Universidade da Beira Interior
Departamento de Informática

Nº 30949 - 2020: *Uso de Técnicas baseadas no Algoritmo Lime Para Reconhecimento Biométrico Interpretável*

Elaborado por:

Artur Miguel da Silva Grilo

Orientador:

Professor Doutor Hugo Pedro Martins Carriço Proença

7 de Setembro de 2020

Agradecimentos

Durante a minha vida sempre pensei que po A conclusão deste trabalho, bem como da grande maior parte da minha vida académica não seria possível sem a ajuda de ínumeras pessoas fantásticas, para aqueles que me ajudaram

1. À minha querida mãe, que apesar de todo o sacrifício, nunca desistiu de mim, apoiando-me em todos os momentos da minha vida. Um obrigado infinito a esta guerreira sem armas que eu tanto amo.

2. À minha namorada, sempre pronta para ajudar, coração mais doce que conheço e que sempre lá esteve nos bons e maus momentos. Ela é sem dúvida o meu braço direito e a luz dos meus olhos.

3. Aos meus irmãos, pelas lições de moral ao irmão mais novo, mas sempre com grande dignidade e valor. Que bom é ter irmãos.

4. Aos meus amigos, Diogo e David, que nunca me falham e que cumprem o ditado, poucos mas bons!

5. À minha avó, que sempre esperou ver o neto licenciado mais que tudo e que apesar de rabugenta e feitio difícil, deixa saudade, espero que fiques feliz onde quer que estejas.

6. À minha família mais próxima por demonstrarem serem um verdadeiro exemplo de pessoas boas e de bons valores. Obrigado pela transmissão de sabedoria, é um prazer pertencer a tal união de pessoas do bem. Amén.

7. Ao meu orientador Professor Doutor Hugo Pedro Martins Carriço Proença pela segunda oportunidade que me deu em desenvolver um projeto com a sua orientação.

8. E por fim, aos meus sobrinhos, por me mostrarem continuamente a beleza que é a ingenuidade de uma criança e quão bonito é o crescimento no nosso seio familiar.

Conteúdo

Lista de Figuras

Acrónimos

AL Algoritmo Lime

UBI Universidade da Beira Interior

UC Unidade Curricular

ML *Machine Learning*

IA Inteligência Artificial

SP Super-Pixel

IDE Ambiente de Desenvolvimento Integrado

DS *Data Science*

SLIC *Simple Linear Iterative Clustering*

CIE Comissão Internacional de Iluminação

MAM Métodos Agnósticos ao Modelo

ONEIROS *Open-ended Neuro-Electronic Intelligent Robot Operating System*

Capítulo

1

Introdução

1.1 Enquadramento

Este projeto, será realizado no âmbito da Unidade Curricular (UC) Projeto (11572) do curso de Engenharia Informática, da Universidade da Beira Interior (UBI).

1.2 Motivação

Este projeto vai de encontro a um conceito muito atual e que tem ganho cada vez mais destaque em na informática moderna, *Machine Learning* (ML), um ramo de Inteligência Artificial (IA).

O interesse por esta área, IA , não só pela sua crescente importância no mundo da informática mas também por tudo o que advém desta para benefício humano, evolução, maior facilidade e acessibilidade na resolução de problemas, fizeram com que decidisse realizar este projeto.

O outro motivo é, visto que a utilização de modelos de ML é cada vez mais abrangente, estes são utilizados em várias áreas críticas, como na área da saúde em diagnósticos e faz todo o sentido tornar este mecanismo interpretável, principalmente em aplicações tão sensíveis como esta.

1.3 Objetivos

O principal objetivo deste projeto, tal como o seu título indica, passa por perceber e implementar o Algoritmo Lime (AL) sob um modelo de reconhecimento biométrico de pares de imagens. Para ser mais preciso, a partir da uti-

lização do algoritmo em estudo, o objetivo passa por tornar um modelo não-interpretável num modelo interpretável. Através desta interpretabilidade, conceito chave deste relatório, o utilizador consegue perceber as previsões que o modelo oferece. Como utilizador de um modelo de ML, torna-se fulcral perceber o porquê das previsões dos modelos, visto a crescente complexidade que estes apresentam bem como a sua falta de interpretabilidade.

