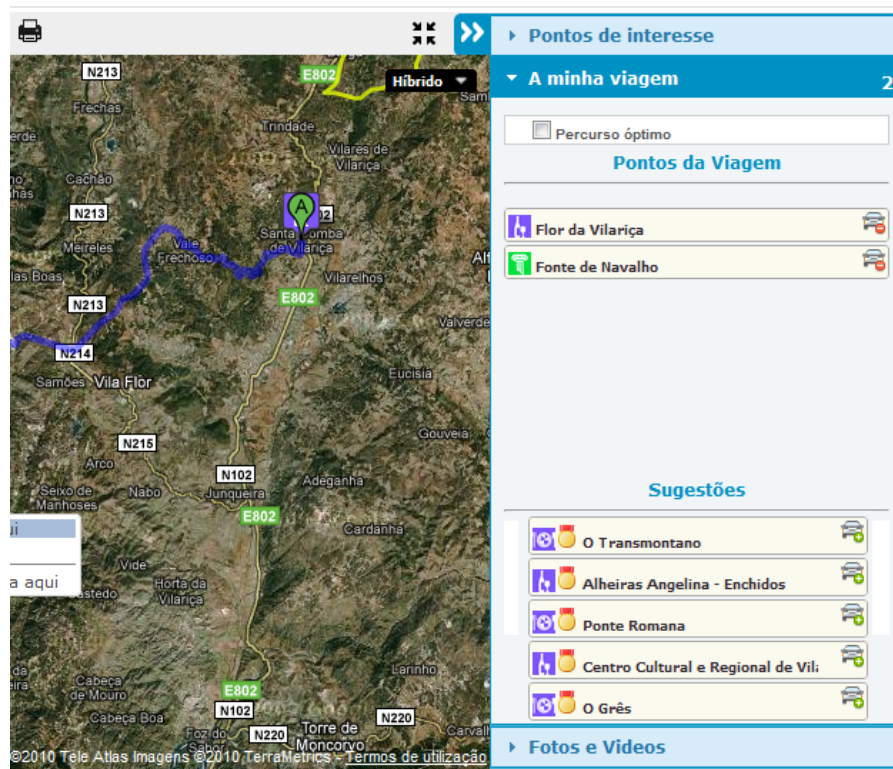


Figura 1.3: *Mockup* de parte da aplicação geográfica onde é possível ver em que contexto deverão surgir as sugestões.

- ter em conta a distância entre os pontos sugeridos e o itinerário que o utilizador já tenha planeado, ou seja, considerar somente os POI num raio de n quilómetros do percurso e/ou dar mais prioridade aos mais próximos em detrimento dos mais afastados;
- ter em conta o perfil do utilizador, tirando partido da indicação que explicitamente é feita dos seus gostos;
- inferir o perfil do utilizador com base nos POI por ele escolhidos para “A Minha Viagem” e considerá-lo para a sugestão de pontos de interesse;
- criar as condições para que, a partir do momento que o site fique online, comecem a ser recolhidos os dados necessários para a indicação de sugestões com base em filtragem colaborativa assente no comportamento dos restantes utilizadores.

1.4 Trabalho Relacionado

Actualmente a problemática do excesso de informação está presente em muitas áreas, o que tem levado ao desenvolvimento de metodologias que permitam seleccionar quais

os dados mais relevantes num certo contexto. Um exemplo particular deste problema é a escolha do que ver quando se realiza uma visita a uma determinada região.

Talvez a solução mais conhecida do problema geral seja a adoptada pelo site da *Amazon*, com a sua funcionalidade *Customers Who Bought This Item Also Bought...*, que assume que o utilizador pode estar interessado em itens relacionados com aquele que está a visualizar, com base na análise do padrão de compras dos restantes clientes [LJB01]. Embora não precise da construção de um perfil através da classificação de vários itens, é baseado somente em um item, daí que, se aplicado no contexto de um *trip planner*, não consideraria o conjunto dos POI seleccionados como um todo. Poderia somente gerar uma sugestão para cada ponto de interesse do itinerário.

É também de salientar o concurso lançado pelo site *Netflix*, destinado a premiar com um milhão de dólares a melhoria mais substancial ao seu algoritmo de recomendação de filmes *Cinematch*, cujo vencedor foi a equipa *BellKor's Pragmatic Chaos*, constituída por investigadores dos laboratórios *AT&T Labs*, *Commendo Research & Consulting GmbH* [TJB09], *Pragmatic Theory* [PC09] e *Yahoo! Research Israel* [Kor09].

Uma das bases do sucesso do motor de busca *Google* é o algoritmo *PageRank*, desenvolvido por Larry Page na Universidade de Stanford [PBMW99], que determina a relevância dos resultados e a sua consequente ordenação, acabando por ser um motor de sugestões de páginas para um determinado termo de pesquisa. Inicialmente só considerando o grafo de links entre páginas web, recentemente começou a ter em conta também o perfil inferido do utilizador, com base no histórico do seu comportamento.

No caso específico do *e-tourism*, há que considerar a componente geográfica ao determinar quais as sugestões que irão ser fornecidas ao utilizador. Por exemplo, o sistema *LoL@*, desenvolvido pelo *Telecommunications Research Center Vienna* [KMPA03], funcionando num dispositivo móvel, exhibe os locais de interesse na proximidade do utilizador; adicionalmente, uma das funcionalidades com implementação planeada é a filtragem destes pontos de acordo com o perfil e comportamento do próprio utilizador e de outros.

Uma equipa de investigadores da Universidade de Ciência e Tecnologia de Hong Kong e da *Microsoft Research Asia* [ZZXY10] utilizam uma abordagem baseada no histórico de actividades realizadas e localizações registadas; de seguida procede-se ao *data mining* sobre bases de dados e a web, com vista à extracção de sugestões de locais a visitar e actividades a realizar. Destaque-se que as referidas bases de dados são construídas com registos de trajectos GPS realizados por turistas ao longo de dois anos e meio.

A disponibilização de máquinas fotográficas com capacidade de *geotagging* levou a que sites de alojamento de fotografia disponibilizem informação extra relativa à localização. A tentativa de tirar partido desta informação deu origem a trabalhos relevantes, como o de uma equipa da TU Delft [CSdVR10], que baseou as sugestões de pontos de interesse numa cidade a visitar por um utilizador nos *geotags* de fotos que tirou noutras cidades, para isso recorrendo à API que o *Flickr* disponibiliza.

1.5 Estrutura do Documento

Este documento, para além deste capítulo introdutório onde o problema que será abordado é contextualizado, contém mais cinco capítulos e um apêndice.

No Capítulo 2 é apresentada uma pesquisa ao estado da arte nas áreas da concepção de sites turísticos, ferramentas de *trip planning*, sistemas de recomendação, filtragem colaborativa e motores de sugestão aplicados ao *e-tourism*; depois, no Capítulo 3, a metodologia proposta é exposta; de seguida, os Capítulos 4 e 5 relatam a implementação do projecto, debruçando-se respectivamente sobre o mecanismo de sugestões estático e dinâmico. Por último, são apresentadas as conclusões e o trabalho futuro no Capítulo 6. Existe também um Apêndice A com detalhes técnicos que vão sendo referidos ao longo do texto.

Introdução

Capítulo 2

Revisão Bibliográfica

O trabalho desenvolvido no âmbito desta dissertação acaba por envolver uma série de áreas, que vão desde a concepção de uma categoria muito específica de sites (os portais turísticos) até aplicações de *data mining*. Após no capítulo anterior terem sido mencionados trabalhos muito próximos deste, neste pretende-se fornecer uma visão global do estado da arte das várias áreas tocadas por este projecto.

2.1 Concepção de Sites Turísticos

De entre a enorme variedade de sites existentes por toda a Web, os de cariz turístico conseguem distinguir-se dos restantes e partilhar um conjunto de características próprias. Mesmo de pontos de vista relativamente afastados do da informática, como o da linguística, é possível encontrar peculiaridades nos portais turísticos e nos guias em papel (Lonely Planet, American Express, Michelin, Routard, . . .), o que até permite o aparecimento de sites que parodiam este género, fazendo uso dos mesmos estilos [HKW10].

Como veículos de promoção de um determinado destino, os sites turísticos acabam por estar limitados pela sua intangibilidade: enquanto na Web é possível apresentar 30 segundos de uma música ou um capítulo amostra de um livro, o mesmo não é possível relativamente a um país ou região. Mesmo assim, Beldona e Cai [BC06], no contexto dos portais para o turismo rural, consideram que estes acabam por constituir o melhor veículo de promoção, graças à riqueza com que a mensagem consegue ser transmitida.

Estes autores adicionalmente organizam os factores que contribuem para a permanência de visitantes num site turístico em três categorias: *conteúdo*, *interactividade* e *valor promocional*¹, como pode ser visto na Tabela 2.1.

¹Embora atribuam esta classificação a Gillespie et al. referindo um texto que deixou de estar disponível.

Conteúdo	Interactividade	Valor Promocional
Alojamento	<i>Media RFP</i>	Ofertas Online
Bares e Restaurantes	<i>Meeting Planners RFP</i>	Newsletters
Atrações Turísticas	Reservas de Alojamento	<i>E-Cards</i>
Artes Performativas	Prendas Online	
Transportes	Avaliação do Serviço	
Compras	Registo no site	
Desporto e Recreação	Pesquisa	
Mapas	Ferramentas de <i>Trip Planning</i>	
<i>Press Releases</i>		
Calendário de Eventos		
Facilidades para Congressos		
Informação Turística		

Tabela 2.1: Os factores que contribuem para a permanência de visitantes, organizados por Beldona e Cai [BC06] em três categorias.

Contudo as funcionalidades variam conforme o tipo de turismo promovido. É neste sentido que Hall [Hal00], relativamente ao enoturismo, avalia um conjunto diferente de factores, como se pode observar na Tabela 2.2.

Funcionalidade
Informação regional alargada
Informação sobre rotas do vinho
Mapas Lista de facilidades para o visitante
Lista de serviços para o visitante
Links para sites sobre vinhos
Links para sites turísticos
Links para sites sobre eventos e festivais de vinho
Opções de compra online

Tabela 2.2: Os factores que contribuem para a permanência de visitantes no caso específico do enoturismo, segundo Hall [Hal00].

Fica fortalecida a ideia que a componente geográfica é essencial, dado ser referida em ambas as listas (mapas, ferramentas de *trip planning* e rotas do vinho).

2.2 Ferramentas de *Trip Planning*

As capacidades de *trip planning* têm passado a ser cada vez mais comuns, a partir do momento que *mashups* baseados em tecnologias de *mapping* online, como as API Google Maps ou Bing Maps se começaram a vulgarizar. Pan, Crotts e Muller [PCM07] desenvolveram um protótipo que permite o planeamento de viagens à cidade de Charleston,

Carolina do Sul, EUA utilizando uma interface Web construída sobre o Google Maps, contendo um conjunto de funcionalidades:

- Acrescentar POI à viagem;
- Alteração da ordem de visita aos pontos de interesse seleccionados;
- Remoção de POI da viagem;
- Visualização do percurso construído sobre o mapa.

Estas funcionalidades acabam por constituir o conjunto mínimo para uma ferramenta poder ser considerada de *trip planning*, sendo observáveis em todos os exemplos disponíveis na Web.

2.3 Sistemas de Recomendação

Vários sistemas têm sido implementados com o intuito de proporcionar recomendações de itens, algo que está a ganhar cada vez mais importância no actual contexto de excesso de informação, onde se torna necessário seleccionar um pequeno número de elementos de um conjunto várias ordens de magnitude superior.

Uma das estratégias que tem sido alvo de experimentação consiste na utilização do perfil do utilizador que procura sugestões, com o objectivo de proporcionar resultados personalizados; é algo que o motor de busca Google realiza sempre que uma pesquisa seja efectuada por um utilizador autenticado, por exemplo.

Emerge no entanto o problema da construção do perfil do utilizador. Castellano et al. propõem uma estratégia baseada em *fuzzy logic* e continua adaptação ao historial de visualizações. Começam por propor a compatibilidade entre a estrutura do perfil e a do item, por forma a ser possível determinar quais os itens que coincidem, dentro de um valor de tolerância, com um perfil; à medida que o utilizador vai visualizando itens, aqueles que viu há mais tempo deixam de ter tanto peso na determinação do seu perfil e aqueles que viu mais recentemente passam a ter mais peso [CDF⁺10] ou, alternativamente, a construção do perfil tem uma fase inicial *plástica*, em que é mais sensível a actualizações, e uma final *estável*, menos sensível [MTD⁺09]. Uma vantagem desta abordagem reside no facto da construção do perfil ser implícita e baseada somente no comportamento do utilizador mas, por outro, lado obriga a que todos os itens estejam devidamente classificados, o que nem sempre se torna viável.

Para além da relevância dos resultados, Iquinta et al. [IdGLS09] debruçaram-se sobre a problemática de conseguir surpreender o utilizador com itens de que este não estava à espera. Os autores distinguem no entanto *serendipismo* (o objectivo deste estudo particular) de *novidade*; enquanto que o segundo conceito diz respeito a itens que o utilizador

pudesse ter descoberto de uma forma autónoma, o primeiro concerne a itens que não seriam descobertos de outra forma. A estratégia que utilizam baseia-se na atribuição de valores para as probabilidades de cada item ser do interesse e de não ser do interesse do utilizador, tendo em conta o seu perfil. Onde um sistema clássico indicaria somente os itens com valores elevados da primeira probabilidade, neste são sugeridos também aqueles onde se verifique uma maior diferença entre os dois valores. O resultado pode ser visualmente ilustrado com duas sugestões de visita à Pinacoteca do Vaticano, somente com selecções obtidas por uma metodologia clássica, na Figura 2.1, e com acrescento de itens serendipiosos, na Figura 2.2.

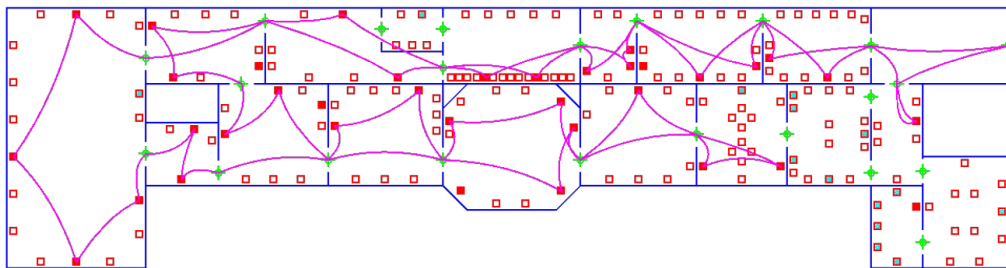


Figura 2.1: Sugestão personalizada de um percurso na Pinacoteca do Vaticano, utilizando metodologias clássicas.

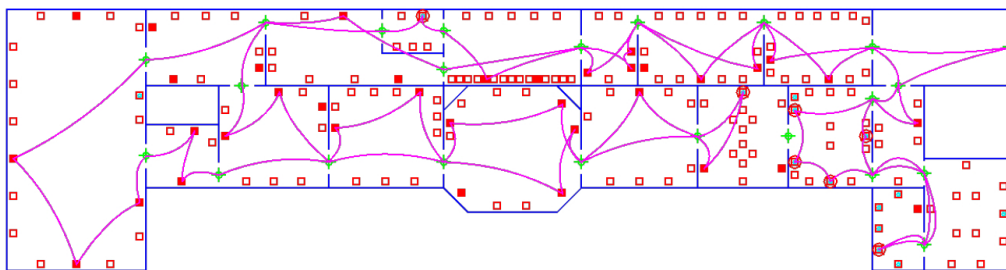


Figura 2.2: Sugestão personalizada, envolvendo agora itens serendipiosos.

2.4 Filtragem Colaborativa

Dentro dos sistemas de recomendação, particular destaque têm recebido recentemente os baseados em filtragem colaborativa, sendo a sua presença notória em sites de *e-commerce* como o da Amazon, por exemplo, com recomendações do género «*clientes que compraram este livro também compraram...*».

Uma das abordagens mais simples e fáceis de implementar é a *Slope One* [LM05], tendo-se tornado numa referência por esse motivo. Basicamente os autores em vez de

utilizarem modelos matemáticos para previsão lineares clássicos ou polinomiais, sugerem um linear e com inclinação 1,0 permitindo grande rapidez e, surpreendentemente, resultados ao nível dos lineares clássicos.

A investigação recente nesta área tem-se debruçado sobre vários refinamentos. Lou-same e Sánchez [LS10] discutem algumas formas de tirar partido das avaliações multicritério (classificações dadas separadamente ao argumento, à interpretação, à realização e ao visual de um filme, em oposição a uma nota global, por exemplo), construindo estratégias baseadas no conceito de *item view*, a tradução da visão pessoal de cada utilizador, que dá diferentes importâncias a diferentes critérios; os resultados mostram melhorias que atingem os 7,7% relativamente a metodologias clássicas, como a correlação de Pearson.

Na área de *web data mining*, Maratea e Petrosino [MP09] conseguem obter resultados mais relevantes de recomendação procurando utilizadores com um historial de navegação semelhante, no que diz respeito à sequência de categorias visitadas, em oposição às visitas a páginas individuais e à utilização de funções de agregação que não consideram a ordem de visita.

2.5 Motores de Sugestão Aplicados ao *E-Tourism*

O *e-tourism* há vários anos que beneficia dos desenvolvimentos na área dos sistemas de recomendação. São de particular destaque o *TripMatcher* da Triplehop e o *MePrint* da VacationCoach, que actualmente são os que têm mais sucesso [SWR⁺02]. Contudo têm um domínio de operação ligeiramente diferente do trabalho desta dissertação: a determinação do destino de umas férias.

No entanto, vários trabalhos de investigação têm conduzido a melhorias nesta área, embora ainda sem grande aplicação no terreno. Tornar o sistema adaptativo foi um caminho utilizado que deu frutos; Mahmood, Ricci e Venturini partiram de um sistema rígido e introduziram a capacidade de este se melhorar aprendendo a optimização da sua estratégia ([MR07] e [MR08]); posteriormente, o portal de turismo austríaco foi utilizado para testes e foi detectado um aumento da percentagem de aceitação por parte dos utilizadores das sugestões oferecidas [MRV09].

2.6 Conclusão

Todas as áreas que são abrangidas por este trabalho estão longe de estar estagnadas e estão em pleno desenvolvimento. Contudo, muito pouco material foca o processo de recomendação quando ainda não existem dados a partir dos quais se possam inferir gostos e padrões. Também ainda não é muito explorada a utilização de *cloud computing*, não obtendo a Google Prediction API qualquer resultado em motores de busca como o Google Scholar, IEEE Xplore ou ACM Digital Library. É particularmente promissor o

Revisão Bibliográfica

trabalho desenvolvido por [CSdVR10], que parece ter certos conceitos que podem ser desenvolvidos e adaptados para o problema proposto.

Capítulo 3

Metodologia

Várias abordagens são passíveis de serem aplicadas na resolução deste problema, mas perante a quantidade de factores que é necessário considerar, uma metodologia que integre cada um deles isoladamente e os vá agregando parece ser a mais adequada.

3.1 Mecanismos de Sugestão Estáticos e Dinâmicos

Os mecanismos de sugestão podem ser divididos em duas grandes categorias: estáticos e dinâmicos. O motivo desta discriminação prende-se com a natureza de cada um, com o facto de devolverem ou não valores diferentes conforme o perfil e acções do utilizador. Enquanto os primeiros podem pura e simplesmente ficar armazenados numa coluna da tabela pontos de interesse, os segundos têm de ser sempre recalculados a cada mudança no itinerário ou perfil do utilizador.

Considerar as classificações que o IGESPAR¹ faz do património de uma determinada região é independente do utilizador, ou seja, não varia qualquer que seja a rota escolhida ou gostos definidos, além de não precisar permanentemente de actualizações.

Pelo contrário, a um utilizador que está a planear um percurso passando pelos mesmos locais que outras pessoas, pode-lhe ser sugerido um POI que essas outras pessoas escolheram, caso demonstrem ter os mesmos gostos. É portanto um factor que se altera a cada mudança do conjunto de pontos de interesse a serem visitados, a cada clique do rato, precisando de adaptação constante.

De maneira a conseguir abordar cada factor isoladamente, pode ser utilizado um desenvolvimento iterativo, em que cada ciclo assenta no anterior e agrega mais um factor.

¹ Resultante da extinção e fusão do IPA e do IPPAR, é o organismo que em Portugal se encarrega de, entre outras funções, propor a classificação de imóveis de interesse nacional e de interesse público de relevância arquitectónica e arqueológica, e do estabelecimento de zonas especiais de protecção, assim como da sua gestão, restauro e pareceres sobre intervenções nestes.

Esta decomposição permite abordar problemas complexos pois isola cada sub-problema e lida com ele em profundidade, em contraste com a tentativa de ter uma visão global logo à partida.

3.2 Integração no Portal e Interface de Comunicação

Como é preciso obter informação relativa às escolhas que o utilizador vai fazendo, torna-se necessário definir a interface entre o motor de sugestões e o resto do portal. Do conjunto possível de interacções, as mais relevantes para a tarefa de recomendação são as relativas à construção do percurso e definição de gostos. À medida que estas acções vão tendo lugar, as sugestões oferecidas devem ir reflectindo as mudanças sucessivas. O diagrama UML apresentado na Figura 3.1 representa os casos de uso mencionados. Adicionalmente, o itinerário definido é constantemente armazenado no servidor e pode ser posteriormente recuperado para continuação da edição noutra sessão, embora sem a intervenção explícita do utilizador (daí a sua omissão do diagrama).

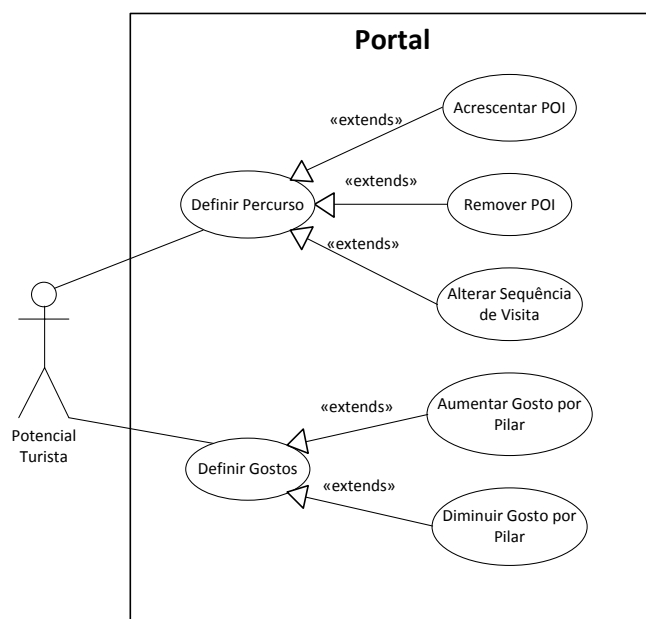


Figura 3.1: Casos de uso relevantes para a geração de recomendações.

Desta forma, a concepção do componente como *stateless* revela-se mais vantajosa em termos de facilidade de implementação e utilização, contrastando com a complexidade inerente a um que persista o seu estado e acompanhe as alterações incrementais e não incrementais realizadas pelo utilizador ao longo de uma ou mais sessões.

No contexto do sistema onde se integra, a interface pode assim ser definida através de um único método disponibilizado *Suggest* que, perante o itinerário e os gostos definidos, devolve um conjunto de sugestões, como ilustrado através do diagrama UML apresentado na Figura 3.2 que mostra o contexto onde o componente a desenvolver se enquadra.