Para conseguir este objetivo, terei que estudar e entender melhor os princípios básicos de ML bem como, a manipulação do modelo em estudo que me foi fornecido e pré-treinado, a fim de alcançar destreza para a implementação do AL no mesmo e na manipulacão dos dados de entrada para teste.

1.4 Organização do Documento

De modo a refletir o trabalho que foi feito, este documento encontra-se estruturado da seguinte forma:

1. O primeiro capítulo – **Introdução** – apresenta o projeto, a motivação para a sua escolha, o enquadramento para o mesmo, os seus objetivos e a respetiva organização do documento.

2. O segundo capítulo – **Introdução a Machine Learning Interpretável** – descreve de forma teórica o conceito de ML, passando pela história da informática, a sua implementação clássica e definição, bem como, o algoritmo em estudo neste projeto, **LIME!** (**LIME!**). A sua definição, em que consiste e a sua importância no contexto de ML, o processo que envolve a sua implementação e a sua utilização em imagens.

3. O terceiro capítulo – **Tecnologias e Ferramentas Utilizadas** – descreve como o nome indica, tecnologias e ferramentas utilizadas durante o desenvolvimento deste projeto.

4. O quarto capítulo – **Implementação e Testes** – começa por descrever o modelo utilizado para melhor compreensão dos pontos seguintes. De seguida dá uma visão da parte prática do projeto, começando por apresentar a demonstração da implementação do Algoritmo SLIC para extração de Super-Píxeis, bem como, a implementação de ruído para criação de perturbações para originar o conjunto de teste e o seu resultado na rede. Para finalizar, continuando na vertente prática, é demonstrado como foi criado o modelo linear que possibilitou a extração dos Super-Píxeis com mais peso.

5. O quinto capítulo – **Conclusões e Trabalho Futuro** – enumera as principais conclusões obtidas ao longo da realização do projeto, bem como, dificuldades apresentadas e falhas.

Capítulo

2

Introdução a Machine Learning Interpretável

2.1 Introdução

Cada vez mais, nos dias que correm, o ser humano é dependente de todo o tipo de tecnologia. Essa tecnologia permite-nos ser mais eficazes, eficientes e de certo modo atingir objetivos que sem esta seria impossível.

O ser humano por mais incrível que seja, toda a sua complexidade física e mental, apresenta limitações. Focando-nos nas mentais, o nosso cérebro é a base de tudo mas é limitado fisicamente pelo crânio. Sempre foi inato no ser humano procurar auxílio para a resolução de problemas, desde os registos em paredes de cavernas, passando pelas calculadoras até chegarmos à criação de sistemas computacionais. Grande propósito de todos estes mecanismos, processamento de dados. A única diferença dos primeiros exemplos para o computador é que este consegue executar estes processamentos de forma automática. Isto está presente no quotidiano humano mesmo que este não o note.

Com este avanço exponencial ao nível da tecnologia é possível um desenvolvimento muito mais rápido em relação à resolução de qualquer tipo de problemas através destes sistemas computacionais.

É aqui que entra o conceito de ML, um dos mais incríveis avanços da tecnologia na história da informática contemporânea e que através de análise de dados para elaboração de modelos analíticos complexos, consegue auxiliar na toma de decisões por parte do utilizador em diversas aplicações.

Finalizando, é também importante referir que, se a partir da utilização do mecanismo de ML os utilizadores são solicitados a confiar em modelos que os

ajudam na toma de decisões a problemas, torna-se essencial, tornar o mesmo interpretável. Este é o conceito chave deste projeto, perceber o porquê destes modelos nos providenciarem tais decisões, pois além de estas serem susceptíveis a erros, torna-se importante para o utilizador compreender todo o mecanismo que gera estes resultados, para que este se torne um modelo confiável.

2.2 Machine Learning

2.2.1 Introdução

A origem do conceito de ML provém da década de 1950 onde Alan Turing, considerado o pai da informática moderna, criou o teste de Turing. Este teste consistia na medição do quanto as máquinas conseguiriam representar raciocínio tal como o ser humano.

Esta ideia inicial de igualar máquinas e pessoas tornou-se, com o passar do tempo e na superação de vários obstáculos, na superação da limitação do raciocínio humano. Com este maior objetivo, este mecanismo apresenta uma importância vital na necessidade de inovar, condição indispensável para o humano, tanto a nível pessoal como de negócios.

2.2.2 Breve Definição do Conceito

ML é um ramo de IA que fornece aos sistemas a capacidade de fazer e melhorar previsões ou comportamentos através de dados fornecidos a estes.

O processo de aprendizagem consiste na utilização do conjunto de entrada como exemplos de treino, com a finalidade de detetar padrões e tomar melhores decisões no futuro, de forma iterativa através da execução de algoritmos. Com isto a aprendizagem dos computadores torna-se automática e sem intervenção ou assistência humana.

Na figura 2.1 está representado de uma forma ilustrativa o mecanismo clássico de ML.

Figura 2.1: Mecanismo clássico de Machine Learning.

2.2.3 Machine Learning Supervisionado

Este projeto segue a ideologia de ML supervisionado, que cobre todos os problemas que retornam previsões. Este possui um conjunto de dados de treino para o qual já sabe o resultado de interesse e que, a partir deste e da experiência adquirida, aprende a prever novos resultados. O objetivo desta aprendizagem consiste em treinar um modelo preditivo, onde o conjunto de dados de entrada é mapeado a nível de características. No caso em estudo neste projeto, verificação biométrica, o mapeamento dos dados de entrada, é logicamente, as diferentes partes, oculares e perioculares, presentes nas imagens cedidas ao modelo para treino.

Para uma bem sucessida criação de um modelo de ML supervisionado, tem de se seguir algumas etapas:

1. Máxima recolha de dados para treinar o modelo com uma boa base quantitativa mas também qualitativa. Estes dados são mapeados por características e já contêm o resultado de interesse.

2. Utilização do conjunto de dados, enunciado no ponto anterior, na implementação de um algoritmo que gera um modelo de ML. Este algoritmo de aprendizagem irá utilizar os dados para treinar o modelo.

3. Utilizar o modelo originado com novos dados de teste para obtenção de novas previsões ponderadas originadas automaticamente pelo modelo.

2.2.4 Modelo de Caixa Negra

Um modelo de caixa negra é aquele em que não são revelados os seus mecanismos internos e que não podem ser entendidos pelos seus parâmetros.

Como referido na implementação clássica de ML sobre um problema, esta funcionava como um espécie de caixa negra, em que, são conferidos ao modelo, uma estrutura de dados de entrada a partir dos quais são geradas previsões, sem qualquer tipo de explicação.

No que concerne a este projeto, torna-se importante a descrição deste tipo de modelos de ML (apesar que até certo ponto todos eles são), visto que o modelo utlizado na execução deste relatório se trata de uma rede neuronal.

Ao contrário dos modelos interpretáveis, como por exemplo, a regressão linear ou árvores de decisão, os modelos são bastante interpretáveis mas em relação a redes neuronais não se passa o mesmo. As previsões destas são bastante difíceis de interpretar devido à caracterização do seu conjunto de entrada. No entanto estas oferecem um desempenho muito superior aos modelos interpretáveis, na medida em que, pode ser treinado um número muito

avultado de dados ao mesmo tempo e as previsões são mais acertadas. Representações mais complexas são modeladas a partir de um número maior de parâmetros.

Na figura está representado o que foi referido anteriormente.

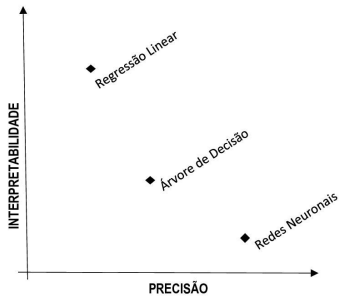

Figura 2.2: Interpretabilidade vs Precisão de dois modelos interpretáveis e da rede neuronal.