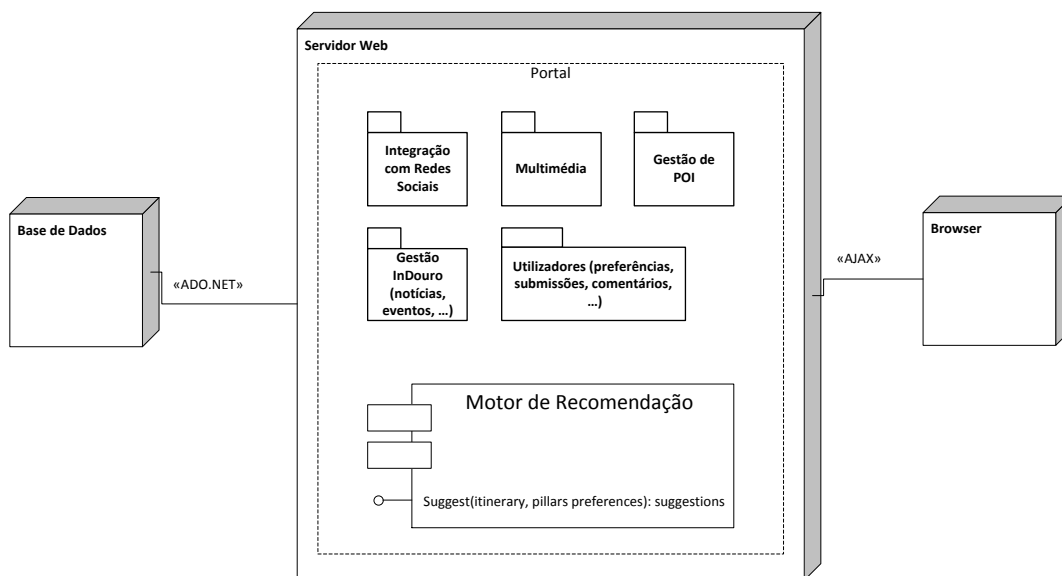


Figura 3.2: Parte do modelo de componentes, mostrando o contexto onde o motor de sugestões a desenvolver se enquadra.

3.3 Modo de Implementação e Teste

A metodologia escolhida também tem de considerar a forma como o resultado vai ser integrado no portal. O facto de ser necessário lidar com muitos dados de cada vez, torna necessário que seja um componente a correr no servidor e não no cliente (browser), em concordância com a Figura 3.2. Uma consequência é a impossibilidade de considerar interacção directa com o utilizador final. Adicionalmente, o facto que estarem envolvidos muitos factores impede uma resposta instantânea. Perante estes factos pode ser assumido que a resposta seja dada assincronamente, pois de outra forma a página onde os resultados deverão ser apresentados ficaria parada durante um período de tempo que não é adequado para a velocidade de resposta esperada dos elementos da interface de utilizador.

Levando em conta todas estes factos, pode ser considerado como produto final uma biblioteca dinâmica (DLL). Esta poderá ser acedida a partir de ASP.NET, que por sua vez pode ser chamado através de pedidos AJAX feitos a partir do cliente (quando um novo POI é adicionado a um plano de viagem, por exemplo).

Outra vantagem desta escolha é o facto de, embora exista um *mockup* da interface do planeador (já apresentado na Figura 1.3), o desenvolvimento do site ainda não ter abrangido essa parte. Como uma DLL não é específica de tecnologias web, o seu teste pode ser mesmo feito através de uma simples aplicação WPF. Foi o caso e o aspecto da aplicação de testes pode ser visto na Figura 3.3.

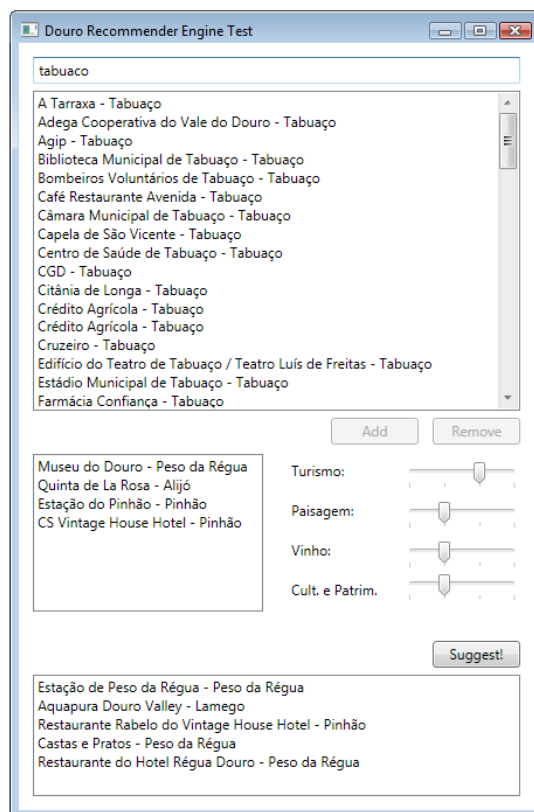


Figura 3.3: Aplicação para testar a DLL com a lógica de recomendação.

A caixa de texto na parte superior permite filtrar os POI listados, de uma forma não sensível a maiúsculas/minúsculas, nem a acentos e cedilhas. Os botões Add e Remove permitem adicionar POI seleccionados na *list box* de cima para a *list box* do meio (representando o itinerário). Os vários *sliders* permitem especificar os gostos do utilizador relativamente aos vários pilares. Por fim, o botão Suggest! invoca o motor de recomendações e apresenta os resultados na *list box* inferior. Havendo utilização da mesma interface programática pelas diversas implementações iterativas, este utilitário poderá assim ser utilizado tanto com a primeira, como com a última, como com qualquer outra versão do recomendador.

3.4 Conclusão

O desenvolvimento será realizado de uma forma iterativa e cumulativa, dividida no tratamento de características dependentes e independentes das escolhas do utilizador. O funcionamento do programa poderá ser testado com recurso a uma simples aplicação WPF enquanto a implementação do *trip planner* no portal não estiver concluída.

Metodologia

Capítulo 4

Mecanismo de Sugestões Estático

Um portal turístico pode ser desenvolvido com grande rapidez se assentar sobre um conjunto de tecnologias já maduras e testadas. Assim, na génese do projecto foi escolhido um CMS baseado em Windows, IIS, ASP.NET e SQL Server. Toda a lógica adicionada pelo INESC Porto assenta num conjunto de tabelas na mesma base de dados utilizada pelo CMS, mas organizadas no *schema* DOUROREGION, cujo modelo é apresentado na Figura 4.1.

Contudo, inicialmente esta não se encontrava suficientemente estável para poder ser utilizada, pelo que os primeiros testes com vista à elaboração de sugestões foram desenvolvidos sobre a base de dados do projecto Mobidouro, já mencionado na Secção 1.1. Mas progressivamente foi revelando um amadurecimento, pelo que acabou por ser possível utilizá-la.

4.1 Classificação dos POI com Base nas Fotos Disponíveis no Flickr

Como já foi referido na Secção 1.4, o número de referências para uma página serve como indicador da sua qualidade na ordenação dos resultados de uma pesquisa no motor Google. Fazer uma classificação semelhante dos POI existentes na Região do Douro acaba no entanto por não ser viável devido à ambiguidade das referências aos POI dispersas pela web. Outras formas inequívocas como telefone, NIPC ou código postal podem-se adequar ligeiramente a empresas, mas estão longe de serem adequadas a monumentos ou miradouros.

Uma forma mais adequada pode ser baseada na localização de fotografias, partindo da suposição que o interesse turístico de um POI é proporcional ao número de suas fotos.

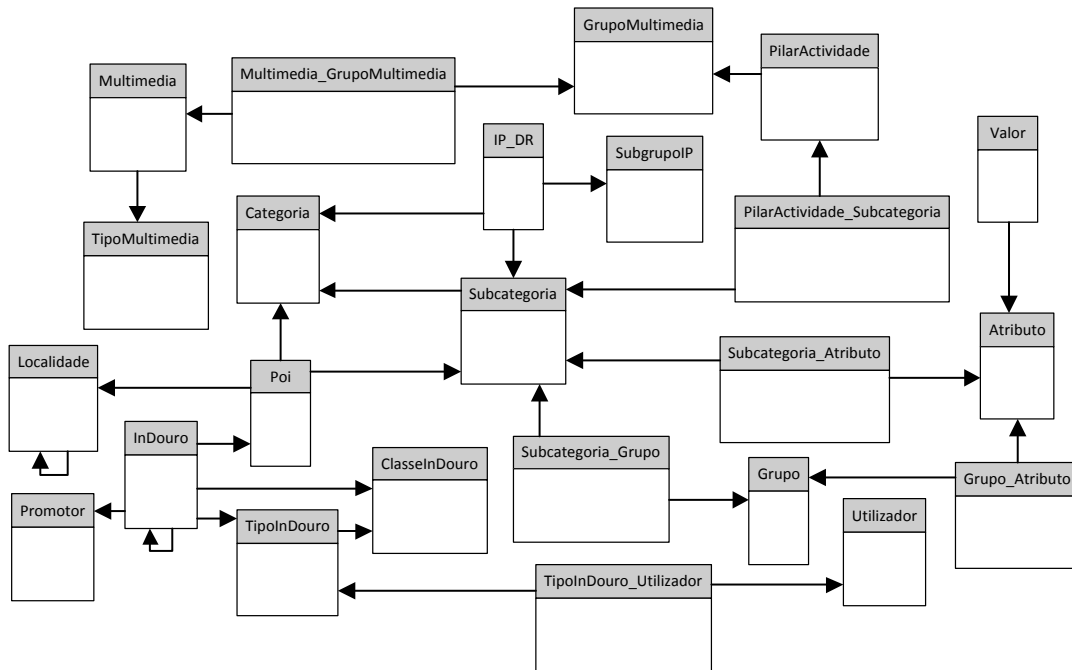


Figura 4.1: Modelo de dados, extraído utilizando o Visio, da base de dados SQL Server utilizada pelo portal, simplificado pela omissão da indicação das colunas das tabelas.

Por exemplo, se em Paris existem mais fotografias da Torre Eiffel do que da Torre Montparnasse, talvez se possa concluir que a primeira tem um interesse turístico superior ao da segunda.

Muitas das fotografias disponíveis no site Flickr possuem informação geográfica e adicionalmente é disponibilizada uma API de acesso, o que torna a sua utilização bastante adequada; em particular, é possível utilizá-la para obter as coordenadas das fotografias tiradas numa área definida por uma *bounding box* como, por exemplo, aquela que envolve a Região do Douro.

Torna-se necessário modificar a base de dados por forma a que esta possa acolher a informação relativa às fotografias, para o que uma nova tabela é suficiente, como a apresentada na Figura 4.2; a sua definição pode ser encontrada no Apêndice A.1.

Para preencher esta tabela pode ser utilizado o programa apresentado no Apêndice A.2, que utiliza a biblioteca Flickr.NET para aceder à API do Flickr.

Como referido, uma das limitações da API do Flickr é a impossibilidade de restringir a localização geográfica das fotos devolvidas numa pesquisa a um polígono, como o perímetro da Região do Douro, sendo somente possível especificar uma *bounding box*; a consequência é o carregamento na base de dados de informação relativa a fotos localizadas em Espanha, por exemplo, como pode ser visto na Figura 4.3(a); assim torna-se necessário eliminar estes registos através de um pós-processamento, como o apresentado

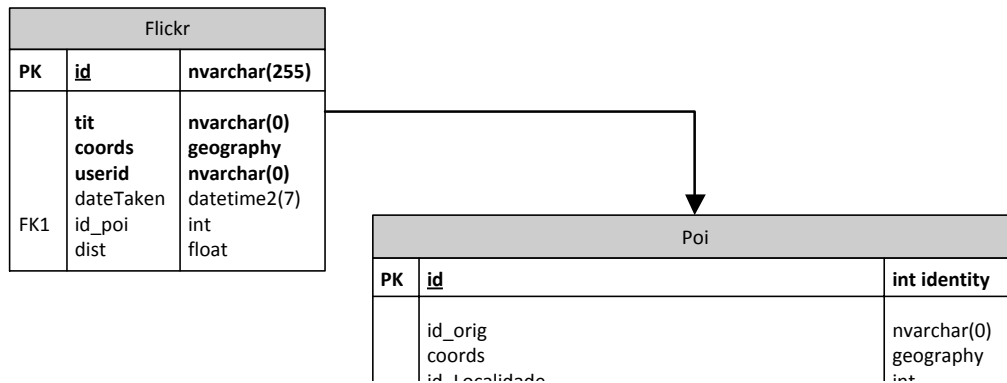


Figura 4.2: Pormenor do modelo de dados modificado, onde é possível observar a tabela destinada a receber informação do site Flickr.

na Listagem 4.1¹. O seu efeito pode ser visualizado na Figura 4.3(b).

```
DELETE FROM DOUROREGION.Flickr
WHERE id NOT IN (
    SELECT f.id
    FROM DOUROREGION.Localidade l, DOUROREGION.Flickr f
    WHERE l.id_Localidade IS NOT NULL
        AND l.poligono.STIntersects(f.coords) = 1
);
```

Listagem 4.1: A eliminação das fotografias não pertencentes à Região do Douro.

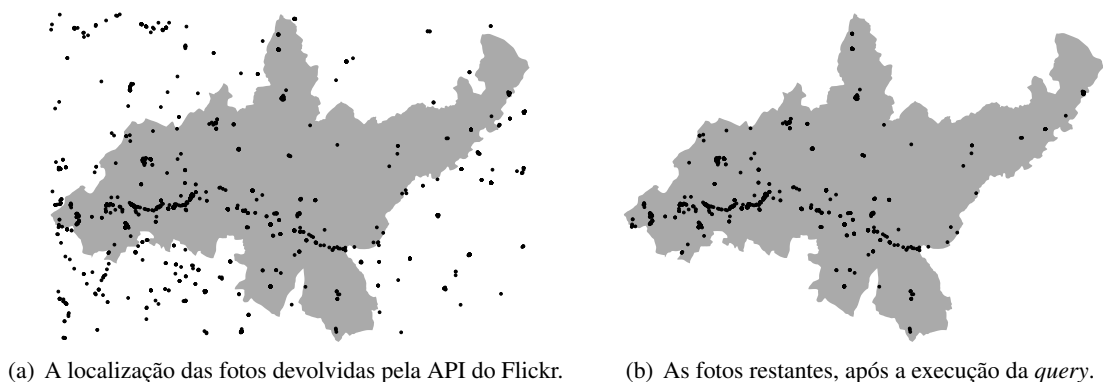


Figura 4.3: A acção da *query* de limpeza apresentada na Listagem 4.1.

O procedimento utiliza os contornos das freguesias da Região do Douro, presentes na

¹Obs.: *query* muito demorada!

tabela DOUROREGION.Localidade² e exclui as fotos cuja localização não se situe em nenhuma delas. É particularmente interessante verificar o quanto o Rio Douro é relevante em termos de interesse fotográfico, pois é possível distinguir boa parte do seu curso pela quantidade de fotos tiradas em torno dele.

Contudo, a base de dados sofre ainda de outro problema relacionado com a precisão das coordenadas. Por vezes, dezenas de fotos tiradas de diferentes locais de uma vila registam todas as mesmas coordenadas, especificamente as do centro da vila. A origem deste comportamento pode provir da georreferenciação *a posteriori*, com recurso a um serviço de *geocoding*; um exemplo é o pacote de software Windows Live Essentials (muito popular por conter o Windows Live Messenger) que oferece essa funcionalidade através da Galeria de Fotografias do Windows Live, e certas versões [Cou10] modificam as fotos mesmo sem previamente requererem a permissão do utilizador.

A API do Flickr indica qual a precisão de cada coordenada mas, nos testes realizados com utilização de uma *bounding box*, todas as fotos reportam terem precisão máxima, o que retira toda a utilidade desta informação.

Duas fotos tiradas no mesmo local naturalmente vão apresentar as mesmas coordenadas; no entanto, se o mesmo acontecer a mais de uma centena, é provável que tenham sido georeferenciadas *a posteriori*. Assim, uma forma de mitigar o problema consiste em estabelecer um *threshold* relativo ao número mínimo de ocorrências para que uma coordenada seja considerada resultante de *geocoding*. Surge então um valor de parametrização do sistema³, específico para este caso de estudo e encontrado através da experimentação com vários valores e observação de conjuntos de fotos que claramente foram tiradas em diferentes locais, mas que apresentam as mesmas coordenadas. A remoção das fotos cujas coordenadas surjam em número igual ou superior a *seis* é conseguida através da *query* apresentada na Listagem 4.2.

```
WITH CoordsQnt AS (
    SELECT coords.ToString() AS Coordenada, COUNT(*) AS Quantidade
    FROM DOUROREGION.Flickr
    GROUP BY coords.ToString()
    HAVING COUNT(*) >= 6
)
DELETE FROM DOUROREGION.Flickr
WHERE coords.ToString() IN (
    SELECT Coordenada
    FROM CoordsQnt
)
```

²Esta tabela, baseada em dados disponibilizados pelo Instituto Geográfico Português, também contém concelhos; contudo, a sua informação geográfica é menos precisa, pois resulta de uma união de várias freguesias e posterior simplificação aproximada. É possível distinguir as freguesias dos concelhos, dado que as primeiras referenciam o concelho a que pertencem através do campo *id_Localidade* e os segundos têm a NULL esse campo.

³O conjunto dos parâmetros que afectam o sistema estão compilados no Apêndice A.4.

);

Listagem 4.2: A CTE `CoordsQnt` descobre as coordenadas e o número de vezes que estas se repetem, caso este valor seja igual ou superior a seis; posteriormente são eliminados os registos relativos às fotos que tenham as coordenadas descobertas.

É preciso considerar que POI podem ser alvo de sugestão, pois muitos não possuem relevância turística. Uma possibilidade é criar uma *view* que filtre somente aqueles que podem ser escolhidos e assim, caso os critérios mudem, bastará alterá-la. Uma possibilidade é a apresentada na Listagem 4.3, que exclui os serviços (bombas de gasolina, farmácias, etc.), com excepção das estações de caminho de ferro, que no caso em estudo, têm interesse. São adicionalmente excluídos os pontos de interesse apagados, denotados por terem a sua relevância regional com o valor -1 , e aqueles que não possuírem coordenadas geográficas, um elemento fundamental em todo o processo, devido a eventuais erros na produção da base de dados. Outra vantagem é serem seleccionadas apenas algumas colunas da tabela original, que possui dezenas.

```
CREATE VIEW DOUROREGION.PoiTur
AS
SELECT p.id,
       p.nome,
       p.id_Categoria,
       p.id_Subcategoria,
       p.tipo,
       p.localidade,
       p.relevancia_regional,
       p.relevancia_local,
       p.coords
FROM DOUROREGION.Poi p JOIN DOUROREGION.Categoria c
  ON p.id_Categoria = c.id
WHERE (c.designacao <> N'Serviços'
       OR p.tipo = N'Estações de caminho de ferro')
       AND (p.relevancia_regional <> -1 OR p.relevancia_regional IS NULL)
       AND p.coords IS NOT NULL;
```

Listagem 4.3: Vista que exclui os POI sem interesse turístico.

Agora, é possível associar a cada fotografia o seu POI mais próximo, o que pode ser conseguido através da *query* apresentada na Listagem 4.4⁴. O cursor `@FlickrCursor` percorre todas as fotos e, para cada uma, são calculadas as distâncias a cada POI, recorrendo ao método `STDistance()`; depois são seleccionadas as distâncias mínimas e

⁴Obs.: *query* muito demorada!

finalmente o POI respectivo e a distância são guardados nas colunas `id_poi` e `dist` da tabela `DOUROREGION.Flickr`.

```

DECLARE @FlickrId nvarchar(255)
DECLARE @FlickrCursor CURSOR
SET @FlickrCursor = CURSOR FOR SELECT id FROM DOUROREGION.Flickr
OPEN @FlickrCursor
FETCH NEXT FROM @FlickrCursor INTO @FlickrId WHILE @@FETCH_STATUS = 0
BEGIN
    UPDATE DOUROREGION.Flickr
    SET id_poi = MinDist.PoiId, dist = MinDist.Dist
    FROM (
        SELECT TOP(1) p.id PoiId, f.coords.STDistance(p.coords) Dist
        FROM DOUROREGION.Flickr f, DOUROREGION.PoiTur p
        WHERE f.id = @FlickrId
        ORDER BY Dist
    ) MinDist
    WHERE id = @FlickrId

    FETCH NEXT FROM @FlickrCursor INTO @FlickrId
END
CLOSE @FlickrCursor
DEALLOCATE @FlickrCursor

```

Listagem 4.4: Uma *query* que associa a cada fotografia o seu POI mais próximo e respectiva distância.

Uma inspecção às associações que foram determinadas revela uma elevada percentagem de sucessos, sendo possível verificar casos como “Monumento aos Combatentes da Grande Guerra – Lamego – Portugal”/“Estátua do Soldado Desconhecido” ou “Monumento ao Herói Milhões”/“Estátua de Aníbal Augusto Milhães”, demonstrando pares que não teriam sido encontrados se fossem utilizadas somente as descrições, dada a sua subjectividade e a ocorrência de erros ortográficos. Aliás, não é possível utilizar os nomes das fotografias como critério, pois muitas mantêm o nome dado automaticamente pela máquina fotográfica (por exemplo, DSC00611, IMG_3749 ou P4265784). Uma amostra das 30 associações com maior proximidade pode ser encontrada no Apêndice [A.5](#).

No entanto, é previsível que à medida que a distância entre a foto e o respectivo POI aumente, diminua a probabilidade da associação ter sido correctamente realizada. Uma forma de tentar verificar a relação entre estes dois factores é considerar a existência de palavras comuns entre os nomes da foto e do seu POI, para o que foi desenvolvido o programa apresentado no Apêndice [A.3](#).

Tratando a saída resultante numa folha de cálculo e gerando o gráfico que relaciona a distância com o acumulado de associações correctas, é obtida a Figura 4.4.

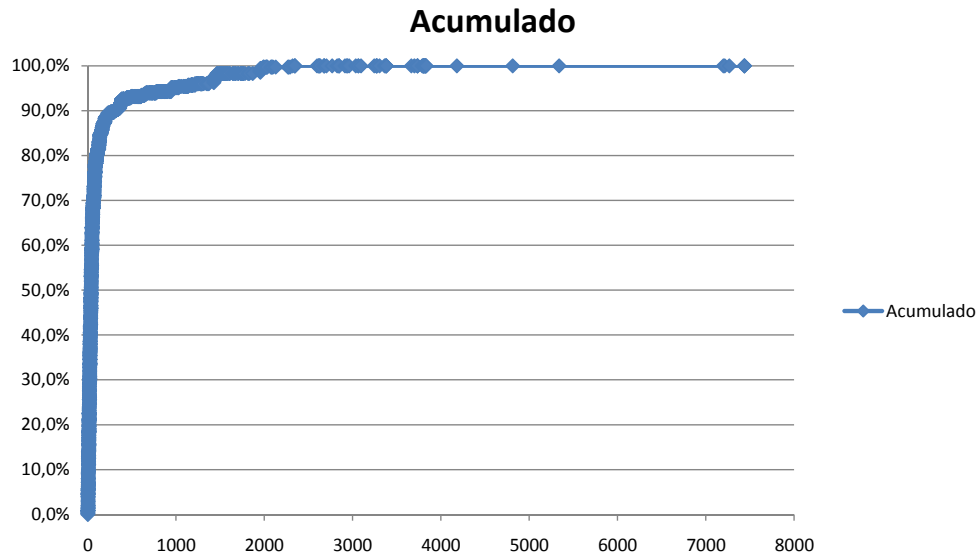


Figura 4.4: Relação entre o acumulado de associações correctas e a distância entre cada foto e o seu POI respectivo.

É possível observar como nos primeiros metros se encontram a maioria das associações consideradas correctas sob este critério, passando a rarearem com o aumento da distância. Inspeccionando o gráfico, constata-se que, para lá de aproximadamente 80%, este método começa a revelar a sua ineficácia, pelo que pode ser estabelecida nos correspondentes 99,4 metros a *distância máxima para que uma associação possa ser considerada válida*, pois é a que corresponde ao acumulado mencionado. Este acaba também por se tornar no segundo parâmetro de configuração do sistema⁵. Num espaço diferente da Região do Douro, este critério pode tomar valores muito diferentes (num espaço predominantemente urbano e não rural, por exemplo).

É possível agora introduzir na tabela DOUROREGION.Poi um valor classificativo com base na *popularidade fotográfica* determinada. O primeiro passo consiste em acrescentar uma coluna para acolher esse valor, o que pode ser conseguido com a *query* exibida na Listagem 4.5

⁵Ver Apêndice A.4.

```
ALTER TABLE DOUROREGION.Poi
ADD flickr float;
```

Listagem 4.5: Uma *query* que prepara a tabela DOUROREGION.Poi para receber a informação relativa à *popularidade fotográfica*.

De seguida, basta preencher essa coluna para que o POI com mais fotos associadas tenha o valor 1,0 e os restantes pontos de interesse tenham um valor proporcional. A *query* exibida na Listagem 4.6 realiza precisamente essa operação.

```
WITH PhotoAbsolute AS (
    SELECT p.id PoiId, COUNT(*) PhotoCnt
    FROM DOUROREGION.Flickr f JOIN DOUROREGION.Poi p ON f.id_poi = p.id
    WHERE f.dist <= 99.4
    GROUP BY p.id
)
UPDATE DOUROREGION.Poi
SET flickr =
    PhotoRelative.PhotoCnt / CAST(PhotoRelative.MaxPhotoCnt AS float)
FROM (
    SELECT pa.PoiId, pa.PhotoCnt, mx.MaxPhotoCnt
    FROM PhotoAbsolute pa, (
        SELECT MAX(PhotoCnt) MaxPhotoCnt
        FROM PhotoAbsolute
    ) mx
) PhotoRelative
WHERE id = PhotoRelative.PoiId;
```

Listagem 4.6: Uma *query* que associa a cada POI a proporção de fotos que tem associadas, relativamente ao ponto de interesse com mais fotos associadas.

Esta alteração precisa também de ser reflectida na vista criada com a Listagem 4.3. A Listagem 4.7 encarrega-se de fazer o devido acrescento.

```
ALTER VIEW DOUROREGION.PoiTur
AS
SELECT p.id,
       p.nome,
       p.id_Categoria,
       p.id_Subcategoria,
       p.tipo,
       p.localidade,
```

```

    p.relevancia_regional,
    p.relevancia_local,
    p.coords,
    p.flickr
FROM DOUROREGION.Poi p JOIN DOUROREGION.Categoria c
    ON p.id_Categoria = c.id
WHERE (c.designacao <> N'Serviços'
    OR p.tipo = N'Estações de caminho de ferro')
    AND (p.relevancia_regional <> -1 OR p.relevancia_regional IS NULL)
    AND p.coords IS NOT NULL;

```

Listagem 4.7: Segunda versão da vista exclui os POI sem interesse turístico, agora exibindo também a coluna flickr.

4.2 Google Places

Um serviço disponibilizado pela Google relevante para este trabalho é o *Google Places*, que permite ao público em geral classificar locais que se encontrem listados. Oferece ainda uma API que permite aceder a informação relativa aos mesmos, desde o nome e morada até às coordenadas geográficas e média das classificações atribuídas, tornando muito adequada a sua integração com este projecto. Na Figura 4.5 pode ser vista a interface acessível ao público.

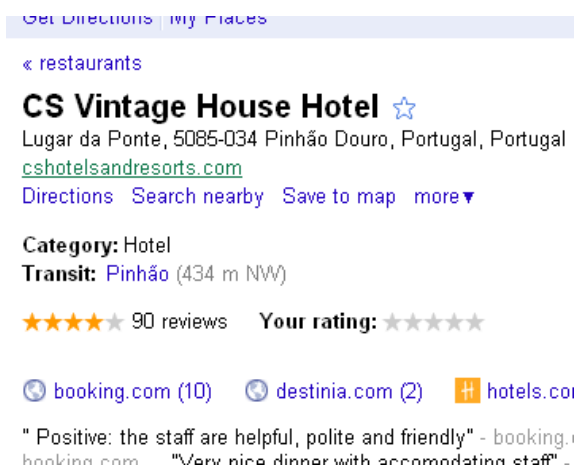


Figura 4.5: Pormenor da interface do *Google Places*, onde é possível classificar um local.

A preparação da base de dados para acolher esta informação resume-se ao acrescento de uma coluna *places* do tipo *float*.

O descarregamento da informação pode ser conseguido recorrendo ao programa apresentado no Apêndice A.6. Embora o código implemente algumas ideias discutidas na

secção anterior para decidir se a associação entre um POI e uma foto foi conseguida com sucesso (para a geração do gráfico), a exigência neste caso tem de ser maior, sendo obrigatório que todas as palavras com mais de três letras surjam em ambos os nomes, uma vez removidos os acentos e as cedilhas, e convertidas as letras maiúsculas para minúsculas.

Um exemplo ilustrativo desta necessidade é o par “Miradouro de São Leonardo da Galafura”/“Pôr-do-Sol no Miradouro da Galafura” em oposição a “Miradouro de São Leonardo da Galafura”/“MIRADOURO SÃO LEONARDO DA GALAFURA”. Muitos nomes no *Google Places* estão completamente em maiúsculas e com ligeiras diferenças na utilização dos artigos e determinantes, enquanto que no Flickr em muitos casos surgem com acrescentos circunstanciais e transcritos parcialmente.

Esta alteração precisa também de ser reflectida na vista criada com a Listagem 4.7. A Listagem 4.8 encarrega-se de fazer o devido acrescento.

```

ALTER VIEW DOUROREGION.PoiTur
AS
SELECT p.id,
       p.nome,
       p.id_Categoria,
       p.id_Subcategoria,
       p.tipo,
       p.localidade,
       p.relevancia_regional,
       p.relevancia_local,
       p.coords,
       p.flickr,
       p.places
FROM DOUROREGION.Poi p JOIN DOUROREGION.Categoria c
    ON p.id_Categoria = c.id
WHERE (c.designacao <> N'Serviços'
       OR p.tipo = N'Estações de caminho de ferro')
       AND (p.relevancia_regional <> -1 OR p.relevancia_regional IS NULL)
       AND p.coords IS NOT NULL;

```

Listagem 4.8: Terceira versão da vista exclui os POI sem interesse turístico, agora exibindo também a coluna places.

Embora seja um promissor serviço quando muitos locais estiverem classificados, infelizmente no caso de estudo só oito classificações estão disponíveis, o que reduz a sua utilidade. No entanto, no futuro e na adaptação da plataforma a outras regiões turísticas, o benefício poderá ser bem maior.

4.3 Classificação Editorial

Embora os meios automáticos de estabelecer classificações com base no Flickr e no Google Places sejam um bom ponto de partida, são insuficientes e precisam de ser complementados por uma classificação estabelecida editorialmente, de preferência realizada por alguém que conheça bem a região e que também considere fontes cujo acesso não possa ser automático.

Este projecto contou precisamente com este género de apoio, e o resultado pode ser encontrado na coluna *relevancia_regional* da tabela *DOUROREGION.Poi*. Trata-se de valores que seguem a escala popularizada pela Michelin, de zero a três estrelas; precisam no entanto de um pré-tratamento, pois esta coluna acumula também a função de indicar que um POI se encontra eliminado, tendo neste caso o valor -1, e se merece destaque na *visita virtual* (um modo de visualização 3D do portal), caso em que lhe são adicionadas 10 unidades. Dado os POI eliminados serem filtrados pela vista definida na Listagem 4.8, é somente necessário lidar com os da visita virtual:

$$rel = rel_{reg} \mod 10$$

... onde a relevância, denotada por *rel*, é obtida a partir do valor presente na coluna *relevancia_regional*, denotado por *rel_{reg}*.

Na avaliação efectuada, para além do conhecimento pessoal, foram utilizadas fontes como o IGESPAR, a UNESCO, Booking.com, Escape by Expresso, assim como diversos livros. Devido à alta qualidade da classificação, é natural que esta tenha um peso elevado nas recomendações realizadas.

Está disponível adicionalmente uma coluna *relevancia_local*, que possui valores entre zero e dois, tendo por objectivo o detalhe de locais num contexto mais reduzido, como num concelho ou freguesia, em oposição a em toda a região. Também pode ser considerado no processo de decisão, mas com um peso menor que a relevância regional.

4.4 Classificação dos Utilizadores

O recurso ao Google Places e o seu uso dos votos do público em geral possui a grande vantagem de já estar disponível aquando do lançamento de um novo site turístico mas, o facto de usar uma base de dados de POI diferente e ter um público demasiado geral, onde uma região da dimensão do caso de estudo representa uma ínfima parte, torna-a inadequada a longo prazo.

Torna-se assim clara a necessidade de prever a existência de um sistema de votações que, numa primeira fase recolha classificações dos utilizadores e, numa segunda fase as utilize na geração de recomendações.

A estrutura de dados para receber as classificações pode ser bastante simples, bastando uma tabela como a exibida na Figura 4.6; a sua definição pode ser encontrada no Apêndice A.7.

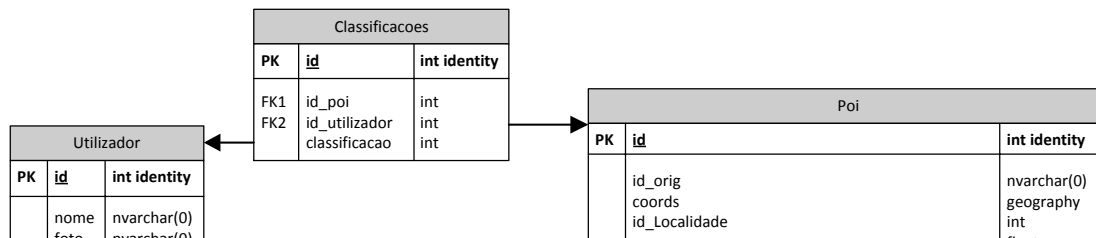


Figura 4.6: Pormenor do modelo de dados, após receber a tabela destinada a guardar as classificações dos utilizadores.

Quando um número considerável de classificações estiver presente nesta tabela, os seus dados poderão ser incluídos no processo de recomendação, como será mencionado no Capítulo 6, dedicado às conclusões e trabalho futuro.

4.5 Conclusão

Preencher a base de dados de um site novo com informação pode ser um desafio, mas a utilização de recursos disponíveis livre e programaticamente, desde as fotos do Flickr até às avaliações do Google Places permitem colmatar parte das necessidades. Contudo, ter estes dados complementados por uma pessoa que conheça a região e dedicada à pesquisa de vários meios tais como livros, revistas e sites, permite ter um corpo de informação suficientemente bom. Uma vez em funcionamento durante algum tempo, se as classificações produzidas pelos utilizadores forem sendo armazenadas, poderão também ser posteriormente incorporadas no processo de sugestão.

Capítulo 5

Mecanismo de Sugestões Dinâmico

No capítulo anterior foram discutidas várias formas de estabelecer uma classificação dos vários pontos de interesse de uma região. Contudo, não é tido em conta nenhum detalhe do perfil nem do percurso do utilizador que pretende obter recomendações. Fazê-lo constitui um desafio pois, aquando do arranque de um portal de uma região turística, este ainda não possui utilizadores, muito menos um histórico de itinerários “A Minha Viagem”.

5.1 Sugestão Aleatória de Pontos de Interesse

A escolha aleatória de alguns pontos talvez seja o ponto de partida mais simples para o problema em questão, o que pode ser conseguido através de uma lógica extremamente básica, como a apresentada na Listagem 5.1, um extracto da primeira versão do recomendador, apresentado na sua totalidade no Apêndice A.8. São seleccionados todos os POI com interesse turístico (recorrendo à vista apresentada na Listagem 4.3) e é associado a um deles (representado pelo seu *id*) um valor aleatório; a enumeração resultante é ordenada segundo esse valor aleatório, os pontos do itinerário do utilizador são removidos (para não ser sugerido um POI já escolhido), os melhores valores são seleccionados e o resultado é devolvido como um *array* de *ids* de pontos de interesse. Uma saída de uma execução é apresentada na Tabela 5.1 e visualizada na Figura 5.1.

```
SqlCommand command = new SqlCommand(@"  
SELECT p.id  
FROM DOUROREGION.PoiTur p;",  
    this.conn);  
SqlDataAdapter da = new SqlDataAdapter(command);
```

Mecanismo de Sugestões Dinâmico

```
DataSet ds = new DataSet();
da.Fill(ds, "DOUROREGION.PoiTur");
return ds
    .Tables["DOUROREGION.PoiTur"]
    .AsEnumerable()
    .Select(dr => (int)dr["id"])
    .Except(itinerary)
    .Select(poiId => new
    {
        Id = poiId,
        RandomComponent = this.rnd.NextDouble(),
    })
    .OrderByDescending(poi => poi.RandomComponent)
    .Select(poi => poi.Id)
    .Take(this.Count)
    .ToArray();
```

Listagem 5.1: Código que obtém aleatoriamente cinco POI.

	Nome	Id	Coordenadas
1	Pousada de Juventude de Vila Real	2135	41° 18' 10,116" N; 7° 44' 20,015" O
2	Igreja Matriz de Valverde	6307	41° 17' 31,999" N; 6° 48' 38,002" O
3	Fonte das Lavandeiras	3288	41° 14' 52,908" N; 7° 25' 50,250" O
4	Estoril	5543	41° 20' 23,351" N; 6° 43' 04,012" O
5	Parque Infantil de Foz Coa	2415	41° 04' 47,537" N; 7° 07' 56,669" O

Tabela 5.1: Cinco POI escolhidos aleatoriamente.

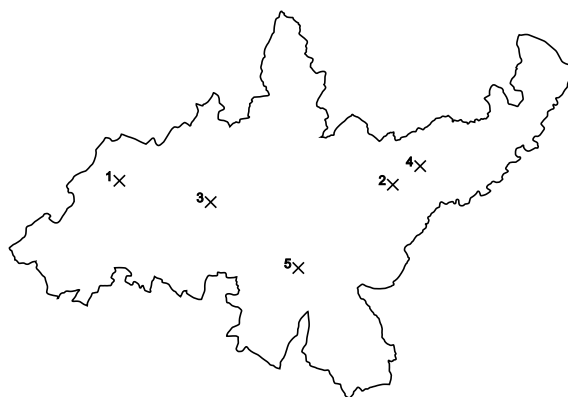


Figura 5.1: Visualização de cinco POI escolhidos aleatoriamente.

Para além da sua simplicidade, uma vantagem adicional é o facto de cada execução proporcionar um conjunto de resultados diferentes. Se ocorrer um refrescamento da lista de sugestões exibidas periódico ou a cada alteração do percurso, este é exposto a um

maior número de possibilidades sem haver sacrifício de espaço na interface do utilizador ou adição da complexidade desta, algo que não seria possível caso fosse utilizado um critério fixo de ordenação.

5.2 Restrição das Sugestões à Proximidade do Itinerário do Utilizador

Contudo, o método anterior não considera o percurso que o utilizador pretende realizar, podendo sugerir pontos de interesse cuja visita implique um desvio demasiado grande. Assim, uma primeira medida pode consistir em restringir os POI considerados àqueles que se encontrem a uma distância máxima do itinerário.

Relevante para este objectivo é o método `OGC STBuffer()` que gera uma área que envolve a geometria sobre a qual é invocado, recebendo como parâmetro o valor da margem. Um exemplo é a aplicação a um ponto, que gera um círculo cujo raio é a margem especificada; outro exemplo é a aplicação à linha de separação de sentidos de uma estrada: se a margem especificada for a largura da faixa de rodagem, é obtido um polígono representando a superfície dessa estrada.

O primeiro passo a ser dado para conseguir utilizar o método consiste na construção de uma geometria baseada na sequência de POI que o utilizador pretende visitar, o que é conseguido através do método `BuildPath()`, que gera uma primitiva `OGC LineString` que liga todos os POI por onde o utilizador pretende passar. Na segunda versão do recomendador, apresentada no Apêndice A.9 (com todas as diferenças em relação à versão anterior devidamente assinaladas), pode ser encontrado o corpo deste método.

De seguida, o código da primeira versão é adaptado de forma a considerar o método `STBuffer()` aplicado à geometria construída, como pode ser visto através do extracto exibido na Listagem 5.2. Embora a verificação da inclusão de um POI no *buffer* criado possa ser feita no cliente, isso implicaria não tirar partido de eventuais índices espaciais que possam ser criados sobre a tabela `DOUROREGION.Poi` (subjacente à vista `DOURO-REGION.PoiTur`) e necessitaria de uma maior transferência de dados entre as camadas de persistência e lógica.

Um resultado de uma execução é apresentado na Tabela 5.2 e a respectiva visualização pode ser encontrada na Figura 5.2. A entrada consiste num itinerário percorrendo o Jardim Botânico da Universidade de Trás-os-Montes e Alto Douro em Vila Real (*id* 2320), Museu do Douro no Peso da Régua (*id* 1934) e os Azulejos da Estação de Caminhos de Ferro do Pinhão (*id* 3254), ou seja o *array* {2320, 1934, 3254}.

```
SqlGeography travelPath = this.BuildPath(itinerary);
SqlCommand command = new SqlCommand(@"
```

Mecanismo de Sugestões Dinâmico

```
SELECT p.id
FROM DOUROREGION.PoiTur p
WHERE @buffer.STIntersects(p.coords) = 1;";
    this.conn);
    command.Parameters.Add(new SqlParameter
    {
        ParameterName = "@buffer",
        UdtTypeName = "geography",
        Value = travelPath.STBuffer(this.Distance),
    });
```

Listagem 5.2: Introdução de uma zona de restrição à selecção dos POI.

	Nome	Id	Coordenadas
1	Polidesportivo de Adorigo	1242	41° 08' 42,828" N; 7° 36' 33,570" O
2	Igreja Matriz de Penajóia	6245	41° 08' 46,000" N; 7° 51' 11,999" O
3	Imagem de Santa Iria	3000	41° 18' 35,856" N; 7° 44' 20,483" O
4	Pizzaria Limonete	5654	41° 09' 51,236" N; 7° 47' 34,826" O
5	Quinta do Crasto	6560	41° 11' 06,754" N; 7° 38' 04,632" O

Tabela 5.2: Um resultado obtido, utilizando o método `STBuffer()`.

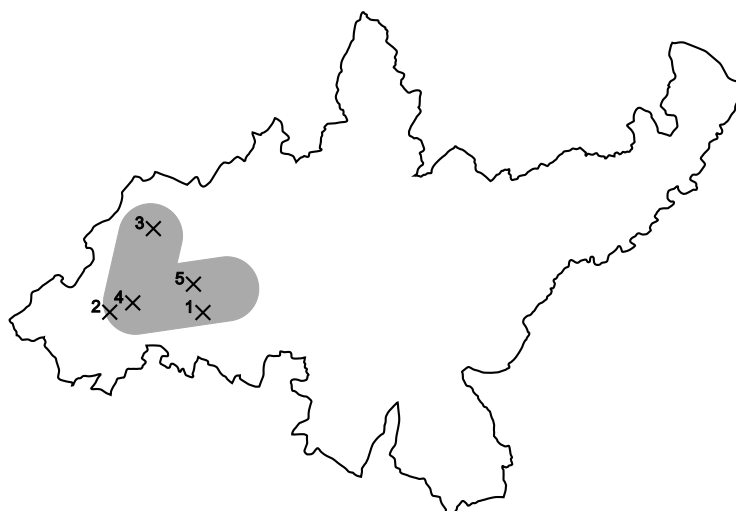


Figura 5.2: Visualização do resultado, com a zona criada pelo método `STBuffer()`, assinalada a cinzento.

5.3 Prioritização dos POI Mais Próximos do Itinerário do Utilizador

Após a considerável redução do espaço de pesquisa aos POI a uma distância máxima do itinerário escolhido pelo utilizador, é possível refinar o processo de selecção dando preferência àqueles que se encontram mais próximos, em detrimento dos mais afastados. Mais uma vez se torna útil o método `STDistance()`, já utilizado na Secção 4.1. A ordenação que precede a escolha dos POI leva em conta a componente da distância (expressa num valor entre 0,0 e 1,0, inversamente proporcional à distância) e a aleatória (também com valores compreendidos entre 0,0 e 1,0) com pesos por omissão de 2,0 e 1,0 respectivamente. Na Listagem 5.3 é apresentado um extracto que implementa esta lógica; o programa completo pode ser consultado no Apêndice A.10.

```
.Tables["DOUROREGION.PoiTur"]
.AsEnumerable()
.Select(dr => new POI
{
    Id = (int)dr["id"],
    Coords = dr["coords"] as SqlGeography,
})
.Except(itineraryPoi)
.Select(poi => new
{
    Poi = poi,
    RandomComponent = this.rnd.NextDouble(),
    DistanceComponent =
        1 - travelPath.STDistance(poi.Coords).Value /
        this.Distance,
})
.OrderByDescending(poi =>
    this.WeightDistance * poi.DistanceComponent +
    this.WeightRandom * poi.RandomComponent)
.Select(poi => poi.Poi.Id)
.Take(this.Count)
.ToArray();
```

Listagem 5.3: A utilização da distância ao percurso como factor de escolha do POI a sugerir.

5.4 Consideração do Perfil Explícito do Utilizador

Na sua definição de perfil, o utilizador pode explicitamente definir os seus níveis de interesse relativamente a vários pilares (que, no caso de estudo, são “Turismo”, “Paisagem”,

“Vinho” e “Cultura & Património”) através de *sliders* disponibilizados pela interface que permitem a selecção de um valor discreto entre 0 e 3, como se pôde ver na Figura 1.2. Cada invocação do método `Suggest()` requer a passagem destes valores sob a forma que um *array* de quatro inteiros.

Cada ponto de interesse tem associada uma subcategoria que, por sua vez, pode ter associada vários pilares. Perante conflitos no interesse mostrado pelo utilizador nos casos em que um ponto de interesse compreende vários pilares, pode ser utilizado aquele com valor mais elevado, de forma a haver mais inclusão. Também se torna necessário definir um peso a atribuir a esta componente. Passar a ter em conta este critério é bastante simples, bastando o excerto apresentado na Listagem 5.4, utilizando um método `GetPillars()` que obtém os pilares associados a um POI. O programa completo pode ser encontrado no Apêndice A.11.

```
.Select(dr => new POI
{
    Id = (int)dr["id"],
    Coords = dr["coords"] as SqlGeography,
    Subcategory = subcategories
        .Single(sc =>
            (sc.Id == (int)dr["id_Subcategoria"]))
})
.Except(itineraryPoi)
.Select(poi => new
{
    Poi = poi,
    RandomComponent = this.rnd.NextDouble(),
    DistanceComponent =
        1 - travelPath.STDistance(poi.Coords).Value /
        this.Distance,
    PillarComponent = poi
        .Subcategory
        .Pillars
        .Select(scPillar => pillars[scPillar - 1])
        // Lidar com caso de não haver pilar associado.
        .Concat(new int[] { 0 }).Max() / 3.0,
})
.OrderByDescending(poi =>
    this.WeightDistance * poi.DistanceComponent +
    this.WeightRandom * poi.RandomComponent +
    this.WeightPillars * poi.PillarComponent)
```

Listagem 5.4: A utilização das preferências do utilizador relativamente a pilares como factor de escolha do POI a sugerir.