2.2.5 Importância da Interpretabilidade

Sendo o conceito de ML tão recorrente na informática moderna, bem como, o uso crescente em aplicações críticas, como por exemplo veículos auto-dirigíveis e diagnósticos médicos, torna-se fulcral treinarmos e criarmos um bom modelo.

Estes modelos não revelam os seus mecanismos internos, portanto, não podem ser entendidos pelos seus parâmetros.

É a partir de Métodos Agnósticos ao Modelo (MAM) que se torna possível a interpretabilidade dos modelos de ML de caixa negra. Estes modelos sendo de uso universal para todos os modelos de aprendizagem faz com que também possam ser usados para modelos que não são de caixa negra, apesar que os tratam como o fossem.

Assim, o conceito de interpretabilidade, apresenta extrema importância para atingir o objetivo de treinar e criar um modelo bom. A partir desta consegue-se a compreensão das previsões geradas e respetivamente, confiança no modelo em uso, passando o utilizador a ter novamente um papel ativo na toma de decisão. Passa a não haver um subjugamento no julgamento humano, como acontecia na implementação clássica. Sem falar que os modelos podem fazer previsões erradas e assim tornam-se mais fáceis de detetar pelo utilizador, ou ainda, em casos onde se torna importante saber o porquê das predições com o objetivo de aprender mais sobre o problema.

Nem sempre fará sentido a interpretabilidade de um modelo, com isto quer dizer que, a sua implementação em ambientes de baixo risco será um

pouco perca de tempo e esforço, na medida em que erros nas previsões não originarão consequências graves. Esta, só fará sentido na medida em que erros nas previsões possam originar consequências graves ou quando se torna importante a percepção de tais resultados por parte do utilzador.

Na figura 2.3 está presente esquematicamente a importância da interpretabilidade em modelos de caixa negra.

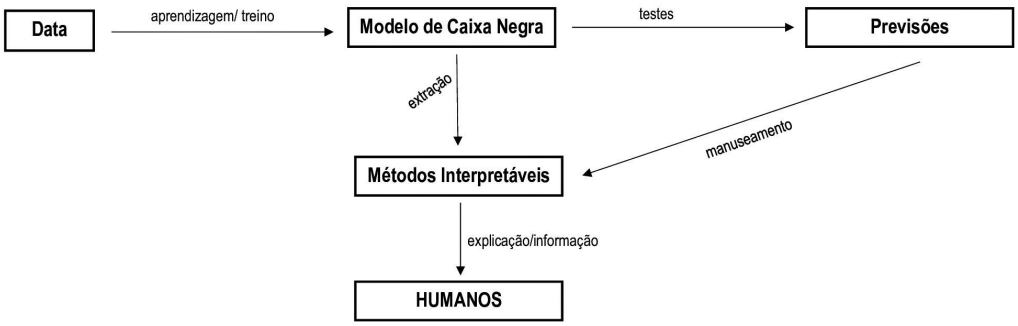

Figura 2.3: Importância de métodos interpretáveis.

2.2.6 Conclusão

O conceito de ML traz com ele várias vantagens ao desenvolvimento humano, tanto a nível de reprodutibilidade, escala e velocidade. A nível de processamento de dados para obter previsões, a tarefa, é muito mais rapidamente executada por estes modelos, quando não é mesmo impossível tão grande processamento de dados por parte humana. Logo, podemos concluir, que os humanos necessitam deste tipo de modelos para a sua própria evolução.

Estes modelos são aplicados em grande abrangência a nível de todo o tipo de aplicações, desde as de ambiente de baixo risco até às de risco elevado. O mecanismo de aprendizagem, levado a cabo pelos modelos, funciona como uma caixa negra. Então, para evitar males maiores, a níveis de consequências, será crucial tornar alguns destes modelos interpretáveis. É quase como deixar a decisão ou julgamento final para o utilizador, permitindo a este uma melhor aprendizagem sobre o problema, bem como, filtro final das novas previsões originadas.