5.5 Consideração do Perfil Implícito do Utilizador

Um problema que existe aquando do lançamento de um site turístico é a inexistência de histórico de planeamentos de viagem que permitam realizar *data mining*. Uma possível forma de resolver este problema consiste em utilizar informação livremente disponível na Web, como no site Flickr, por exemplo, que além de disponibilizar informação geográfica associada a muitas fotos, indica também qual o utilizador que as tirou, assim como a data e hora em que o fez. Com estes dados, é possível reconstruir cada percurso. A Figura 5.3 ilustra 115 itinerários extraídos do referido site.

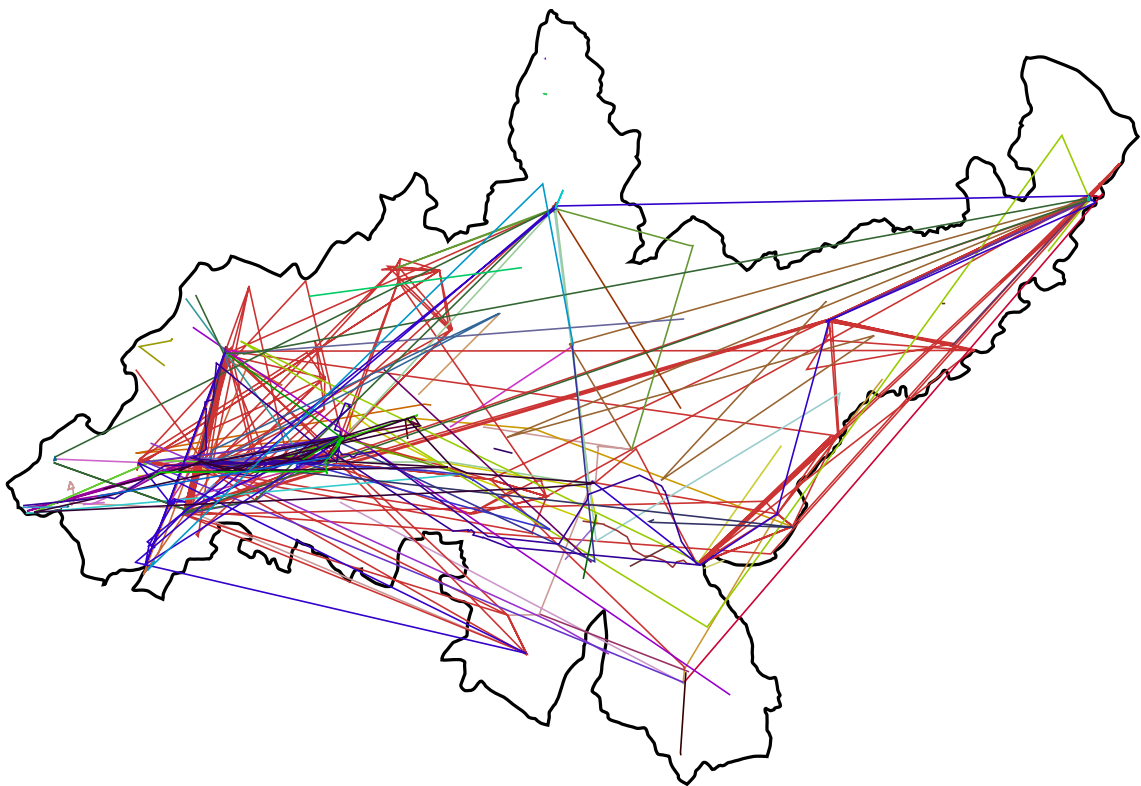


Figura 5.3: Reconstrução de 115 percursos, cada um com uma tonalidade diferente, a partir dos dados disponíveis no site Flickr.

Se forem considerados os POI associados a cada ponto do itinerário que não ultrapassem a distância limite imposta, como discutido na Secção 4.1, é possível reconstruir alguns percursos também no que diz respeito a passagens por pontos de interesse. Como está disponível a categorização de cada um, a partir destas é possível tentar inferir recomendações recorrendo a técnicas baseadas em *machine learning*.

Das várias soluções de *machine learning* disponíveis, foi seleccionada a *Google Prediction API*, acessível através de uma interface REST. Os pacotes de software *Weka* e

Apache Mahout também foram considerados, mas o facto de qualquer um deles introduzir mais uma *framework* (Java) acaba por os tornar menos atractivos¹.

Esta API está em permanente desenvolvimento e selecciona automaticamente qual a técnica de previsão mais adequada (regressão linear, redes neuronais, ...). A sua utilização passa pela preparação de um ficheiro de treino, que é tão-somente um formato tabular plano CSV onde cada linha possui como primeiro valor a suposta resposta, e nos seguintes, os dados que a condicionaram.

O ficheiro de treino é então alvo de um *upload* para a plataforma de armazenamento *Google Storage for Developers* que faz parte da oferta de *cloud computing* desta empresa. Embora este *upload* seja feito utilizando uma ferramenta desenvolvida em Python, este passo situa-se na fase de *setup*, não sendo necessária a presença desta *framework* no ambiente de produção.

Uma forma de treinar o sistema com os itinerários reconstruídos poderia passar por especificar que um POI seria recomendado perante o restante conjunto de pontos de interesse desse percurso. Ou seja, para um percurso (p_1, p_2, p_3, p_4) , a informação de treino seria:

$$\begin{aligned} p_1 &\leftarrow (p_2, p_3, p_4) \\ p_2 &\leftarrow (p_1, p_3, p_4) \\ p_3 &\leftarrow (p_1, p_2, p_4) \\ p_4 &\leftarrow (p_1, p_2, p_3) \end{aligned}$$

Esta ideia parte da suposição que um utilizador que tenha gostos que o levem ir aos pontos p_2 , p_3 e p_4 recomenda o ponto p_1 , pois foi lá. Assim, o sistema é treinado para dar essa sugestão a outro utilizador que decida ir aos mesmos POI.

Contudo esta abordagem sofre de vários problemas. Para uma quantidade de itinerários reconstruídos tão reduzida, não existem dados suficientes que permitam treinar o sistema de forma eficaz (a precisão reportada cifrava-se em 0%). Adicionalmente, o facto de cada percurso ter um número diferente de POI impede a criação de uma tabela de treino com um número homogéneo de colunas, se a cada uma corresponder um ponto de passagem.

Desta forma, foi alternativamente explorada a via de sugerir a categoria de POI com base na proporção de categorias dos pontos do resto do itinerário. Assim, se ao percurso (p_1, p_2, p_3, p_4) corresponderem as categorias $(cat_1, cat_2, cat_3, cat_4)$, o treino basear-se-ia

¹Por forma a reduzir as dificuldades de integração decorrentes da utilização de sistemas heterogéneos, o projecto já considerou substituir a actual solução publicitária devido à utilização que faz da *framework* PHP e da base de dados MySQL.

nesta informação:

$$\begin{aligned} cat_7 &\leftarrow (0, 0, 1/3, 0, 1/3, 0, 1/3, 0, 0, 0) \\ cat_3 &\leftarrow (0, 0, 0, 0, 1/3, 0, 2/3, 0, 0, 0) \\ cat_7 &\leftarrow (0, 0, 1/3, 0, 1/3, 0, 1/3, 0, 0, 0) \\ cat_5 &\leftarrow (0, 0, 1/3, 0, 0, 0, 2/3, 0, 0, 0) \end{aligned}$$

Um programa para gerar um ficheiro de treino com estas condições pode ser encontrado no Apêndice A.12. O resumo da execução do treino é devolvido sob a forma de um objecto JSON; na Tabela 5.3 pode ser encontrado o conteúdo do mesmo, formatado para melhor legibilidade. Observa-se que o tipo de modelo reconhecido é o de *classificação* e que a precisão se situa nos 55%. Este valor é relativamente sensível: num teste eliminando a possibilidade das fotos poderem ser associadas a pontos de interesse da categoria *serviços*, mesmo estações de caminho de ferro, baixava para os 51%; noutro, utilizando subcategorias em vez de categorias, era obtido o valor ainda mais baixo de 33%, pois o reduzido número de percursos reconstruídos não permite predições com maior granularidade.

Kind		prediction#training
Id		feupmieic/treino.csv
Self Link		https://www.googleapis.com/prediction/v1.2/training/%s
Model	Model Type	classification
Info	Classification Accuracy	55%
Training Status		DONE

Tabela 5.3: Resumo apresentado no final de uma sessão de treino.

O resultado da predição:

$$(0, 0, 2/3, 1/3, 0, 0, 0, 0, 0, 0)$$

...é o JSON que pode ser encontrado na Tabela 5.4, após ter sido formatado para melhor legibilidade. O resultado predito é a *Categoria 7*, mas logo de seguida são listadas detalhadamente todas as medidas do quanto cada categoria parece adequar-se.

Seguindo a metodologia utilizada até agora, as várias percentagens podem ser utilizadas para a classificação de todos os POI considerados, de acordo com a sua categoria, bastando para isso utilizar mais um peso, específico desta componente.

A plataforma *Google Storage for Developers* organiza os ficheiros armazenados em partições lógicas denominadas *buckets*, sendo a referência a um ficheiro precedida pelo nome do *bucket* e pelo carácter */* como, por exemplo, “douro/treino.csv”. O nome do

Kind		prediction#output								
Id		feupmieic/flickr_douro4.csv								
Self Link		https://www.googleapis.com/prediction/v1.2/training/%s/predict								
Output Label		Cat7								
Output	Label	Cat8	Cat5	Cat1	Cat7	Cat2	Cat4	Cat6	Cat9	Cat3
Multi	Score	7,5%	8,0%	23,9%	33,0%	8,7%	2,9%	1,3%	14,3%	0,4%

Tabela 5.4: Saída de uma predição.

modelo treinado acaba por ser esta referência; no entanto a sua utilização num endereço supostamente implica que seja sujeita a *URI encoding* que, no exemplo, se torna “douro%2ftreino.csv”.

A especificação da *Google Prediction API* define como endereço de acesso à funcionalidade de predição, o indicado na Tabela 5.4 na entrada *Self Link*, sendo a parte %s substituída pelo nome do modelo “url encoded” sobre o qual se pretende fazer uma previsão. Assim, o URI do exemplo acaba por ser:

```
https://www.googleapis.com/prediction/v1.2/training/douro%2ftreino.csv/predict
```

O problema deste endereço reside no facto de não ser válido [BLFM98], o que dificulta a sua utilização em algumas implementações, muito notavelmente a *framework .NET* nas suas versões mais recentes.

A forma convencional de realizar um pedido POST, necessário para obter uma predição, passa pela utilização da classe `HttpWebRequest`, cujo construtor aceita um objecto da classe `Uri`; esta classe por seu turno aceita uma *string* com o endereço desejado; este endereço é então sujeito a uma normalização, sendo transformado em:

```
https://www.googleapis.com/prediction/v1.2/training/douro/treino.csv/predict
```

No entanto, perante este endereço, a API devolve um erro e não o resultado da predição. Nas primeiras versões da *framework .NET* a classe `Uri` aceitava no seu construtor um parâmetro booleano *dontEscape* e, se este fosse *verdadeiro*, não procedia à normalização do endereço; contudo esta *flag* está actualmente marcada como obsoleta, sendo ignorada. Apesar desta situação ter sido abordada nos fóruns de discussão da *Google Prediction API*², não houve resposta em tempo útil por parte da empresa.

Existem algumas formas de contornar o problema. Uma seria realizar o acesso à API utilizando primitivas de baixo nível, recorrendo a *sockets*, o que iria necessitar de

²http://code.google.com/apis/predict/docs/general_discussion_forum.html?place=topic/prediction-api-discuss/zM150CxOlQc/discussion

um razoável esforço de desenvolvimento. Outra passaria por recorrer a uma ferramenta externa que consiga lidar com estes endereços, uma solução mais viável.

Foi assim escolhido o utilitário *oacurl*³, lançado num processo externo, ao qual o JSON de entrada é injectado no seu *standard input stream* e do qual o JSON de saída é capturado do seu *standard output stream*; tem a vantagem adicional de implementar o protocolo *OAuth*, exigido para o acesso à API.

Infelizmente este utilitário corre sobre a *framework* Java, pelo que um dos objectivos que levaram à escolha da *Google Prediction API* acabou por ser derrotado. Caso a Google resolva o problema num curto/médio prazo (permitindo a especificação do modelo a utilizar no JSON submetido e não no endereço, como acontece com o pedido de início de treino, por exemplo), uma solução integralmente desenvolvida em .NET passa a ser possível antes do site do caso em estudo ficar online. As chamadas a esta API são também neste momento um pouco demoradas devido à utilização desta ferramenta externa, devendo este problema desaparecer a partir do momento em que for possível utilizar exclusivamente .NET.

O programa completo incorporando os elementos discutidos nesta secção pode ser encontrado no Apêndice A.13.

5.6 Incorporação do Mecanismo de Sugestões Estático

Todos os dados consolidados no Capítulo 4 podem agora ser incorporados. Esta incorporação passa por três fases: atribuição de pesos, extracção da base de dados e consideração na elaboração do *ranking* de POI.

Embora todos os pesos possam ser alterados em *run time*, é necessário a atribuição de um valor inicial, subjectivo e produto da intuição, para que não seja exigido ao sistema que utilizar esta biblioteca a especificação de todos eles. A Tabela 5.5 compila os valores utilizados até agora e aqueles relativos aos componentes de sugestão estáticos.

Peso	Valor
Aleatório	1,0
Distância	2,0
Pilares	5,0
Predição	5,0
Relevância Regional	8,0
Relevância Local	3,0
Popularidade Flickr	3,0
Classificação Google Places	2,0

Tabela 5.5: Pesos utilizados para os diversos componentes.

³<http://code.google.com/p/oacurl/>

A extracção da base de dados para um objecto POI é também feita à semelhança do que tem sido feito. A Listagem 5.5 ilustra o procedimento.

```
Id = (int)dr["id"],
Coords = dr["coords"] as SqlGeography,
Category = (int)dr["id_Categoria"],
Subcategory = subcategories
    .Single(sc =>
        (sc.Id == (int)dr["id_Subcategoria"])),
RegionalRelevance =
    (dr["relevancia_regional"] is DBNull) ?
    0 :
    (int)dr["relevancia_regional"] % 10,
LocalRelevance =
    (dr["relevancia_local"] is DBNull) ?
    0 :
    (int)dr["relevancia_local"],
FlickrPopularity =
    (dr["flickr"] is DBNull) ?
    0.0 :
    (double)dr["flickr"],
PlacesRating =
    (dr["places"] is DBNull) ?
    0.0 :
    (double)dr["places"],
```

Listagem 5.5: Extracção dos valores da base de dados.

Por fim, a consideração destes valores no *ranking* também é feita à semelhança dos componentes anteriores, como se pode ver pela Listagem 5.6.

```
Poi = poi,
RandomComponent = this.rnd.NextDouble(),
DistanceComponent =
    1 - travelPath.STDistance(poi.Coords).Value /
    this.Distance,
PillarComponent = poi
    .Subcategory
    .Pillars
    .Select(scPillar => pillars[scPillar - 1])
    // Lidar com caso de não haver pilar associado.
    .Concat(new int[] { 0 }).Max() / 3.0,
PredictionComponent = catClass[poi.Category - 1],
```

```
RegionalRelevanceComponent =
    poi.RegionalRelevance / 3.0,
LocalRelevanceComponent =
    poi.LocalRelevance / 2.0,
FlickrComponent = poi.FlickrPopularity,
PlacesComponent =
    poi.PlacesRating / 5.0,
```

Listagem 5.6: Consideração dos valores no *ranking*.

A implementação desta lógica pode ser encontrada no Apêndice A.14.

5.7 Consideração das Distâncias em Estrada

Chegados a um conjunto ordenado de POI segundo uma classificação que leva em conta a média pesada de todos os componentes, há que finalmente considerar as distâncias em estrada, em oposição às em linha recta, utilizadas até aqui. Estas são particularmente importantes no caso de estudo pois, embora dois pontos situados em margens diferentes do Douro sejam separados por algumas dezenas de metros, caso não exista uma ponte próxima, a distância em estrada acaba por ser muito superior.

No entanto, enquanto as segundas são computacionalmente muito rápidas, as primeiras requerem um tempo de cálculo comparativamente mais elevado. Seguindo a prática de usar a infra-estrutura associada ao Google Maps, acresce o *lag* provocado pelas consultas a um servidor de cálculo de distâncias remoto. Adicionalmente, a utilização desta API está fortemente restringida neste caso, permitindo um máximo de 2500 pedidos por dia na versão gratuita⁴.

Podem ser tomadas algumas medidas para reduzir estes problemas, nomeadamente utilizar uma *cache* guardando as distâncias já calculadas, fazer simplificações como assumir que a distância entre a origem e o destino é a mesma que entre o destino e a origem (isto é, ignorar sentidos únicos) e reduzir o número de POI considerados para um valor reduzido (30, por exemplo), mas é preciso contar sempre com a indisponibilidade do serviço.

Torna-se necessário modificar a base de dados por forma a que esta possa acolher a *cache* de distâncias, para o que uma nova tabela é suficiente, como a apresentada na Figura 5.4; a sua definição pode ser encontrada no Apêndice A.15.

Caso o serviço de cálculo de distâncias não esteja disponível, ou um limite tenha sido atingido, é possível pelo menos proporcionar uma estimativa com base na distância em linha recta e nas distâncias em estrada armazenadas na base dados:

⁴<http://code.google.com/apis/maps/documentation/distancematrix/#Limits>

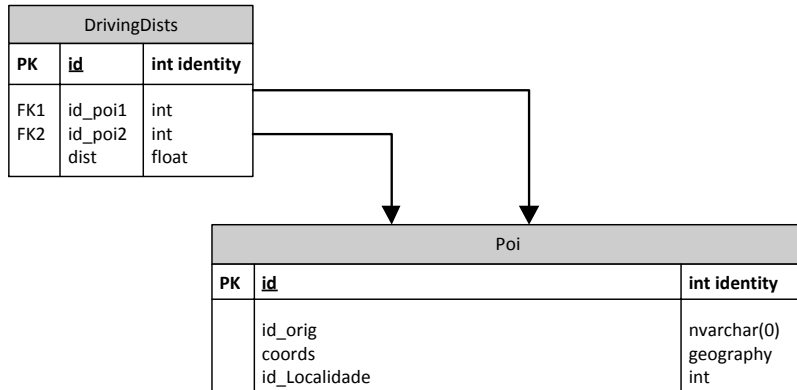


Figura 5.4: Pormenor do modelo de dados modificado, onde é possível observar a tabela destinada a receber a *cache* das distâncias em estrada.

$$d'(p, q) \approx d(p, q) \frac{\sum_{i=1}^n d'(p_i, q_i)}{\sum_{i=1}^n d(p_i, q_i)}$$

Onde $d(p, q)$ e $d'(p, q)$ representam respectivamente a distância em linha recta e em estrada entre os pontos p e q ; pares de pontos cujas distâncias em estrada estão disponíveis na *cache* são representados por p_i e q_i .

Uma vez implementada esta funcionalidade, há que abordar a forma de encontrar a distância entre uma sugestão e o itinerário do utilizador. Se for considerada a Figura 5.5, pode ser visualizado o caso em que o desvio (a tracejado) é menor.

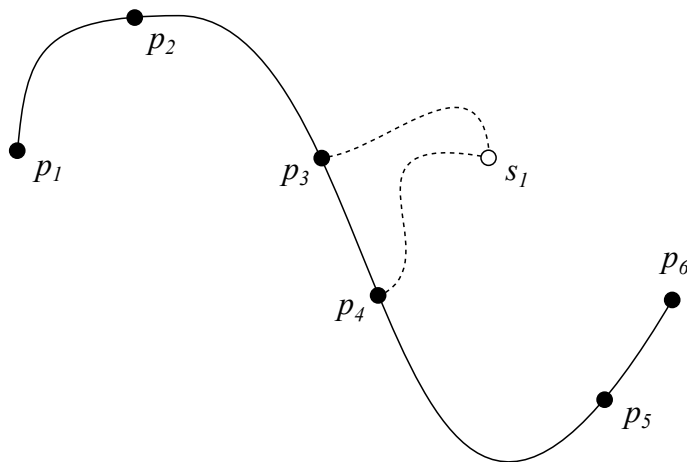


Figura 5.5: Um percurso original (linha sólida) e o desvio (a tracejado) necessário para visitar a sugestão s_1 .

A forma de calcular este desvio é bastante simples:

$$d'(p_3, s_1) + d'(s_1, p_4) - d'(p_3, p_4)$$

Generalizando,

$$\min_{i=1}^{n-1} (d'(p_i, s_j) + d'(s_j, p_{i+1}) - d'(p_i, p_{i+1}))$$

Por último, é preciso considerar os casos em que a menor distância tem lugar quando a visita ao ponto sugerido se realiza antes ou depois do itinerário; a fórmula final é encontrada:

$$\min(d'(s_j, p_1), \min_{i=1}^{n-1} (d'(p_i, s_j) + d'(s_j, p_{i+1}) - d'(p_i, p_{i+1})), d'(p_n, s_j))$$

A implementação desta lógica pode ser encontrada no Apêndice [A.16](#).

5.8 Conclusão

A implementação de mecanismos de sugestão dinâmicos é feita incrementalmente e começa pelo caso mais simples: a escolha aleatória de pontos de interesse. Esta estratégia consegue ir mostrando ao utilizador possibilidades diferentes à medida que vai interagindo com o site. Contudo, por se correr o risco de propor POI muito afastados do percurso planeado, é imposto um limite à distância máxima a que os mesmos possam estar para serem considerados. Feita esta redução do espaço de pesquisa, passa-se então a priorizar os pontos mais próximos do itinerário definido pelo utilizador, em detrimento dos mais afastados.

São então considerados os interesses explicitamente demonstrados pelo utilizador relativamente a vários pilares, sendo dada preferência aos pontos de interesse relacionados com os pilares mais favorecidos. De seguida, um modelo de *machine learning*, treinado com percursos reconstruídos a partir das fotos disponíveis no site Flickr, é utilizado para proporcionar sugestões relativas às categorias em que o utilizador possa estar interessado, dados os tipos de POI que escolheu. Por fim, são consideradas as distâncias em estrada de um pequeno subconjunto de sugestões, dada a reduzida velocidade com que é possível obter estes valores.

Todos estes componentes, em conjunto com os provenientes dos mecanismos de sugestão estáticos são finalmente agregados de uma forma linear, passando pela atribuição de um peso a cada um e selecção das sugestões com a maior classificação determinada.

Mecanismo de Sugestões Dinâmico

Capítulo 6

Conclusões e Trabalho Futuro

No âmbito da construção de uma plataforma web para o turismo e que possa ser enquadrada no âmbito de qualquer região, surgiu o problema de como proporcionar sugestões aos potenciais turistas que estejam a planear uma viagem. Embora os sistemas de recomendação estejam bastante disseminados, nenhuma solução existente é suficientemente adequada de forma a considerar o contexto geográfico como era pretendido.

Assim, foi feito um levantamento do estado da arte nas áreas do *e-tourism*, filtragem colaborativa, concepção de sites turísticos e ferramentas de *trip planning*. De seguida foi feito um planeamento da implementação do sistema de recomendação, que se pretendia realizada de forma iterativa, onde a cada ciclo seria introduzida a consideração de mais um factor de decisão.

Também se chegou à conclusão que existem dois tipos de factores: aqueles que apenas dependem de condições que podem ser definidas *a priori* e aqueles que, pelo contrário, dependem do perfil e das acções do utilizador. Os primeiros podem ser considerados *estáticos* e podem ficar armazenados numa coluna na base de dados, enquanto que os segundos podem ser considerados *dinâmicos* e precisam de ser recalculados a cada alteração no perfil ou itinerário do utilizador.

No caso dos factores estáticos, foi abordada primeiramente a possibilidade de associar cada foto georreferenciada na região alvo disponível no Flickr com um ponto de interesse e, considerando uma distância limite, tentar obter uma medida do seu interesse turístico a partir do número de fotos a que cada POI ficou associado. De seguida foi descarregada a informação disponível no serviço Google Places embora, para a região do caso de estudo, tenha um número de POI com classificações muito reduzido. A informação de melhor qualidade acaba por ser compilada por técnicos conhecedores da região e contratados para o efeito.

Quanto a factores dinâmicos, começou primeiramente por ser considerada a escolha aleatória de pontos, que permite expor o utilizador a uma maior variedade de sugestões do que se fosse utilizado um conjunto de critérios fixo. Depois foi introduzida a restrição das sugestões terem de estar a uma distância limite do itinerário que une os POI seleccionados para visita pelo utilizador. Numa primeira fase, foram consideradas as distâncias geodésicas com vista a dar mais realce aos pontos de interesse mais próximos do percurso declarado; numa segunda são mesmo utilizadas as distâncias em estrada, mas somente para um pequeno conjunto de pontos que se tenha realçado no conjunto dos restantes critérios, devido ao tempo que demora a obtenção destes valores, que consegue justificar a utilização de uma *cache*.

É também utilizado o perfil declarado do utilizador para dar mais realce aos POI que se enquadrem mais nos gostos deste. Quanto ao perfil implícito, este é extrapolado a partir do tipo de POI que foram seleccionados para o itinerário utilizando uma solução de *machine learning*; para isso, foram extraídos percursos disponíveis no Flickr e a categorização dos vários pontos de interesse que os constituem foi utilizada como modelo de treino para a Google Prediction API, sendo esse modelo utilizado para descobrir que tipo de sugestões o utilizador poderá estar interessado em receber.

Todos estes critérios são então combinados, cada um com um peso específico e são finalmente apresentados ao utilizador os resultados com melhor classificação final, conseguindo assim obter uma solução que satisfaz todos os objectivos propostos.

A reconstrução de percursos através de fotos no Flickr, subsequente cruzamento desta informação com uma base de dados de POI e finalmente a geração de dados para treino para a Google Prediction API é uma das principais inovações deste trabalho, demonstrando o poder decorrente do cruzamento de tecnologias díspares.

Várias direcções podem ser tomadas a partir deste trabalho. Uma das suas maiores fragilidades é a muito reduzida quantidade de dados existentes com que trabalha. Talvez recorrendo a bases de dados de POI mais amplas (relativas à totalidade do país, ou à Península Ibérica, por exemplo) seja possível treinar com mais precisão a Google Prediction API, mesmo que depois as predições sejam feitas somente para uma região. Há a possibilidade de passar a ser possível fazer predições relativas às subcategorias, uma granularidade mais interessante.

Foi também aqui proposto um modelo para albergar classificações por parte dos utilizadores, mas é necessário adicionar toda a lógica ao site para isso de facto aconteça. O passo seguinte será a incorporação das votações neste mecanismo de recomendação, naturalmente com um peso específico, tal como os restantes.

As viagens que os vários utilizadores vão construindo também devem ser utilizadas para treinar o modelo, possivelmente com um peso maior que as actuais predições, dada a ausência de incertezas na determinação de um POI. Bastarão pequenas modificações ao programa para geração do ficheiro de treino apresentado.

Conclusões e Trabalho Futuro

Outras bases de dados com API de acesso existem, desde o Panoramio para fotos até à Wikipedia, que começa a ter entradas georreferenciadas. Talvez seja possível utilizar estas fontes ou outras, da mesma forma como foi feito com o Flickr e o Google Places.

Conclusões e Trabalho Futuro

Referências

- [BC06] Srikanth Beldona e Liping A. Cai. An exploratory evaluation of rural tourism websites. *Journal of Convention & Event Tourism*, 8(1):69–80, 2006. http://www.buec.udel.edu/beldona/refereedarticles_files/Rural%20Tourism%20Submission.pdf.
- [BLFM98] T. Berners-Lee, R. Fielding e L. Masinter. RFC 2396: Uniform Resource Identifiers (URI): Generic syntax, August 1998. Status: DRAFT STANDARD.
- [Bra09] Brandia Central. Estudo de Avaliação da Atractividade dos Destinos Turísticos de Portugal Continental para o Mercado Interno. pages 21–31, 2009. <http://www.turismodeportugal.pt/Portugu%C3%AAs/ProTurismo/destinos/destinostur%C3%ADsticos/Anexos/Douro.pdf>.
- [CD11] António Coelho e Leonel Dias. A mobile advertising platform for etourism. In Rob Law, Matthias Fuchs e Francesco Ricci, editors, *Information and Communication Technologies in Tourism 2011: Proceedings of the International Conference in Innsbruck, Austria*, ENTER. Springer, 2011.
- [CDF⁺10] G. Castellano, D. Dell’Agnello, A.M. Fanelli, C. Mencar e M.A. Torsello. A competitive learning strategy for adapting fuzzy user profiles. In *Intelligent Systems Design and Applications (ISDA), 2010 10th International Conference on*, pages 959–964, 2010. doi:10.1109/ISDA.2010.5687063.
- [Cou10] Geoff Coupe. Geotagging in windows live photo gallery—part 2. *Reflections on life at “De Witte Wand”*, 2010. <http://gcoupe.blogspot.com/2010/08/geotagging-in-windows-live-photo.html>.
- [CSdVR10] Maarten Clements, Pavel Serdyukov, Arjen P. de Vries e Marcel J.T. Rein- ders. Using flickr geotags to predict user travel behaviour. In *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR ’10, pages 851–852, New York, NY, USA, 2010. ACM. doi:10.1145/1835449.1835648.
- [Hal00] Colin Michael Hall. *Wine tourism around the world : development, management, and markets*. Butterworth-Heinemann, Oxford ; Boston, 2000.
- [HKW10] Richard W. Hallett e Judith Kaplan-Weinger. *Official tourism websites : a discourse analysis perspective*. Tourism and cultural change. Channel View Publications, Bristol, England ; Buffalo, 2010.

REFERÊNCIAS

- [IdGLS09] L. Iaquina, M. de Gemmis, P. Lops e G. Semeraro. Recommendations toward serendipitous diversions. In *Intelligent Systems Design and Applications, 2009. ISDA '09. Ninth International Conference on*, pages 1049–1054, 2009. doi:10.1109/ISDA.2009.215.
- [KMPA03] Harald Kunczier, Elke Michlmayr, Günther Pospischil e Hermann Anegg. Lol@ – a prototype of a network independent wireless internet service. In *5th International Symposium on Advanced Radio Technologies*, March 2003. http://www.its.bldrdoc.gov/isart/art03/slides03/kun_h/kun_paper.pdf.
- [Kor09] Yehuda Koren. The BellKor Solution to the NetFlix Grand Prize. August 2009. http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf.
- [LJB01] Gregory D. Linden, Jennifer A. Jacobi e Eric A. Benson. Collaborative recommendations using item-to-item similarity mappings. Patent, July 2001. US 6266649.
- [LM05] Daniel Lemire e Anna Maclachlan. Slope one predictors for online rating-based collaborative filtering. In Hillol Kargupta, Society for Industrial and Applied Mathematics, Jaideep Srivastava, Chandrika Kamath e Arnold Goodman, editors, *Proceedings of the Fifth SIAM International Conference on Data Mining*, number v. 119 in SIAM Proceedings series, pages 471–475. Society for Industrial and Applied Mathematics, 2005.
- [LS10] F.P. Lousame e E. Sanchez. Multicriteria predictors using aggregation functions based on item views. In *Intelligent Systems Design and Applications (ISDA), 2010 10th International Conference on*, pages 947–952, 2010. doi:10.1109/ISDA.2010.5687065.
- [MP09] A. Maratea e A. Petrosino. An heuristic approach to page recommendation in web usage mining. In *Intelligent Systems Design and Applications, 2009. ISDA '09. Ninth International Conference on*, pages 1043–1048, 2009. doi:10.1109/ISDA.2009.252.
- [MR07] Tariq Mahmood e Francesco Ricci. Learning and adaptivity in interactive recommender systems. In Maria L. Gini, Robert J. Kauffman, Donna Sarppa, Chrysanthos Dellarocas e Frank Dignum, editors, *ICEC*, volume 258 of *ACM International Conference Proceeding Series*, pages 75–84. ACM, 2007. doi:10.1145/1282100.1282114.
- [MR08] Tariq Mahmood e Francesco Ricci. Adapting the interaction state model in conversational recommender systems. In Dieter Fensel e Hannes Werthner, editors, *ICEC*, volume 342 of *ACM International Conference Proceeding Series*, page 33. ACM, 2008. doi:10.1145/1409540.1409584.
- [MRV09] Tariq Mahmood, Francesco Ricci e Adriano Venturini. Learning adaptive recommendation strategies for online travel planning. In Wolfram

REFERÊNCIAS

- Höpken, Ulrike Gretzel e Rob Law, editors, *Information and Communication Technologies in Tourism 2009*, pages 149–160. Springer Vienna, 2009. doi:10.1007/978-3-211-93971-0_13.
- [MTD⁺09] C. Mencar, M.A. Torsello, D. Dell’Agnello, G. Castellano e C. Castiello. Modeling user preferences through adaptive fuzzy profiles. In *Intelligent Systems Design and Applications, 2009. ISDA ’09. Ninth International Conference on*, pages 1031–1036, 2009. doi:10.1109/ISDA.2009.67.
- [ORN⁺11] Lino Oliveira, André Rodrigues, Hélder Nunes, Leonel Dias, António Coelho, José Manuel Oliveira, Eurico Carrapatoso e Mário Jorge Leitão. Plataforma web de informação geográfica para o turismo. In *6ª Conferência Ibérica de Sistemas e Tecnologias de Informação*, CISTI, 2011.
- [PBMW99] Lawrence Page, Sergey Brin, Rajeev Motwani e Terry Winograd. The page-rank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. <http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf>.
- [PC09] Martin Piotte e Martin Chabbert. The Pragmatic Theory solution to the Netflix Grand Prize. August 2009. http://www.netflixprize.com/assets/GrandPrize2009_BPC_PragmaticTheory.pdf.
- [PCM07] Bing Pan, John C. Crotts e Brian Muller. Developing web-based tourist information tools using google map. In Marianna Sigala, Luisa Mich e Jamie Murphy, editors, *Information and Communication Technologies in Tourism 2007*, pages 503–512. Springer Vienna, 2007. <http://www.ota.cofc.edu/pan/PanCrottsMullerDevelopingGoogleMap.pdf>.
- [SWR⁺02] Steffen Staab, Hannes Werthner, Francesco Ricci, Alexander Zipf, Ulrike Gretzel, Daniel R. Fesenmaier, Cécile Paris e Craig A. Knoblock. Intelligent systems for tourism. *IEEE Intelligent Systems*, 17(6):53–64, 2002. <http://www.inf.unibz.it/~ricci/papers/RicciIEEEIntSys.pdf>.
- [TJB09] Andreas Töscher, Michael Jahrer e Robert M. Bell. The Big-Chaos Solution to the Netflix Grand Prize. September 2009. http://www.netflixprize.com/assets/GrandPrize2009_BPC_BigChaos.pdf.
- [Tox] Anders Toxboe. Shopping cart design pattern. *UI-Patterns.com*. <http://ui-patterns.com/patterns/ShoppingCart>.
- [Wor10] World Travel & Tourism Council. Portugal. *Travel & Tourism Economic Impact 2010*, 2010. http://www.wttc.org/bin/pdf/original_pdf_file/portugal.pdf.
- [ZZXY10] Vincent W. Zheng, Yu Zheng, Xing Xie e Qiang Yang. Collaborative location and activity recommendations with gps history data. In *Proceedings of the 19th international conference on World wide web, WWW ’10*, pages 1029–1038, New York, NY, USA, 2010. ACM. doi:10.1145/1772690.1772795.

REFERÊNCIAS

Anexo A

Listagens e Tabelas de Apoio

A.1 Tabela Flickr

A definição da tabela destinada a acolher a informação proveniente do site Flickr.

```
CREATE TABLE DOUROREGION.Flickr (  
    id nvarchar(255) NOT NULL PRIMARY KEY,  
    tit nvarchar(MAX) NOT NULL,  
    coords geography NOT NULL,  
    userid nvarchar(MAX) NOT NULL,  
    dateTaken datetime2,  
    id_poi int FOREIGN KEY REFERENCES DOUROREGION.Poi(id),  
    dist float  
);
```

A.2 Flickr Dump

O objectivo deste pequeno utilitário é popular a base de dados com informação relativa às fotos da Região do Douro que estão disponíveis no Flickr. Requer uma *API key* (que pode ser obtida no próprio site¹) e a biblioteca Flickr.NET².

No início do método `Main()`, podem ser necessárias alterações à *connection string*; a *API key* (aqui exibindo somente asteriscos) também precisa de ser substituída; para uma outra região, a *bounding box* também precisa de ser modificada. Embora haja um limite de 4000 resultados por pesquisa, a decomposição da pesquisa por anos permite ultrapassá-lo; se a região tivesse um número muito superior de fotos, a decomposição poderia ser feita por espaços temporais mais pequenos (meses ou semanas, por exemplo).

```
using System;  
using System.Collections.Generic;  
using System.Data;  
using System.Data.SqlClient;
```

¹<http://www.flickr.com/services/api/keys/>

²<http://www.codeplex.com/Wiki/View.aspx?ProjectName=FlickrNet>

Listagens e Tabelas de Apoio

```
using FlickrNet;
using Microsoft.SqlServer.Types;

class FlickrDump
{
    private static SqlConnectionStringBuilder builder;
    private static Flickr flickr;

    static void Main(string[] args)
    {
        builder = new SqlConnectionStringBuilder();
        // Pode necessitar de alterações.
        builder["Server"] = "(local)";
        builder["Database"] = "wmcmDouroRegionV3pt";
        builder["Integrated Security"] = true;

        // É possível pedir uma chave em:
        // http://www.flickr.com/services/api/keys/
        string APIkey = "*****";
        flickr = new Flickr(APIkey);

        BoundaryBox regioaDouro = new BoundaryBox();
        regioaDouro.MinimumLongitude = -8.115298;
        regioaDouro.MinimumLatitude = 40.7474;
        regioaDouro.MaximumLongitude = -6.191301;
        regioaDouro.MaximumLatitude = 41.734698;

        // A API requer um agente limitativo nas pesquisas; o mais antigo
        // upload de uma foto geotagged da Região do Douro remonta a 2004.
        for (int y = 2004; y <= DateTime.Now.Year; y++)
        {
            DumpYearBBox(y, regioaDouro);
        }
    }

    /// <summary>
    /// Carrega na base de dados todos os registos relativos a um
    /// determinado ano e bounding box. Uma única pesquisa devolve 4000
    /// resultados no máximo; ao fazer uma subpesquisa por ano é possível
    /// ultrapassar este limite.
    /// </summary>
    /// <param name="year">Ano da subpesquisa</param>
    /// <param name="bbox">Bounding box</param>
    private static void DumpYearBBox(int year, BoundaryBox bbox)
    {
        Console.WriteLine(String.Format("A descarregar o ano {0}.", year));
        PhotoSearchOptions pso = new PhotoSearchOptions();
        pso.BoundingBox = bbox;
        pso.MinUploadDate = new DateTime(year, 1, 1, 0, 0, 0, 0);
        pso.MaxUploadDate = new DateTime(year, 12, 31, 23, 59, 59, 999);
        if (pso.MaxUploadDate > DateTime.Now)
        {
            pso.MaxUploadDate = DateTime.Now;
        }
        pso.Extras = PhotoSearchExtras.DateTaken | PhotoSearchExtras.Geo;
        pso.Page = 1;
    }
}
```

Listagens e Tabelas de Apoio

```
using (SqlConnection conn =
    new SqlConnection(builder.ConnectionString))
{
    conn.Open();
    SqlDataAdapter da =
        new SqlDataAdapter(@"
SELECT *
FROM DOUROREGION.Flickr;",
        conn);
    DataSet ds = new DataSet();
    da.Fill(ds, "DOUROREGION.Flickr");
    DataTable dt = ds.Tables["DOUROREGION.Flickr"];
    PhotoCollection douroPhotos;
    HashSet<string> ids = new HashSet<string>();
    do
    {
        douroPhotos = flickr.PhotosSearch(pso);
        foreach (Photo photo in douroPhotos)
        {
            // A API pode devolver duplicados.
            if (!ids.Contains(photo.PhotoId))
            {
                ids.Add(photo.PhotoId);
                DataRow newRow = dt.NewRow();
                newRow["id"] = photo.PhotoId;
                newRow["tit"] = photo.Title;
                newRow["coords"] = SqlGeography.Point(
                    photo.Latitude,
                    photo.Longitude,
                    4326); // SRID EPSG:4326
                newRow["userid"] = photo.UserId;
                newRow["dateTaken"] = photo.DateTaken;
                dt.Rows.Add(newRow);
            }
        }
        Console.WriteLine(String.Format("{0}/{1}... feito.",
            pso.Page,
            douroPhotos.Pages));
    } while (pso.Page++ < douroPhotos.Pages);
    SqlCommandBuilder cb = new SqlCommandBuilder(da);
    da.UpdateCommand = cb.GetUpdateCommand();
    da.Update(dt);
}
}
```

A.3 Qualidade das Associações POI/Flickr

```
using System;
using System.Data;
```

Listagens e Tabelas de Apoio

```
using System.Data.SqlClient;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Text;

class MatchesQuality
{
    static void Main(string[] args)
    {
        SqlConnectionStringBuilder builder =
            new SqlConnectionStringBuilder();
        // Pode necessitar de alterações.
        builder["Server"] = "(local)";
        builder["Database"] = "wmcmDouroRegionV3pt";
        builder["Integrated Security"] = true;
        using (SqlConnection conn =
            new SqlConnection(builder.ConnectionString))
        {
            conn.Open();
            SqlDataAdapter da = new SqlDataAdapter(@"
SELECT f.tit Flickr, p.nome POI, f.dist Distancia
FROM DOUROREGION.Flickr f JOIN DOUROREGION.Poi p ON f.id_poi = p.id
ORDER BY dist;",
            conn);
            DataSet ds = new DataSet();
            da.Fill(ds);
            DataTable dt = ds.Tables[0];
            char[] separators = new Char[] { ',', ' ', '_', '-', '/' };
            using (TextWriter tw = new StreamWriter("matches.csv"))
            {
                var foo = ds.Tables[0]
                    .AsEnumerable()
                    .Select(dr => new
                    {
                        Flickr = dr["Flickr"].ToString(),
                        FlickrSet = RemoveDiacritics(dr["Flickr"].ToString())
                            .ToLower()
                            .Split(
                                separators,
                                StringSplitOptions.RemoveEmptyEntries)
                            .Where(token => token.Length > 3),
                        Poi = dr["POI"].ToString(),
                        PoiSet = RemoveDiacritics(dr["POI"].ToString())
                            .ToLower()
                            .Split(
                                separators,
                                StringSplitOptions.RemoveEmptyEntries)
                            .Where(token => token.Length > 3),
                        Dist = (double)dr["Distancia"],
                    })
                    .Select(tuple => new
                    {
                        Flickr = tuple.Flickr,
                        Poi = tuple.Poi,
```

Listagens e Tabelas de Apoio

```
        Dist = tuple.Dist,
        IsMatch = tuple
            .FlickrSet
            .Intersect(tuple.PoiSet)
            .Any(),
    });
    foreach (var tuple in foo)
    {
        tw.Write(' ');
        tw.Write(tuple.Flickr);
        tw.Write("\n\n");
        tw.Write(tuple.Poi);
        tw.Write("\n");
        tw.Write(tuple.Dist);
        tw.Write(';');
        tw.WriteLine(tuple.IsMatch);
    }
}

public static String RemoveDiacritics(String s)
{
    String normalizedString = s.Normalize(NormalizationForm.FormD);
    StringBuilder stringBuilder = new StringBuilder();

    for (int i = 0; i < normalizedString.Length; i++)
    {
        Char c = normalizedString[i];
        if (CharUnicodeInfo.GetUnicodeCategory(c)
            != UnicodeCategory.NonSpacingMark)
        {
            stringBuilder.Append(c);
        }
    }

    return stringBuilder.ToString();
}
```

A.4 Parametrizações do sistema

Parâmetro	Página	Valor por Omissão
número mínimo de ocorrências para que uma coordenada seja considerada resultante de <i>geocoding</i>	24	6
distância máxima entre uma foto e o respectivo POI para que a associação possa ser considerada válida	27	99,4 m

A.5 Associações entre fotografias e POI

A seguinte *query* selecciona as associações entre fotos e pontos de interesse com as menores distâncias e o resultado da sua execução é apresentado na Tabela [A.1](#).

```
SELECT TOP (30) f.tit Flickr, p.nome POI, ROUND(f.dist, 3) Distancia
FROM DOUROREGION.Flickr f JOIN DOUROREGION.Poi p ON f.id_poi = p.id
ORDER BY Distancia;
```

A.6 Google Places Dump

Este programa descarrega a informação disponível no Google Places para a base de dados. Antes da sua execução, é necessário acrescentar uma coluna à tabela de POI para receber os valores, algo que pode ser conseguido através da seguinte *query*:

```
ALTER TABLE DOUROREGION.Poi
ADD places float;
```

Adicionalmente, necessita da biblioteca `Json.NET`³ e de uma *API Key*⁴.

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Globalization;
using System.Linq;
using System.Net;
using System.Text;
using Microsoft.SqlServer.Types;
using Newtonsoft.Json.Linq;

class GooglePlacesDump
{
    // É possível pedir uma chave em
    // https://code.google.com/apis/console/
    const string KEY = "*****";
    const double RADIUS = 99.4;
    const string LANGUAGE = "pt-PT";

    public static void Main(string[] args)
    {
        SqlConnectionStringBuilder builder =
            new SqlConnectionStringBuilder();
        // Pode necessitar de alterações.
        builder["Server"] = "(local)";
```

³<http://json.codeplex.com/>

⁴<https://code.google.com/apis/console/>

Listagens e Tabelas de Apoio

	Flickr	POI	Dist. <i>m</i>
1	Miranda_Douro_ rua	São Pedro	1,535
2	Palácio dos Távoras	Paço dos Távoras	2,241
3	cp1413 (0186+9711+E209)	Quinta da Boa Passagem	2,286
4	Fonte - Portugal	Fonte Nova da Rua Frei Domingos Vieira	2,432
5	Jardim - Portugal	Fonte Nova da Rua Frei Domingos Vieira	2,432
6	Casa - Detalle fachada	Casa de Ls Cachorros Zambargonhados	2,681
7	Miranda do Douro - Portugal	Escultura de José António Nobre	2,809
8	Old friends	Nanda - Manualidades	3,026
9	Narrow street	Nanda - Manualidades	3,026
10	Flower vases in formation	Nanda - Manualidades	3,026
11	Monumento aos Combatentes da Grande Guerra - Lamego - Portugal	Estátua do Soldado Desconhecido	3,130
12	Porto São Bento - Régua	Barriga Farta	3,162
13	Estação de Ermida	Barriga Farta	3,162
14	Mirandela: Palácio dos Távoras	Paço dos Távoras	3,327
15	Mirandela: Palácio dos Távoras	Paço dos Távoras	3,327
16	Gently caressing the hill	Longroiva	3,816
17	Mesão Frio - Portugal	Busto de Conselheiro José Maria Alpoim	3,865
18	Monumento ao Herói Milhões - Murça - Portugal	Estátua de Aníbal Augusto Milhães	3,922
19	Sé-Catedral de Miranda do Douro: traseiras	Sé de Miranda do Douro / Igreja de Miranda	4,036
20	Pelourinho de Magueija	Pelourinho da Magueijinha	4,282
21	Miranda_Douro_largo_museu	Escultura de José António Nobre	4,331
22	Miranda do Douro	Escultura de José António Nobre	4,331
23	Miranda do Douro (III)	Antiga Casa da Câmara de Miranda do Douro / Museu da Terra de Miranda	4,394
24	Miranda do Douro (I)	Antiga Casa da Câmara de Miranda do Douro / Museu da Terra de Miranda	4,394
25	Miranda do Douro (II)	Antiga Casa da Câmara de Miranda do Douro / Museu da Terra de Miranda	4,394
26	P7120107-Miranda do Douro	Antigas Portas de Entrada da Cidade	4,449
27	Lamego - Portugal	Museu de Lamego	4,683
28	Solar dos Pimentéis - Castelo Branco - Portugal	Palácio dos Pimentéis / Solar dos Pimentéis	4,970
29	Miranda do Douro	Antiga Casa da Câmara de Miranda do Douro / Museu da Terra de Miranda	5,058
30	Murça - Portugal	Porca de Murça	5,091

Tabela A.1: As associações entre fotos e POI com as menores distâncias.

Listagens e Tabelas de Apoio

```
builder["Database"] = "wmcmDouroRegionV3pt";
builder["Integrated Security"] = true;
using (SqlConnection conn =
    new SqlConnection(builder.ConnectionString))
{
    conn.Open();
    SqlDataAdapter da = new SqlDataAdapter(@"
SELECT id, nome, coords, places
FROM DOUROREGION.Poi
WHERE (id_Categoria <> 9
    OR tipo = N'Estações de caminho de ferro')
    AND relevancia_regional <> -1
    AND coords IS NOT NULL;",
        conn);
    DataSet ds = new DataSet();
    da.Fill(ds, "DOUROREGION.Poi");
    DataTable dt = ds.Tables["DOUROREGION.Poi"];
    foreach (DataRow dr in dt.Rows)
    {
        SqlGeography coords = dr["coords"] as SqlGeography;
        if (coords == null) continue;
        string name = dr["nome"].ToString();
        string reference = GetReference(coords, name);
        if (reference == null) continue;
        double rating = GetRating(reference);
        if (rating == -1) continue;
        dr["places"] = rating;
    }
    SqlCommandBuilder cb = new SqlCommandBuilder(da);
    da.UpdateCommand = cb.GetUpdateCommand();
    da.Update(dt);
}

private static string GetReference(SqlGeography coords, string name)
{
    char[] separators = new char[] { ',', ' ', '_', '-', '/' };
    string reference = null;

    Console.WriteLine("POI: " + name);
    var nameComponents = RemoveDiacritics(name)
        .ToLower()
        .Split(
            separators,
            StringSplitOptions.RemoveEmptyEntries)
        .Where(str => (str.Length > 3))
        .OrderBy(str => str);
    int nameComponentsCount = nameComponents.Count();
    WebClient wb = new WebClient();
    wb.Encoding = Encoding.UTF8;
    CultureInfo invariant = CultureInfo.InvariantCulture;
    StringBuilder query = new StringBuilder();
    query.Append("location=");
    query.Append(coords.Lat.Value.ToString(invariant));
    query.Append(',');
```

Listagens e Tabelas de Apoio

```
query.Append(coords.Long.Value.ToString(invariant));
query.Append("&radius=");
query.Append(RADIUS.ToString(invariant));
query.Append("&language=");
query.Append(LANGUAGE);
query.Append("&key=");
query.Append(KEY);
query.Append("&sensor=false");
UriBuilder ubuilder = new UriBuilder
{
    Scheme = "https",
    Host = "maps.googleapis.com",
    Path = "maps/api/place/search/json",
    Query = query.ToString(),
};
string placesFoundStr;
try
{
    placesFoundStr = wb.DownloadString(ubuilder.Uri);
}
catch (WebException)
{
    return null;
}
JsonObject placesFound = JObject.Parse(placesFoundStr);
double minDist = Double.MaxValue;
foreach (JsonObject place in placesFound["results"])
{
    string placeName = place["name"].ToString();
    var placeNameComponents = RemoveDiacritics(placeName)
        .ToLower()
        .Split(
            separators,
            StringSplitOptions.RemoveEmptyEntries)
        .Where(str => (str.Length > 3))
        .OrderBy(str => str);
    if (nameComponents.SequenceEqual(placeNameComponents))
    {
        JToken placeLocation = place["geometry"]["location"];
        SqlGeography placeCoords = SqlGeography.Point(
            (double)placeLocation["lat"],
            (double)placeLocation["lng"],
            4326);
        double distance = coords.STDistance(placeCoords).Value;
        if (distance < minDist)
        {
            minDist = distance;
            reference = place["reference"].ToString();
            Console.WriteLine(
                String.Format("Match: \"{0}\"; distance: {1}",
                    placeName,
                    distance));
        }
    }
}
```

Listagens e Tabelas de Apoio

```
        return reference;
    }

private static double GetRating(string reference)
{
    WebClient wb = new WebClient();
    wb.Encoding = Encoding.UTF8;
    StringBuilder query = new StringBuilder();
    query.Append("reference=");
    query.Append(reference);
    query.Append("&language=");
    query.Append(LANGUAGE);
    query.Append("&key=");
    query.Append(KEY);
    query.Append("&sensor=false");
    UriBuilder ubuilder = new UriBuilder
    {
        Scheme = "https",
        Host = "maps.googleapis.com",
        Path = "maps/api/place/details/json",
        Query = query.ToString(),
    };
    string detailsStr;
    try
    {
        detailsStr = wb.DownloadString(ubuilder.Uri);
    }
    catch (WebException)
    {
        return -1.0;
    }
    JObject details = JObject.Parse(detailsStr);
    double rating = -1.0;
    if (details["status"].ToString() == "OK" &&
        details["result"]
            .Children<JProperty>()
            .Any(property => (property.Name == "rating")))
    {
        rating = (double)details["result"]["rating"];
        Console.WriteLine("Rating: " + rating.ToString());
    }
    return rating;
}

public static String RemoveDiacritics(String s)
{
    String normalizedString = s.Normalize(NormalizationForm.FormD);
    StringBuilder stringBuilder = new StringBuilder();

    for (int i = 0; i < normalizedString.Length; i++)
    {
        Char c = normalizedString[i];
        if (CharUnicodeInfo.GetUnicodeCategory(c) !=
            UnicodeCategory.NonSpacingMark)
        {

```

```
        stringBuilder.Append(c);
    }
}

return stringBuilder.ToString();
}
}
```

A.7 Tabela Classificações

Esta tabela destina-se a receber as classificações que os utilizadores dão aos vários POI.

```
CREATE TABLE DOUROREGION.Classificacoes (
    id int IDENTITY NOT NULL PRIMARY KEY,
    id_poi int FOREIGN KEY REFERENCES DOUROREGION.Poi(id),
    id_utilizador int FOREIGN KEY REFERENCES DOUROREGION.Utilizador(id),
    classificacao int
);
```

A.8 Versão 1: Pontos Aleatórios

Esta é a primeira versão do recomendador, que se limita a devolver pontos escolhidos aleatoriamente.

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Linq;

namespace DouroRecommenderEngine
{
    /// <summary>
    /// Implementa o motor de sugestões, exibindo o método Suggest,
    /// principal interface de comunicação.
    /// </summary>
    public class Recommender
    {
        private SqlConnection conn;
        private Random rnd;

        /// <summary>
        /// Define qual o número de sugestões a ser devolvidas por cada
        /// chamada ao recomendador.
        /// </summary>
        public int Count { get; set; }
    }
}
```

Listagens e Tabelas de Apoio

```
/// <summary>
/// Constrói um novo objecto, parametrizado com uma connection
/// string.
/// </summary>
/// <param name="connStr">Connection string para ligação à base de
/// dados.</param>
public Recommender(string connStr)
{
    this.conn = new SqlConnection(connStr);
    this.conn.Open();
    this.rnd = new Random();
    this.Count = 5;
}

/// <summary>
/// Executa a lógica de recomendação.
/// </summary>
/// <param name="itinerary">Array com a sequência dos IDs dos POI
/// que o utilizador pretende visitar.</param>
/// <param name="pillars">Perfil do utilizador, definido como o
/// nível de interesse (0 a 3) para cada pilar.</param>
/// <returns>Array com os IDs dos pontos de interesse
/// sugeridos.</returns>
public int[] Suggest(int[] itinerary, int[] pillars)
{
    if (itinerary.Length == 0)
        return null;
    SqlCommand command = new SqlCommand(@"
SELECT p.id
FROM DOUOREGION.PoiTur p;",
        this.conn);
    SqlDataAdapter da = new SqlDataAdapter(command);
    DataSet ds = new DataSet();
    da.Fill(ds, "DOUOREGION.PoiTur");
    return ds
        .Tables["DOUOREGION.PoiTur"]
        .AsEnumerable()
        .Select(dr => (int)dr["id"])
        .Except(itinerary)
        .Select(poiId => new
        {
            Id = poiId,
            RandomComponent = this.rnd.NextDouble(),
        })
        .OrderByDescending(poi => poi.RandomComponent)
        .Select(poi => poi.Id)
        .Take(this.Count)
        .ToArray();
}
}
```

A.9 Versão 2: Na Proximidade do Itinerário

Esta é a segunda versão do recomendador, que acrescenta a retrição dos pontos considerados terem que estar a uma distância máxima do percurso do utilizador. As modificações encontram-se assinaladas a **negrito**.

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using Microsoft.SqlServer.Types;

namespace DouroRecommenderEngine
{
    /// <summary>
    /// Implementa o motor de sugestões, exibindo o método Suggest,
    /// principal interface de comunicação.
    /// </summary>
    public class Recommender
    {
        private SqlConnection conn;
        private Random rnd;

        /// <summary>
        /// Define qual o número de sugestões a ser devolvidas por cada
        /// chamada ao recomendador.
        /// </summary>
        public int Count { get; set; }

        /// <summary>
        /// Define a distância máxima a que um POI pode estar do itinerário
        /// do utilizador para poder ser recomendado.
        /// </summary>
        public double Distance { get; set; }

        /// <summary>
        /// Constrói um novo objecto, parametrizado com uma connection
        /// string.
        /// </summary>
        /// <param name="connStr">Connection string para ligação à base de
        /// dados.</param>
        public Recommender(string connStr)
        {
            this.conn = new SqlConnection(connStr);
            this.conn.Open();
            this.rnd = new Random();
            this.Count = 5;
            this.Distance = 7000.0;
        }

        /// <summary>
        /// Executa a lógica de recomendação.
        /// </summary>
    }
}
```

Listagens e Tabelas de Apoio

```
/// <param name="itinerary">Array com a sequência dos IDs dos POI
/// que o utilizador pretende visitar.</param>
/// <param name="pillars">Perfil do utilizador, definido como o
/// nível de interesse (0 a 3) para cada pilar.</param>
/// <returns>Array com os IDs dos pontos de interesse
/// sugeridos.</returns>
public int[] Suggest(int[] itinerary, int[] pillars)
{
    if (itinerary.Length == 0)
        return null;
    SqlGeography travelPath = this.BuildPath(itinerary);
    // Embora a chamada STIntersects também possa ser feita na
    // camada lógica (neste componente de recomendação), realizá-la
    // antes na camada de persistência permite tirar partido de
    // eventuais índices espaciais.
    SqlCommand command = new SqlCommand(@"
SELECT p.id
FROM DOUOREGION.PoiTur p
WHERE @buffer.STIntersects(p.coords) = 1;",
        this.conn);
    command.Parameters.Add(new SqlParameter
    {
        ParameterName = "@buffer",
        UdtTypeName = "geography",
        Value = travelPath.STBuffer(this.Distance),
    });
    SqlDataAdapter da = new SqlDataAdapter(command);
    DataSet ds = new DataSet();
    da.Fill(ds, "DOUOREGION.PoiTur");
    return ds
        .Tables["DOUOREGION.PoiTur"]
        .AsEnumerable()
        .Select(dr => (int)dr["id"])
        .Except(itinerary)
        .Select(poiId => new
        {
            Id = poiId,
            RandomComponent = this.rnd.NextDouble(),
        })
        .OrderByDescending(poi => poi.RandomComponent)
        .Select(poi => poi.Id)
        .Take(this.Count)
        .ToArray();
}

/// <summary>
/// Constrói um LineString que une os pontos que o utilizador
/// pretende visitar.
/// </summary>
/// <param name="itinerary">Array com a sequência dos IDs dos POI
/// que o utilizador pretende visitar.</param>
/// <returns>O LineString construído (ou um Point, caso o array
/// contenha apenas um ponto.)</returns>
private SqlGeography BuildPath(int[] itinerary)
{

```

```

var path = itinerary.Select(poiId => this.GetCoords(poiId));
SqlGeography first = path.First();
if (itinerary.Length == 1) return first;
SqlGeographyBuilder builder = new SqlGeographyBuilder();
builder.SetSrid(4326);
builder.BeginGeography(OpenGisGeographyType.LineString);
builder.BeginFigure(first.Lat.Value, first.Long.Value);
foreach (SqlGeography coord in path.Skip(1))
{
    builder.AddLine(coord.Lat.Value, coord.Long.Value);
}
builder.EndFigure();
builder.EndGeography();
return builder.ConstructedGeography;
}

/// <summary>
/// Obtém as coordenadas do POI da base de dados.
/// </summary>
/// <param name="poiId">Id do POI.</param>
/// <returns>Coordenadas do POI.</returns>
private SqlGeography GetCoords(int poiId)
{
    SqlCommand command = new SqlCommand(@"
SELECT coords
FROM DOUROREGION.Poi
WHERE id = @poiId;",
    this.conn);
    command.Parameters.AddWithValue("@poiId", poiId);
    return command.ExecuteScalar() as SqlGeography;
}
}

```

A.10 Versão 3: Distância como Factor de Escolha

Nesta versão é introduzida a *distância entre o percurso do utilizador e cada potencial sugestão* como factor de escolha. As modificações encontram-se assinaladas a **negrito** e as remoções a ~~rasurado~~; o corpo do método `GetCoords()` não sofreu modificações, pelo que é omitido, podendo ser encontrado no Apêndice [A.9](#).

```

using System;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using Microsoft.SqlServer.Types;

namespace DouroRecommenderEngine
{
    /// <summary>

```

```

/// Classe que mapeia algumas colunas da vista DOUROREGION.PoiTur.
/// </summary>
internal class POI
{
    public int Id { get; set; }
    public SqlGeography Coords { get; set; }

    // Métodos para permitir o uso do método Except().
    override public bool Equals(object other)
    {
        POI otherPoi = other as POI;
        return (this.Id == otherPoi.Id);
    }
    override public int GetHashCode()
    {
        return this.Id.GetHashCode();
    }
}

/// <summary>
/// Implementa o motor de sugestões, exibindo o método Suggest,
/// principal interface de comunicação.
/// </summary>
public class Recommender
{
    private SqlConnection conn;
    private Random rnd;

    /// <summary>
    /// Peso atribuído à componente aleatória.
    /// </summary>
    public double WeightRandom { get; set; }

    /// <summary>
    /// Peso atribuído à componente de distância.
    /// </summary>
    public double WeightDistance { get; set; }

    /// <summary>
    /// Define qual o número de sugestões a ser devolvidas por cada
    /// chamada ao recomendador.
    /// </summary>
    public int Count { get; set; }

    /// <summary>
    /// Define a distância máxima a que um POI pode estar do itinerário
    /// do utilizador para poder ser recomendado.
    /// </summary>
    public double Distance { get; set; }

    /// <summary>
    /// Constrói um novo objecto, parametrizado com uma connection
    /// string.
    /// </summary>
    /// <param name="connStr">Connection string para ligação à base de

```

Listagens e Tabelas de Apoio

```
/// dados.</param>
public Recommender(string connStr)
{
    this.conn = new SqlConnection(connStr);
    this.conn.Open();
    this.rnd = new Random();
    this.Count = 5;
    this.Distance = 7000.0;
    this.WeightRandom = 1.0;
    this.WeightDistance = 2.0;
}

/// <summary>
/// Executa a lógica de recomendação.
/// </summary>
/// <param name="itinerary">Array com a sequência dos IDs dos POI
/// que o utilizador pretende visitar.</param>
/// <param name="pillars">Perfil do utilizador, definido como o
/// nível de interesse (0 a 3) para cada pilar.</param>
/// <returns>Array com os IDs dos pontos de interesse
/// sugeridos.</returns>
public int[] Suggest(int[] itinerary, int[] pillars)
{
    if (itinerary.Length == 0)
        return null;
    POI[] itineraryPoi = itinerary
    .Select(poiId => new POI
    {
        Id = poiId,
        Coords = this.GetCoords(poiId),
    })
    .ToArray();
    SqlGeography travelPath = this.BuildPath(itineraryPoi);
    // Embora a chamada STIntersects também possa ser feita na
    // camada lógica (neste componente de recomendação), realizá-la
    // antes na camada de persistência permite tirar partido de
    // eventuais índices espaciais.
    SqlCommand command = new SqlCommand(@"
SELECT p.id, p.coords
FROM DOUROREGION.PoiTur p
WHERE @buffer.STIntersects(p.coords) = 1;",
    this.conn);
    command.Parameters.Add(new SqlParameter
    {
        ParameterName = "@buffer",
        UdtTypeName = "geography",
        Value = travelPath.STBuffer(this.Distance),
    });
    SqlDataAdapter da = new SqlDataAdapter(command);
    DataSet ds = new DataSet();
    da.Fill(ds, "DOUROREGION.PoiTur");
    return ds
        .Tables["DOUROREGION.PoiTur"]
        .AsEnumerable()
        .Select(dr => new POI
```

Listagens e Tabelas de Apoio

```

    {
        Id = (int)dr["id"],
        Coords = dr["coords"] as SqlGeography,
    })
    .Except(itineraryPoi)
    .Select(poi => new
    {
        Poi = poi,
        RandomComponent = this.rnd.NextDouble(),
        DistanceComponent =
            1 - travelPath.STDistance(poi.Coords).Value /
            this.Distance,
    })
    .OrderByDescending(poi =>
        this.WeightDistance * poi.DistanceComponent +
        this.WeightRandom * poi.RandomComponent)
    .Select(poi => poi.Poi.Id)
    .Take(this.Count)
    .ToArray();
}

private SqlGeography GetCoords(int poiId);

/// <summary>
/// Constrói um LineString que une os pontos que o utilizador
/// pretende visitar.
/// </summary>
/// <param name="itinerary">Array com a sequência dos POI que o
/// utilizador pretende visitar.</param>
/// <returns>O LineString construído (ou um Point, caso o array
/// contenha apenas um ponto.)</returns>
private SqlGeography BuildPath(POI[] itinerary)
{
    var path = itinerary.Select(poiId => this.GetCoords(poiId)).
    SqlGeography first = itinerary.First().Coords;
    if (itinerary.Length == 1) return first;
    SqlGeographyBuilder builder = new SqlGeographyBuilder();
    builder.SetSrid(4326);
    builder.BeginGeography(OpenGisGeographyType.LineString);
    builder.BeginFigure(first.Lat.Value, first.Long.Value);
    foreach (POI p in itinerary.Skip(1))
    {
        builder.AddLine(p.Coords.Lat.Value, p.Coords.Long.Value);
    }
    builder.EndFigure();
    builder.EndGeography();
    return builder.ConstructedGeography;
}
}

```

A.11 Versão 4: Perfil Explicitamente Definido pelo Utilizador

Nesta versão é introduzida a *consideração das preferências do utilizador relativamente aos vários pilares* como factor de escolha. As modificações encontram-se assinaladas a **negrito**; os corpos dos métodos `GetCoords()`, `Equals()`, `GetHashCode()` e `BuildPath()` não sofreram modificações, pelo que são omitidos, podendo ser encontrados nos Apêndices [A.9](#) (o primeiro) e [A.10](#) (os três últimos).

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using Microsoft.SqlServer.Types;

namespace DouroRecommenderEngine
{
    /// <summary>
    /// Classe que mapeia algumas colunas da tabela
    /// DOUROREGION.Subcategoria.
    /// </summary>
    internal class Subcategory
    {
        public int Id { get; set; }
        public int[] Pillars { get; set; }
    }

    /// <summary>
    /// Classe que mapeia algumas colunas da vista DOUROREGION.PoiTur.
    /// </summary>
    internal class POI
    {
        public int Id { get; set; }
        public SqlGeography Coords { get; set; }
        public Subcategory Subcategory { get; set; }

        override public bool Equals(object other);
        override public int GetHashCode();
    }

    /// <summary>
    /// Implementa o motor de sugestões, exibindo o método Suggest,
    /// principal interface de comunicação.
    /// </summary>
    public class Recommender
    {
        private SqlConnection conn;
        private Random rnd;

        /// <summary>
        /// Peso atribuído à componente aleatória.
        /// </summary>
        public double WeightRandom { get; set; }
    }
}
```

Listagens e Tabelas de Apoio

```
/// <summary>
/// Peso atribuído à componente de distância.
/// </summary>
public double WeightDistance { get; set; }

/// <summary>
/// Peso atribuído à componente de pilares.
/// </summary>
public double WeightPillars { get; set; }

/// <summary>
/// Define qual o número de sugestões a ser devolvidas por cada
/// chamada ao recomendador.
/// </summary>
public int Count { get; set; }

/// <summary>
/// Define a distância máxima a que um POI pode estar do itinerário
/// do utilizador para poder ser recomendado.
/// </summary>
public double Distance { get; set; }

/// <summary>
/// Constrói um novo objecto, parametrizado com uma connection
/// string.
/// </summary>
/// <param name="connStr">Connection string para ligação à base de
/// dados.</param>
public Recommender(string connStr)
{
    this.conn = new SqlConnection(connStr);
    this.conn.Open();
    this.rnd = new Random();
    this.Count = 5;
    this.Distance = 7000.0;
    this.WeightRandom = 1.0;
    this.WeightDistance = 2.0;
    this.WeightPillars = 5.0;
}

/// <summary>
/// Executa a lógica de recomendação.
/// </summary>
/// <param name="itinerary">Array com a sequência dos IDs dos POI
/// que o utilizador pretende visitar.</param>
/// <param name="pillars">Perfil do utilizador, definido como o
/// nível de interesse (0 a 3) para cada pilar.</param>
/// <returns>Array com os IDs dos pontos de interesse
/// sugeridos.</returns>
public int[] Suggest(int[] itinerary, int[] pillars)
{
    if (itinerary.Length == 0 || pillars.Length == 0)
        return null;
    POI[] itineraryPoi = itinerary
```

Listagens e Tabelas de Apoio

```

        .Select(poiId => new POI
        {
            Id = poiId,
            Coords = this.GetCoords(poiId),
        })
        .ToArray();
Subcategory[] subcategories = this.FetchSubcategories();
        SqlGeography travelPath = this.BuildPath(itineraryPoi);
        // Embora a chamada STIntersects também possa ser feita na
        // camada lógica (neste componente de recomendação), realizá-la
        // antes na camada de persistência permite tirar partido de
        // eventuais índices espaciais.
        SqlCommand command = new SqlCommand(@"
SELECT p.id, p.coords, id_Subcategoria
FROM DOUROREGION.PoiTur p
WHERE @buffer.STIntersects(p.coords) = 1;",
        this.conn);
        command.Parameters.Add(new SqlParameter
        {
            ParameterName = "@buffer",
            UdtTypeName = "geography",
            Value = travelPath.STBuffer(this.Distance),
        });
        SqlDataAdapter da = new SqlDataAdapter(command);
        DataSet ds = new DataSet();
        da.Fill(ds, "DOUROREGION.PoiTur");
        return ds
            .Tables["DOUROREGION.PoiTur"]
            .AsEnumerable()
            .Select(dr => new POI
            {
                Id = (int)dr["id"],
                Coords = dr["coords"] as SqlGeography,
                Subcategory = subcategories
                    .Single(sc =>
                        (sc.Id == (int)dr["id_Subcategoria"])),
            })
            .Except(itineraryPoi)
            .Select(poi => new
            {
                Poi = poi,
                RandomComponent = this.rnd.NextDouble(),
                DistanceComponent =
                    1 - travelPath.STDistance(poi.Coords).Value /
                    this.Distance,
                PillarComponent = poi
                    .Subcategory
                    .Pillars
                    .Select(scPillar => pillars[scPillar - 1])
                    // Lidar com caso de não haver pilar associado.
                    .Concat(new int[] { 0 }).Max() / 3.0,
            })
            .OrderByDescending(poi =>
                this.WeightDistance * poi.DistanceComponent +
                this.WeightRandom * poi.RandomComponent +

```

```

        this.WeightPillars * poi.PillarComponent)
        .Select(poi => poi.Poi.Id)
        .Take(this.Count)
        .ToArray();
    }

    private SqlGeography GetCoords(int poiId);
    private SqlGeography BuildPath(POI[] itinerary);

    /// <summary>
    /// Obtém da base de dados as subcategorias e respectivos pilares.
    /// </summary>
    /// <returns>Um array de objectos Subcategory.</returns>
    private Subcategory[] FetchSubcategories()
    {
        SqlCommand command = new SqlCommand(@"
SELECT sc.id, pasc.id_PilarActividade
FROM DOUROREGION.Subcategoria sc
    JOIN DOUROREGION.PilarActividade_Subcategoria pasc
    ON sc.id = pasc.id_Subcategoria
WHERE pasc.id_PilarActividade <= 4;",
        this.conn);
        SqlDataAdapter da = new SqlDataAdapter(command);
        DataSet ds = new DataSet();
        da.Fill(ds);
        DataTable dt = ds.Tables[0];
        return dt
            .AsEnumerable()
            .Select(dr => new
            {
                Id = (int)dr["id"],
                Pillar = (int)dr["id_PilarActividade"],
            })
            .GroupBy(scl => scl.Id)
            .Select(scg => new Subcategory
            {
                Id = scg.Key,
                Pillars = scg.Select(scl => scl.Pillar).ToArray(),
            })
            .ToArray();
    }
}

```

A.12 Geração do Ficheiro de Treino

A saída deste programa é um ficheiro com o nome “training.csv”, para treino da Google Prediction API, relacionando as categorias dos POI de cada percurso.

```
using System.Data;
```

Listagens e Tabelas de Apoio

```
using System.Data.SqlClient;
using System.Globalization;
using System.IO;
using System.Linq;

public class FlickrPoi
{
    public string PhotoId { get; set; }
    public string UserId { get; set; }
    public int Cat { get; set; }
    public int Subcat { get; set; }
}

public class GenerateTrainingFile
{
    static void Main(string[] args)
    {
        SqlConnectionStringBuilder builder =
            new SqlConnectionStringBuilder();
        // Pode necessitar de alterações.
        builder["Server"] = "(local)";
        builder["Database"] = "wmcmDouroRegionV3pt";
        builder["Integrated Security"] = true;
        using (SqlConnection conn =
            new SqlConnection(builder.ConnectionString))
        {
            conn.Open();
            SqlDataAdapter da = new SqlDataAdapter(@"
SELECT f.id AS PhotoId,
       f.userid AS UserId,
       p.id_Categoria AS Cat,
       p.id_Subcategoria AS Subcat
FROM DOUROREGION.Flickr f JOIN DOUROREGION.Poi p ON f.id_poi = p.id
WHERE f.dist <= 99.4;", conn);
            DataSet ds = new DataSet();
            da.Fill(ds);
            using (TextWriter tw = new StreamWriter("training.csv"))
            {
                var trails = ds
                    .Tables[0]
                    .AsEnumerable()
                    .Select(dr => new FlickrPoi
                    {
                        PhotoId = dr["PhotoId"].ToString(),
                        UserId = dr["UserId"].ToString(),
                        Cat = (int)dr["Cat"],
                        Subcat = (int)dr["Subcat"],
                    })
                    .GroupBy(fp => fp.UserId)
                    // A lógica não tem sentido aplicada a utilizadores que
                    // só tenham uma foto na região.
                    .Where(trail => (trail.Count() > 1));
                int[] catCounters = new int[10];
                foreach (var trail in trails)
                {
```

Listagens e Tabelas de Apoio

```
foreach (FlickrPoi FPi in trail)
{
    for (int i = 0; i < catCounters.Length; i++)
    {
        catCounters[i] = 0;
    }
    foreach (FlickrPoi FPj in trail)
    {
        if (FPi.PhotoId == FPj.PhotoId) continue;
        catCounters[FPj.Cat - 1]++;
    }
    tw.Write("\"Cat\"");
    tw.Write(FPi.Cat);
    tw.Write(' ');
    double catTotal = trail.Count() - 1.0;
    foreach (int catCount in catCounters)
    {
        tw.Write(', ');
        tw.Write(
            (catCount / catTotal).ToString(
                CultureInfo.InvariantCulture));
    }
    tw.WriteLine();
}
}
}
```

A.13 Versão 5: Consideração do Perfil Implícito do Utilizador

Nesta versão é incorporada a utilização da Google Prediction API para obter uma recomendação personalizada a nível de categorias. É necessário uma chave de acesso Google API⁵, a qual deve ser atribuída à propriedade `Key` da instância da classe `Recommender` utilizada. Adicionalmente a *framework* Java precisa de estar instalada, tendo o directório onde se encontra o ficheiro `java.exe` de fazer parte da variável de ambiente `Path`.

O ficheiro `oacurl-1.2.0.jar`⁶ tem de estar disponível num directório, cuja *path* tem de ser atribuída à propriedade `OacurlPath` da instância da classe `Recommender` utilizada. A partir da linha de comandos, nesse directório deve ser executado o comando:

```
java -cp oacurl-1.2.0.jar com.google.oacurl.Login --scope http
s://www.googleapis.com/auth/prediction
```

... e a autorização tem de ser concedida na página Google que aparece.

⁵<https://code.google.com/apis/console/>

⁶<http://code.google.com/p/oacurl/downloads/list>

O ficheiro gerado pelo programa disponível no Apêndice A.12 deve ser carregado para a plataforma Google Storage for Developers e uma sessão de treino deve ser invocada sobre ele. No site da Google é possível obter detalhes sobre este processo⁷.

O nome do modelo treinado deve ser então atribuído à propriedade `Model` da instância da classe `Recommender` utilizada (utilizando o carácter `'/'` e não `'%2F'`). Adicionalmente, necessita da biblioteca `Json.NET`⁸.

As modificações encontram-se assinaladas a **negrito**; os corpos dos métodos `GetCoords()`, `Equals()`, `GetHashCode()`, `BuildPath()` e `FetchSubcategories()` não sofreram modificações, pelo que são omitidos, podendo ser encontrados nos Apêndices A.9 (o primeiro), A.10 (os três seguintes) e A.11 (o último).

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Web;
using Microsoft.SqlServer.Types;
using Newtonsoft.Json.Linq;

namespace DouroRecommenderEngine
{
    /// <summary>
    /// Classe que mapeia algumas colunas da tabela
    /// DOUROREGION.Subcategoria.
    /// </summary>
    internal class Subcategory
    {
        public int Id { get; set; }
        public int[] Pillars { get; set; }
    }

    /// <summary>
    /// Classe que mapeia algumas colunas da vista DOUROREGION.PoiTur.
    /// </summary>
    internal class POI
    {
        public int Id { get; set; }
        public SqlGeography Coords { get; set; }
        public int Category { get; set; }
        public Subcategory Subcategory { get; set; }

        override public bool Equals(object other);
        override public int GetHashCode();
    }

    /// <summary>
    /// Implementa o motor de sugestões, exibindo o método Suggest,
```

⁷http://code.google.com/apis/predict/docs/hello_world.html

⁸<http://json.codeplex.com/>

Listagens e Tabelas de Apoio

```
/// principal interface de comunicação.
/// </summary>
public class Recommender
{
    private SqlConnection conn;
    private Random rnd;

    // Endereço de acesso à Google Prediction API
    private const string PREDICT_URL =
"https://www.googleapis.com/prediction/v1.2/training/{0}/predict?key={1}";

    /// <summary>
    /// Peso atribuído à componente aleatória.
    /// </summary>
    public double WeightRandom { get; set; }

    /// <summary>
    /// Peso atribuído à componente de distância.
    /// </summary>
    public double WeightDistance { get; set; }

    /// <summary>
    /// Peso atribuído à componente de pilares.
    /// </summary>
    public double WeightPillars { get; set; }

    /// <summary>
/// Peso atribuído à componente de predição.
/// </summary>
public double WeightPrediction { get; set; }

    /// <summary>
    /// Define qual o número de sugestões a ser devolvidas por cada
    /// chamada ao recomendador.
    /// </summary>
    public int Count { get; set; }

    /// <summary>
    /// Define a distância máxima a que um POI pode estar do itinerário
    /// do utilizador para poder ser recomendado.
    /// </summary>
    public double Distance { get; set; }

    /// <summary>
/// Directório onde se encontra o ficheiro oacurl-1.2.0.jar
/// </summary>
public string OacurlPath { get; set; }

    /// <summary>
/// Modelo treinado com as categorias dos itinerários disponíveis.
/// </summary>
public string Model { get; set; }

    /// <summary>
/// Chave para acesso à Google Prediction API.

```

Listagens e Tabelas de Apoio

```

/// </summary>
public string Key { get; set; }

/// <summary>
/// Constrói um novo objecto, parametrizado com uma connection
/// string.
/// </summary>
/// <param name="connStr">Connection string para ligação à base de
/// dados.</param>
public Recommender(string connStr)
{
    this.conn = new SqlConnection(connStr);
    this.conn.Open();
    this.rnd = new Random();
    this.Count = 5;
    this.Distance = 7000.0;
    this.WeightRandom = 1.0;
    this.WeightDistance = 2.0;
    this.WeightPillars = 5.0;
this.WeightPrediction = 5.0;
}

/// <summary>
/// Executa a lógica de recomendação.
/// </summary>
/// <param name="itinerary">Array com a sequência dos IDs dos POI
/// que o utilizador pretende visitar.</param>
/// <param name="pillars">Perfil do utilizador, definido como o
/// nível de interesse (0 a 3) para cada pilar.</param>
/// <returns>Array com os IDs dos pontos de interesse
/// sugeridos.</returns>
public int[] Suggest(int[] itinerary, int[] pillars)
{
    if (itinerary.Length == 0 || pillars.Length == 0)
        return null;
    POI[] itineraryPoi = itinerary
        .Select(poiId => new POI
        {
            Id = poiId,
            Coords = this.GetCoords(poiId),
Category = this.GetCategory(poiId),
        })
        .ToArray();
double[] catClass = this.GetCatRecommendations(itineraryPoi);
    Subcategory[] subcategories = this.FetchSubcategories();
    SqlGeography travelPath = this.BuildPath(itineraryPoi);
    // Embora a chamada STIntersects também possa ser feita na
    // camada lógica (neste componente de recomendação), realizá-la
    // antes na camada de persistência permite tirar partido de
    // eventuais índices espaciais.
    SqlCommand command = new SqlCommand(@"
SELECT *
FROM DOUROREGION.PoiTur p
WHERE @buffer.STIntersects(p.coords) = 1;",
        this.conn);

```

Listagens e Tabelas de Apoio

```
command.Parameters.Add(new SqlParameter
{
    ParameterName = "@buffer",
    UdtTypeName = "geography",
    Value = travelPath.STBuffer(this.Distance),
});
SqlDataAdapter da = new SqlDataAdapter(command);
DataSet ds = new DataSet();
da.Fill(ds, "DOUOREGION.PoiTur");
return ds
    .Tables["DOUOREGION.PoiTur"]
    .AsEnumerable()
    .Select(dr => new POI
    {
        Id = (int)dr["id"],
        Coords = dr["coords"] as SqlGeography,
        Category = (int)dr["id_Categoria"],
        Subcategory = subcategories
            .Single(sc =>
                (sc.Id == (int)dr["id_Subcategoria"])),
    })
    .Except(itineraryPoi)
    .Select(poi => new
    {
        Poi = poi,
        RandomComponent = this.rnd.NextDouble(),
        DistanceComponent =
            1 - travelPath.STDistance(poi.Coords).Value /
            this.Distance,
        PillarComponent = poi
            .Subcategory
            .Pillars
            .Select(scPillar => pillars[scPillar - 1])
            // Lidar com caso de não haver pilar associado.
            .Concat(new int[] { 0 }).Max() / 3.0,
        PredictionComponent = catClass[poi.Category - 1],
    })
    .OrderByDescending(poi =>
        this.WeightDistance * poi.DistanceComponent +
        this.WeightRandom * poi.RandomComponent +
        this.WeightPillars * poi.PillarComponent +
        this.WeightPrediction * poi.PredictionComponent)
    .Select(poi => poi.Poi.Id)
    .Take(this.Count)
    .ToArray();
}

private SqlGeography GetCoords(int poiId);
private SqlGeography BuildPath(POI[] itinerary);
private Subcategory[] FetchSubcategories();

/// <summary>
/// Obtém recomendações relativas às categorias em que o utilizador
/// possa estar interessado, com base no percurso definido.
/// </summary>
```

```

/// <param name="itinerary">Array com a sequência dos POI que o
/// utilizador pretende visitar.</param>
/// <returns>Array com os níveis de interesse (de 0,0 a 1,1) que o
/// utilizador pode ter relativamente a cada categoria.</returns>
private double[] GetCatRecommendations(POI[] itinerary)
{
    if (String.IsNullOrEmpty(this.OacurlPath) ||
        String.IsNullOrEmpty(this.Model) ||
        String.IsNullOrEmpty(this.Key))
    {
        return new double[10];
    }

    int[] catCounter = new int[10];
    foreach (POI poi in itinerary)
    {
        catCounter[poi.Category - 1]++;
    }
    double catTotal = catCounter.Sum();
    var catProportions =
        catCounter.Select(count => count / catTotal);
    JObject predictIn = JObject.FromObject(new
    {
        input = new
        {
            csvInstance = catProportions,
        }
    });

    JObject predictOut = this.InvokePrediction(predictIn);

    double[] scores = new double[10];
    foreach (JObject labelScorePair in predictOut["outputMulti"])
    {
        string label = labelScorePair["label"].ToString();
        // Remover "Cat" do início.
        int category = Int32.Parse(label.Substring(3));
        scores[category - 1] = (double)labelScorePair["score"];
    }
    // Passar classificação da categoria escolhida para 1,0.
    double max = scores.Max();
    return scores.Select(score => score / max).ToArray();
}

/// <summary>
/// Faz um pedido de predição à Google Prediction API.
/// </summary>
/// <param name="predictInput">JSON com as proporções das
/// categorias.</param>
/// <returns></returns>
private JObject InvokePrediction(JObject predictInput)
{
    Process p = new Process();
    p.StartInfo.UseShellExecute = false;
    p.StartInfo.RedirectStandardOutput = true;

```

```

        p.StartInfo.RedirectStandardInput = true;
        p.StartInfo.CreateNoWindow = true;
        p.StartInfo.WorkingDirectory = this.OacurlPath;
        p.StartInfo.FileName = "java.exe";
        StringBuilder sb = new StringBuilder();
        sb.Append("-cp ");
        sb.Append("./oacurl-1.2.0.jar ");
        sb.Append("com.google.oacurl.Fetch ");
        sb.Append("-X POST ");
        sb.Append("-t JSON ");
        sb.Append(String.Format(
            PREDICT_URL,
            HttpUtility.UrlEncode(this.Model),
            this.Key));
        sb.Append(' ');
        p.StartInfo.Arguments = sb.ToString();
        p.Start();
        p.StandardInput.WriteLine(predictInput.ToString());
        p.StandardInput.Close();
        string output = p.StandardOutput.ReadToEnd();
        p.WaitForExit();
        return JObject.Parse(output);
    }

    /// <summary>
    /// Obtém a categoria do POI da base de dados.
    /// </summary>
    /// <param name="poiId">Id do POI.</param>
    /// <returns>Categoria do POI.</returns>
    private int GetCategory(int poiId)
    {
        SqlCommand command = new SqlCommand(@"
SELECT id_Categoria
FROM DOUROREGION.Poi
WHERE id = @poiId;",
            this.conn);
        command.Parameters.AddWithValue("@poiId", poiId);
        return (int)command.ExecuteScalar();
    }
}
}

```

A.14 Versão 6: Incorporação dos Componentes de Sugestão Estáticos

Nesta versão é considerada a informação recolhida no âmbito dos mecanismos de sugestão estáticos. As modificações encontram-se assinaladas a **negrito**; os corpos dos métodos `GetCoords()`, `Equals()`, `GetHashCode()`, `BuildPath()`, `FetchSubcategories()`, `GetCatRecommendations()`, `InvokePrediction()`, `GetCategory()` não sofreram modificações, pelo que são omitidos, podendo ser encontrados

nos Apêndices [A.9](#) (o primeiro), [A.10](#) (os três seguintes), [A.11](#) (o quinto) e [A.13](#) (três restantes).

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Diagnostics;
using System.Globalization;
using System.Linq;
using System.Net;
using System.Text;
using System.Web;
using Microsoft.SqlServer.Types;
using Newtonsoft.Json.Linq;

namespace DouroRecommenderEngine
{
    /// <summary>
    /// Classe que mapeia algumas colunas da tabela
    /// DOUROREGION.Subcategoria.
    /// </summary>
    internal class Subcategory
    {
        public int Id { get; set; }
        public int[] Pillars { get; set; }
    }

    /// <summary>
    /// Classe que mapeia algumas colunas da vista DOUROREGION.PoiTur.
    /// </summary>
    internal class POI
    {
        public int Id { get; set; }
        public SqlGeography Coords { get; set; }
        public int Category { get; set; }
        public Subcategory Subcategory { get; set; }
        public int RegionalRelevance { get; set; }
        public int LocalRelevance { get; set; }
        public double FlickrPopularity { get; set; }
        public double PlacesRating { get; set; }

        override public bool Equals(object other);
        override public int GetHashCode();
    }

    /// <summary>
    /// Implementa o motor de sugestões, exibindo o método Suggest,
    /// principal interface de comunicação.
    /// </summary>
    public class Recommender
    {
        private SqlConnection conn;
```

Listagens e Tabelas de Apoio

```
private Random rnd;

// Endereço de acesso à Google Prediction API
private const string PREDICT_URL =
"https://www.googleapis.com/prediction/v1.2/training/{0}/predict?key={1}";

/// <summary>
/// Peso atribuído à componente aleatória.
/// </summary>
public double WeightRandom { get; set; }

/// <summary>
/// Peso atribuído à componente de distância.
/// </summary>
public double WeightDistance { get; set; }

/// <summary>
/// Peso atribuído à componente de pilares.
/// </summary>
public double WeightPillars { get; set; }

/// <summary>
/// Peso atribuído à componente de predição.
/// </summary>
public double WeightPrediction { get; set; }

/// <summary>
/// Peso atribuído à componente relevância regional.
/// </summary>
public double WeightRegionalRelevance { get; set; }

/// <summary>
/// Peso atribuído à componente relevância local.
/// </summary>
public double WeightLocalRelevance { get; set; }

/// <summary>
/// Peso atribuído à componente classificação popularidade Flickr.
/// </summary>
public double WeightFlickrPopularity { get; set; }

/// <summary>
/// Peso atribuído à componente classificação Google Places.
/// </summary>
public double WeightPlacesRating { get; set; }

/// <summary>
/// Define qual o número de sugestões a ser devolvidas por cada
/// chamada ao recomendador.
/// </summary>
public int Count { get; set; }

/// <summary>
/// Define a distância máxima a que um POI pode estar do itinerário
/// do utilizador para poder ser recomendado.
```

Listagens e Tabelas de Apoio

```
/// </summary>
public double Distance { get; set; }

/// <summary>
/// Directório onde se encontra o ficheiro oacurl-1.2.0.jar
/// </summary>
public string OacurlPath { get; set; }

/// <summary>
/// Modelo treinado com as categorias dos itinerários disponíveis.
/// </summary>
public string Model { get; set; }

/// <summary>
/// Chave para acesso à Google Prediction API.
/// </summary>
public string Key { get; set; }

/// <summary>
/// Constrói um novo objecto, parametrizado com uma connection
/// string.
/// </summary>
/// <param name="connStr">Connection string para ligação à base de
/// dados.</param>
public Recommender(string connStr)
{
    this.conn = new SqlConnection(connStr);
    this.conn.Open();
    this.rnd = new Random();
    this.Count = 5;
    this.Distance = 7000.0;
    this.WeightRandom = 1.0;
    this.WeightDistance = 2.0;
    this.WeightPillars = 5.0;
    this.WeightPrediction = 5.0;
    this.WeightRegionalRelevance = 8.0;
    this.WeightLocalRelevance = 3.0;
    this.WeightFlickrPopularity = 3.0;
    this.WeightPlacesRating = 2.0;
}

/// <summary>
/// Executa a lógica de recomendação.
/// </summary>
/// <param name="itinerary">Array com a sequência dos IDs dos POI
/// que o utilizador pretende visitar.</param>
/// <param name="pillars">Perfil do utilizador, definido como o
/// nível de interesse (0 a 3) para cada pilar.</param>
/// <returns>Array com os IDs dos pontos de interesse
/// sugeridos.</returns>
public int[] Suggest(int[] itinerary, int[] pillars)
{
    if (itinerary.Length == 0 || pillars.Length == 0)
        return null;
    POI[] itineraryPoi = itinerary
```

Listagens e Tabelas de Apoio

```

        .Select(poiId => new POI
        {
            Id = poiId,
            Coords = this.GetCoords(poiId),
            Category = this.GetCategory(poiId),
        })
        .ToArray();
double[] catClass = this.GetCatRecommendations(itineraryPoi);
Subcategory[] subcategories = this.FetchSubcategories();
SqlGeography travelPath = this.BuildPath(itineraryPoi);
// Embora a chamada STIntersects também possa ser feita na
// camada lógica (neste componente de recomendação), realizá-la
// antes na camada de persistência permite tirar partido de
// eventuais índices espaciais.
SqlCommand command = new SqlCommand(@"
SELECT *
FROM DOUOREGION.PoiTur p
WHERE @buffer.STIntersects(p.coords) = 1;",
    this.conn);
command.Parameters.Add(new SqlParameter
{
    ParameterName = "@buffer",
    UdtTypeName = "geography",
    Value = travelPath.STBuffer(this.Distance),
});
SqlDataAdapter da = new SqlDataAdapter(command);
DataSet ds = new DataSet();
da.Fill(ds, "DOUOREGION.PoiTur");
return ds
    .Tables["DOUOREGION.PoiTur"]
    .AsEnumerable()
    .Select(dr => new POI
    {
        Id = (int)dr["id"],
        Coords = dr["coords"] as SqlGeography,
        Category = (int)dr["id_Categoria"],
        Subcategory = subcategories
            .Single(sc =>
                (sc.Id == (int)dr["id_Subcategoria"])),
        RegionalRelevance =
            (dr["relevancia_regional"] is DBNull) ?
            0 :
            (int)dr["relevancia_regional"] % 10,
        LocalRelevance =
            (dr["relevancia_local"] is DBNull) ?
            0 :
            (int)dr["relevancia_local"],
        FlickrPopularity =
            (dr["flickr"] is DBNull) ?
            0.0 :
            (double)dr["flickr"],
        PlacesRating =
            (dr["places"] is DBNull) ?
            0.0 :
            (double)dr["places"],
    })

```

Listagens e Tabelas de Apoio

```
    })
    .Except(itineraryPoi)
    .Select(poi => new
    {
        Poi = poi,
        RandomComponent = this.rnd.NextDouble(),
        DistanceComponent =
            1 - travelPath.STDistance(poi.Coords).Value /
            this.Distance,
        PillarComponent = poi
            .Subcategory
            .Pillars
            .Select(scPillar => pillars[scPillar - 1])
            // Lidar com caso de não haver pilar associado.
            .Concat(new int[] { 0 }).Max() / 3.0,
        PredictionComponent = catClass[poi.Category - 1],
        RegionalRelevanceComponent =
            poi.RegionalRelevance / 3.0,
        LocalRelevanceComponent =
            poi.LocalRelevance / 2.0,
        FlickrComponent = poi.FlickrPopularity,
        PlacesComponent =
            poi.PlacesRating / 5.0,
    })
    .OrderByDescending(poi =>
        this.WeightDistance * poi.DistanceComponent +
        this.WeightRandom * poi.RandomComponent +
        this.WeightPillars * poi.PillarComponent +
        this.WeightPrediction * poi.PredictionComponent +
        this.WeightRegionalRelevance *
            poi.RegionalRelevanceComponent +
        this.WeightLocalRelevance *
            poi.LocalRelevanceComponent +
        this.WeightFlickrPopularity * poi.FlickrComponent +
        this.WeightPlacesRating * poi.PlacesComponent)
    .Select(poi => poi.Poi.Id)
    .Take(this.Count)
    .ToArray();
}

private SqlGeography GetCoords(int poiId);
private SqlGeography BuildPath(POI[] itinerary);
private Subcategory[] FetchSubcategories();
private double[] GetCatRecommendations(POI[] itinerary);
private JObject InvokePrediction(JObject predictInput);
private int GetCategory(int poiId);
}
}
```

A.15 Tabela *Cache* das Distâncias em Estradas

A definição da tabela destinada a acolher a *cache* em estrada.

```
CREATE TABLE DOUROREGION.DrivingDists (
    id int IDENTITY NOT NULL PRIMARY KEY,
    id_poi1 int FOREIGN KEY REFERENCES DOUROREGION.Poi(id),
    id_poi2 int FOREIGN KEY REFERENCES DOUROREGION.Poi(id),
    dist float
);
```

A.16 Versão 7: Consideração das Distâncias em Estrada

Esta versão conjuga a consideração das distâncias em estrada com as abordagens anteriores. As modificações encontram-se assinaladas a **negrito>**; como se trata da versão final, o código é apresentado na íntegra, sem omissões de métodos como nas listagens das versões intermédias.

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Diagnostics;
using System.Globalization;
using System.Linq;
using System.Net;
using System.Text;
using System.Web;
using Microsoft.SqlServer.Types;
using Newtonsoft.Json.Linq;

namespace DouroRecommenderEngine
{
    /// <summary>
    /// Classe que mapeia algumas colunas da tabela
    /// DOUROREGION.Subcategoria.
    /// </summary>
    internal class Subcategory
    {
        public int Id { get; set; }
        public int[] Pillars { get; set; }
    }

    /// <summary>
    /// Classe que mapeia algumas colunas da vista DOUROREGION.PoiTur.
    /// </summary>
    internal class POI
    {
        public int Id { get; set; }
        public SqlGeography Coords { get; set; }
        public int Category { get; set; }
        public Subcategory Subcategory { get; set; }
        public int RegionalRelevance { get; set; }
    }
}
```

Listagens e Tabelas de Apoio

```
public int LocalRelevance { get; set; }
public double FlickrPopularity { get; set; }
public double PlacesRating { get; set; }

// Métodos para permitir o uso do método Except().
override public bool Equals(object other)
{
    POI otherPoi = other as POI;
    return (this.Id == otherPoi.Id);
}
override public int GetHashCode()
{
    return this.Id.GetHashCode();
}
}

/// <summary>
/// Implementa o motor de sugestões, exibindo o método Suggest,
/// principal interface de comunicação.
/// </summary>
public class Recommender
{
    private SqlConnection conn;
    private Random rnd;

    // Endereço de acesso à Google Prediction API
    private const string PREDICT_URL =
"https://www.googleapis.com/prediction/v1.2/training/{0}/predict?key={1}";

    /// <summary>
    /// Peso atribuído à componente aleatória.
    /// </summary>
    public double WeightRandom { get; set; }

    /// <summary>
    /// Peso atribuído à componente de distância.
    /// </summary>
    public double WeightDistance { get; set; }

    /// <summary>
    /// Peso atribuído à componente de pilares.
    /// </summary>
    public double WeightPillars { get; set; }

    /// <summary>
    /// Peso atribuído à componente de predição.
    /// </summary>
    public double WeightPrediction { get; set; }

    /// <summary>
    /// Peso atribuído à componente de distância de condução.
    /// </summary>
    public double WeightDrivingDistance { get; set; }

    /// <summary>
```

Listagens e Tabelas de Apoio

```
/// Peso atribuído à componente relevância regional.
/// </summary>
public double WeightRegionalRelevance { get; set; }

/// <summary>
/// Peso atribuído à componente relevância local.
/// </summary>
public double WeightLocalRelevance { get; set; }

/// <summary>
/// Peso atribuído à componente classificação popularidade Flickr.
/// </summary>
public double WeightFlickrPopularity { get; set; }

/// <summary>
/// Peso atribuído à componente classificação Google Places.
/// </summary>
public double WeightPlacesRating { get; set; }

/// <summary>
/// Define qual o número de sugestões a ser devolvidas por cada
/// chamada ao recomendador.
/// </summary>
public int Count { get; set; }

/// <summary>
/// Define a distância máxima a que um POI pode estar do itinerário
/// do utilizador para poder ser recomendado.
/// </summary>
public double Distance { get; set; }

/// <summary>
/// Directório onde se encontra o ficheiro oacurl-1.2.0.jar
/// </summary>
public string OacurlPath { get; set; }

/// <summary>
/// Modelo treinado com as categorias dos itinerários disponíveis.
/// </summary>
public string Model { get; set; }

/// <summary>
/// Chave para acesso à Google Prediction API.
/// </summary>
public string Key { get; set; }

/// <summary>
/// Constrói um novo objecto, parametrizado com uma connection
/// string.
/// </summary>
/// <param name="connStr">Connection string para ligação à base de
/// dados.</param>
public Recommender(string connStr)
{
    this.conn = new SqlConnection(connStr);
}
```

Listagens e Tabelas de Apoio

```
this.conn.Open();
this.rnd = new Random();
this.Count = 5;
this.Distance = 7000.0;
this.WeightRandom = 1.0;
this.WeightDistance = 2.0;
this.WeightPillars = 5.0;
this.WeightPrediction = 5.0;
this.WeightDrivingDistance = 3.0;
this.WeightRegionalRelevance = 8.0;
this.WeightLocalRelevance = 3.0;
this.WeightFlickrPopularity = 3.0;
this.WeightPlacesRating = 2.0;
}

/// <summary>
/// Executa a lógica de recomendação.
/// </summary>
/// <param name="itinerary">Array com a sequência dos IDs dos POI
/// que o utilizador pretende visitar.</param>
/// <param name="pillars">Perfil do utilizador, definido como o
/// nível de interesse (0 a 3) para cada pilar.</param>
/// <returns>Array com os IDs dos pontos de interesse
/// sugeridos.</returns>
public int[] Suggest(int[] itinerary, int[] pillars)
{
    if (itinerary.Length == 0 || pillars.Length == 0)
        return null;
    POI[] itineraryPoi = itinerary
        .Select(poiId => new POI
        {
            Id = poiId,
            Coords = this.GetCoords(poiId),
            Category = this.GetCategory(poiId),
        })
        .ToArray();
    double[] catClass = this.GetCatRecommendations(itineraryPoi);
    Subcategory[] subcategories = this.FetchSubcategories();
    SqlGeography travelPath = this.BuildPath(itineraryPoi);
    // Embora a chamada STIntersects também possa ser feita na
    // camada lógica (neste componente de recomendação), realizá-la
    // antes na camada de persistência permite tirar partido de
    // eventuais índices espaciais.
    SqlCommand command = new SqlCommand(@"
SELECT *
FROM DOUROREGION.PoiTur p
WHERE @buffer.STIntersects(p.coords) = 1;",
        this.conn);
    command.Parameters.Add(new SqlParameter
    {
        ParameterName = "@buffer",
        UdtTypeName = "geography",
        Value = travelPath.STBuffer(this.Distance),
    });
    SqlDataAdapter da = new SqlDataAdapter(command);
```

Listagens e Tabelas de Apoio

```
DataSet ds = new DataSet();
da.Fill(ds, "DOUROREGION.PoiTur");
return ds
    .Tables["DOUROREGION.PoiTur"]
    .AsEnumerable()
    .Select(dr => new POI
    {
        Id = (int)dr["id"],
        Coords = dr["coords"] as SqlGeography,
        Category = (int)dr["id_Categoria"],
        Subcategory = subcategories
            .Single(sc =>
                (sc.Id == (int)dr["id_Subcategoria"])),
        RegionalRelevance =
            (dr["relevancia_regional"] is DBNull) ?
            0 :
            (int)dr["relevancia_regional"] % 10,
        LocalRelevance =
            (dr["relevancia_local"] is DBNull) ?
            0 :
            (int)dr["relevancia_local"],
        FlickrPopularity =
            (dr["flickr"] is DBNull) ?
            0.0 :
            (double)dr["flickr"],
        PlacesRating =
            (dr["places"] is DBNull) ?
            0.0 :
            (double)dr["places"],
    })
    .Except(itineraryPoi)
    .Select(poi => new
    {
        Poi = poi,
        RandomComponent = this.rnd.NextDouble(),
        DistanceComponent =
            1 - travelPath.STDistance(poi.Coords).Value /
            this.Distance,
        PillarComponent = poi
            .Subcategory
            .Pillars
            .Select(scPillar => pillars[scPillar - 1])
            // Lidar com caso de não haver pilar associado.
            .Concat(new int[] { 0 }).Max() / 3.0,
        PredictionComponent = catClass[poi.Category - 1],
        RegionalRelevanceComponent =
            poi.RegionalRelevance / 3.0,
        LocalRelevanceComponent =
            poi.LocalRelevance / 2.0,
        FlickrComponent = poi.FlickrPopularity,
        PlacesComponent =
            poi.PlacesRating / 5.0,
    })
    .OrderByDescending(poi =>
        this.WeightDistance * poi.DistanceComponent +
```

Listagens e Tabelas de Apoio

```
this.WeightRandom * poi.RandomComponent +
this.WeightPillars * poi.PillarComponent +
this.WeightPrediction * poi.PredictionComponent +
this.WeightRegionalRelevance *
    poi.RegionalRelevanceComponent +
this.WeightLocalRelevance *
    poi.LocalRelevanceComponent +
this.WeightFlickrPopularity * poi.FlickrComponent +
this.WeightPlacesRating * poi.PlacesComponent)
.Take(30)
// Agora há um melhor valor que distância em linha recta,
// pelo que este valor é descartado.
.OrderByDescending(poi =>
    this.WeightRandom * poi.RandomComponent +
    this.WeightPillars * poi.PillarComponent +
    this.WeightPrediction * poi.PredictionComponent +
    this.WeightRegionalRelevance *
        poi.RegionalRelevanceComponent +
    this.WeightLocalRelevance *
        poi.LocalRelevanceComponent +
    this.WeightFlickrPopularity * poi.FlickrComponent +
    this.WeightPlacesRating * poi.PlacesComponent +
    this.WeightDrivingDistance * (1 -
        Math.Min(this.DrivingDist(itineraryPoi, poi.Poi),
            this.Distance) / this.Distance))
.Select(poi => poi.Poi.Id)
.Take(this.Count)
.ToArray();
}

/// <summary>
/// Obtém as coordenadas do POI da base de dados.
/// </summary>
/// <param name="poiId">Id do POI.</param>
/// <returns>Coordenadas do POI.</returns>
private SqlGeography GetCoords(int poiId)
{
    SqlCommand command = new SqlCommand(@"
SELECT coords
FROM DOUROREGION.Poi
WHERE id = @poiId;",
    this.conn);
    command.Parameters.AddWithValue("@poiId", poiId);
    return command.ExecuteScalar() as SqlGeography;
}

/// <summary>
/// Constrói um LineString que une os pontos que o utilizador
/// pretende visitar.
/// </summary>
/// <param name="itinerary">Array com a sequência dos POI que o
/// utilizador pretende visitar.</param>
/// <returns>O LineString construído (ou um Point, caso o array
/// contenha apenas um ponto.)</returns>
private SqlGeography BuildPath(POI[] itinerary)
```

Listagens e Tabelas de Apoio

```
{
    SqlGeography first = itinerary.First().Coords;
    if (itinerary.Length == 1) return first;
    SqlGeographyBuilder builder = new SqlGeographyBuilder();
    builder.SetSrid(4326);
    builder.BeginGeography(OpenGisGeographyType.LineString);
    builder.BeginFigure(first.Lat.Value, first.Long.Value);
    foreach (POI p in itinerary.Skip(1))
    {
        builder.AddLine(p.Coords.Lat.Value, p.Coords.Long.Value);
    }
    builder.EndFigure();
    builder.EndGeography();
    return builder.ConstructedGeography;
}

/// <summary>
/// Obtém da base de dados as subcategorias e respectivos pilares.
/// </summary>
/// <returns>Um array de objectos Subcategory.</returns>
private Subcategory[] FetchSubcategories()
{
    SqlCommand command = new SqlCommand(@"
SELECT sc.id, pasc.id_PilarActividade
FROM DOUROREGION.Subcategoria sc
    JOIN DOUROREGION.PilarActividade_Subcategoria pasc
    ON sc.id = pasc.id_Subcategoria
WHERE pasc.id_PilarActividade <= 4;",
        this.conn);
    SqlDataAdapter da = new SqlDataAdapter(command);
    DataSet ds = new DataSet();
    da.Fill(ds);
    DataTable dt = ds.Tables[0];
    return dt
        .AsEnumerable()
        .Select(dr => new
        {
            Id = (int)dr["id"],
            Pillar = (int)dr["id_PilarActividade"],
        })
        .GroupBy(scl => scl.Id)
        .Select(scg => new Subcategory
        {
            Id = scg.Key,
            Pillars = scg.Select(scl => scl.Pillar).ToArray(),
        })
        .ToArray();
}

/// <summary>
/// Obtém recomendações relativas às categorias em que o utilizador
/// possa estar interessado, com base no percurso definido.
/// </summary>
/// <param name="itinerary">Array com a sequência dos POI que o
/// utilizador pretende visitar.</param>
```

Listagens e Tabelas de Apoio

```
/// <returns>Array com os níveis de interesse (de 0,0 a 1,1) que o
/// utilizador pode ter relativamente a cada categoria.</returns>
private double[] GetCatRecommendations(POI[] itinerary)
{
    if (String.IsNullOrEmpty(this.OacurlPath) ||
        String.IsNullOrEmpty(this.Model) ||
        String.IsNullOrEmpty(this.Key))
    {
        return new double[10];
    }

    int[] catCounter = new int[10];
    foreach (POI poi in itinerary)
    {
        catCounter[poi.Category - 1]++;
    }
    double catTotal = catCounter.Sum();
    var catProportions =
        catCounter.Select(count => count / catTotal);
    JObject predictIn = JObject.FromObject(new
    {
        input = new
        {
            csvInstance = catProportions,
        }
    });

    JObject predictOut = this.InvokePrediction(predictIn);

    double[] scores = new double[10];
    foreach (JObject labelScorePair in predictOut["outputMulti"])
    {
        string label = labelScorePair["label"].ToString();
        // Remover "Cat" do início.
        int category = Int32.Parse(label.Substring(3));
        scores[category - 1] = (double)labelScorePair["score"];
    }
    // Passar classificação da categoria escolhida para 1,0.
    double max = scores.Max();
    return scores.Select(score => score / max).ToArray();
}

/// <summary>
/// Faz um pedido de predição à Google Prediction API.
/// </summary>
/// <param name="predictInput">JSON com as proporções das
/// categorias.</param>
/// <returns></returns>
private JObject InvokePrediction(JObject predictInput)
{
    Process p = new Process();
    p.StartInfo.UseShellExecute = false;
    p.StartInfo.RedirectStandardOutput = true;
    p.StartInfo.RedirectStandardInput = true;
    p.StartInfo.CreateNoWindow = true;
```

Listagens e Tabelas de Apoio

```
p.StartInfo.WorkingDirectory = this.OacurlPath;
p.StartInfo.FileName = "java.exe";
StringBuilder sb = new StringBuilder();
sb.Append("-cp ");
sb.Append("./oacurl-1.2.0.jar ");
sb.Append("com.google.oacurl.Fetch ");
sb.Append("-X POST ");
sb.Append("-t JSON \"");
sb.Append(String.Format(
    PREDICT_URL,
    HttpUtility.UrlEncode(this.Model),
    this.Key));
sb.Append('\"');
p.StartInfo.Arguments = sb.ToString();
p.Start();
p.StandardInput.WriteLine(predictInput.ToString());
p.StandardInput.Close();
string output = p.StandardOutput.ReadToEnd();
p.WaitForExit();
return JObject.Parse(output);
}

/// <summary>
/// Obtém a categoria do POI da base de dados.
/// </summary>
/// <param name="poiId">Id do POI.</param>
/// <returns>Categoria do POI.</returns>
private int GetCategory(int poiId)
{
    SqlCommand command = new SqlCommand(@"
SELECT id_Categoria
FROM DOUROREGION.Poi
WHERE id = @poiId;",
    this.conn);
    command.Parameters.AddWithValue("@poiId", poiId);
    return (int)command.ExecuteScalar();
}

/// <summary>
/// Determina o menor desvio a um percurso que é necessário fazer
/// para visitar uma sugestão.
/// </summary>
/// <param name="itinerary">Array com a sequência dos POI que o
/// utilizador pretende visitar.</param>
/// <param name="suggestion">POI considerado.</param>
/// <returns>A menor distância em estrada (metros).</returns>
private double DrivingDist(POI[] itinerary, POI suggestion)
{
    if (itinerary.Length == 0) return -1.0;
    double[] drivingDists = new double[itinerary.Length + 1];
    drivingDists[0] = this.GetDrivingDist(
        suggestion, itinerary[0]);
    for (int i = 0; i < itinerary.Length - 1; i++)
    {
        drivingDists[i + 1] =
```

Listagens e Tabelas de Apoio

```
        this.GetDrivingDist(itinerary[i], suggestion)
        + this.GetDrivingDist(itinerary[i + 1], suggestion)
        - this.GetDrivingDist(itinerary[i], itinerary[i + 1]));
    }
    drivingDists[itinerary.Length] = this.GetDrivingDist(
        itinerary[itinerary.Length - 1], suggestion);
    return drivingDists.Min();
}

/// <summary>
/// Calcula a distância em estrada entre dois pontos.
/// </summary>
/// <param name="poi1">POI de origem.</param>
/// <param name="poi2">POI de destino.</param>
/// <returns>A distância em metros.</returns>
private double GetDrivingDist(POI poi1, POI poi2)
{
    double dist;

    dist = this.GetCachedDist(poi1, poi2);
    if (dist != -1.0) return dist;

    // Tentar ignorar estradas de sentido único.
    dist = this.GetCachedDist(poi2, poi1);
    if (dist != -1.0) return dist;

    // Pedido ao servidor remoto.
    dist = this.QueryDirectionsServer(poi1, poi2);
    if (dist != -1.0)
    {
        this.SetCachedDist(poi1, poi2, dist);
        return dist;
    }

    // Limite de pedidos atingido.
    dist = this.EstimateDrivingDistance(poi1, poi2);
    return dist;
}

/// <summary>
/// Obtém a distância entre dois pontos armazenada na base de
/// dados.
/// </summary>
/// <param name="poi1">POI de origem.</param>
/// <param name="poi2">POI de destino.</param>
/// <returns>A distância em metros ou -1,0 caso não seja
/// encontrada.</returns>
private double GetCachedDist(POI poi1, POI poi2)
{
    SqlCommand command = new SqlCommand(@"
SELECT dist
FROM DOUROREGION.DrivingDists
WHERE id_poi1 = @poiId1 AND id_poi2 = @poiId2;",
        this.conn);
    command.Parameters.AddWithValue("@poiId1", poi1.Id);
```

```

        command.Parameters.AddWithValue("@poiId2", poi2.Id);
        object result = command.ExecuteScalar();
        return (result != null) ? (double)result : -1.0;
    }

    /// <summary>
    /// Guarda a distância entre dois pontos na base de dados.
    /// </summary>
    /// <param name="poi1">POI de origem.</param>
    /// <param name="poi2">POI de destino.</param>
    /// <param name="dist">Distância em metros.</param>
    private void SetCachedDist(POI poi1, POI poi2, double dist)
    {
        SqlCommand command = new SqlCommand(@"
INSERT INTO DOUROREGION.DrivingDists (id_poi1, id_poi2, dist)
VALUES (@poiId1, @poiId2, @dist);",
        this.conn);
        command.Parameters.AddWithValue("@poiId1", poi1.Id);
        command.Parameters.AddWithValue("@poiId2", poi2.Id);
        command.Parameters.AddWithValue("@dist", dist);
        command.ExecuteNonQuery();
    }

    /// <summary>
    /// Consulta a Google Distance Matrix API.
    /// </summary>
    /// <param name="poi1">POI de origem.</param>
    /// <param name="poi2">POI de destino.</param>
    /// <param name="dist">Distância em metros.</param>
    private double QueryDirectionsServer(POI poi1, POI poi2)
    {
        WebClient wb = new WebClient();
        CultureInfo invariant = CultureInfo.InvariantCulture;
        StringBuilder query = new StringBuilder();
        query.Append("origins=");
        query.Append(poi1.Coords.Lat.Value.ToString(invariant));
        query.Append(',');
        query.Append(poi1.Coords.Long.Value.ToString(invariant));
        query.Append("&destinations=");
        query.Append(poi2.Coords.Lat.Value.ToString(invariant));
        query.Append(',');
        query.Append(poi2.Coords.Long.Value.ToString(invariant));
        query.Append("&sensor=false");
        UriBuilder ubuilder = new UriBuilder
        {
            Scheme = "http",
            Host = "maps.googleapis.com",
            Path = "maps/api/distancematrix/json",
            Query = query.ToString(),
        };
        string distDetails = wb.DownloadString(ubuilder.Uri);
        JObject d = JObject.Parse(distDetails);
        if (d["status"].ToString() != "OK") return -1.0;
        if (d["rows"][0]["elements"][0]["status"].ToString()
            == "ZERO_RESULTS") return -1.0;
    }

```

```

        // Não é possível passar directamente para double.
        return (int)d["rows"][0]["elements"][0]["distance"]["value"];
    }

    /// <summary>
    /// Estima a distância em estrada entre dois pontos, com base na
    /// distância em linha recta entre os dois.
    /// </summary>
    /// <param name="poi1">POI de origem.</param>
    /// <param name="poi2">POI de destino.</param>
    /// <returns>Estimativa, com base na razão média entre as distância
    /// em estrada e em linha recta armazenadas na base de
    /// dados.</returns>
    private double EstimateDrivingDistance(POI poi1, POI poi2)
    {
        SqlCommand command = new SqlCommand(@"
SELECT SUM(dd.dist) / SUM(p1.coords.STDistance(p2.coords))
FROM DOUROREGION.DrivingDists dd
JOIN DOUROREGION.Poi p1 ON dd.id_poi1 = p1.id
JOIN DOUROREGION.Poi p2 ON dd.id_poi2 = p2.id;",
        this.conn);
        double ratio = (double)command.ExecuteScalar();
        return poi1.Coords.STDistance(poi2.Coords).Value * ratio;
    }
}

```