2.3 Métodos Interpretativos Agnósticos ao Modelo

2.3.1 Introdução

A concretização do objetivo principal deste projeto centra-se na utilização de métodos agnósticos ao modelo ao invés dos modelos interpretativos específicos ou métodos interpretativos. Esta escolha centra-se no princípio fundamental que estes métodos apresentam, a flexibilidade. Este princípio, em termos de interpretabilidade, torna bem mais fácil a sua implementação visto que estes métodos podem ser utilizados em qualquer modelo de ML e também, a nível de explicação onde também não há uma limitação a uma forma, pode-se utilizar por exemplo, imagens, gráficos ou uma fórmula linear. Isto deve-se em grande parte à sua flexibilidade, e com isto quer dizer que, estes modelos podem ser utilizados em qualquer modelo de ML.

Nesta secção serão enunciados dois métodos interpretativos agnósticos ao modelo, em que, um consiste numa técnica local, isto é, o método interpretativo explica previsões individualmente, e o último numa técnica global, que consiste, na explicação do comportamento de todo o modelo através do método.

2.3.2 Algoritmo LIME

2.3.2.1 Breve Definição

LIME (Local interpretable model-agnostic explanations) é um algoritmo que foi desenvolvido pelos investigadores Marco Tulio Ribeiro, Sameer Singh e Carlos Guestrin na Universidade de Washington e consiste numa técnica que permite e tem como principal objetivo tornar as previsões de qualquer modelo de ML interpretáveis.

O algoritmo é independente do modelo em uso(model-agnostic), o que significa que pode ser aplicado a qualquer modelo de ML. Isto é conseguido por que este "aprende"o comportamento do modelo através de perturbações efetuadas na estrutura de entrada. Portanto, o algoritmo permite-nos, a partir da instância de entrada específica modificada ou perturbada, ver as diferenças e o respectivo impacto nas previsões. A partir deste novo conjunto de dados perturbados, e visto que o modelo pode ser sondado quantas vezes for pretendido, o objetivo passa por perceber o porquê de determinada previsão.

O conceito de interpretabilidade do modelo torna-se possível, através do algoritmo, através da proximidade da instância perturbada que foi gerada em relação à instância original.

2.3.2.2 Modelos Substitutos Locais

1. Selecionar uma instância de interesse, aquela para a qual vai ser aplicada o modelo da caixa negra de forma a obter uma previsão.

2. Perturbar, ou criar ruído, na instância anteriormente referida, obtendo assim, um novo conjunto de dados de entrada do modelo. É a partir desta estrutura que é treinado um modelo interpretável, que considera a proximidade das instâncias criadas com a instância de interesse.

3. Classificar estas amostras modificadas com a proximidade à instância de interesse.

4. Treinar um modelo bem classificado e ponderado no conjunto de dados com variações e explicar a previsão interpretando o modelo local.

Matematicamente, a explicação deste processo, pode ser obtida a partir da fórmula abaixo apresentada:

$$explicação(x) = \arg\min_{g \in G} L(f, g, \pi_x) + \Omega(g).$$

Sendo G a família de explicações possíveis ou a classe de modelos potencialmente interpretáveis e g o modelo de explicação para a instância x, então $g \in G$.

Portanto o modelo g é aquele que minimiza a perda L. Esta perda mede o quão perto a explicação da instância x está da previsão do modelo original f. Assim sendo $f(x)$ é a probabilidade que x pertence a uma determinada classe.

$\Omega(g)$ é a medida de complexidade da explicação do modelo. Para modelos lineares, como o que se trata neste projeto, pode ser definido como o número de pesos diferentes de zero. Este parâmetro deve ser mantido baixo com a finalidade de poder ser interpretado pelo utilizador.

π_x trata-se de uma medida de proximidade entre uma instância z a x, de modo a definir a localidade em torno de x, ou seja, define quão grande é a vizinhança em torno da instância x que consideramos para a explicação.

Na prática, LIME otimiza apenas a parte de perda. O utilizador deve determinar a complexidade, por exemplo, selecionando o número máximo de recursos que o modelo de regressão linear pode utilizar.

Concluindo, assumindo que $L(f, g, \pi_x)$ seja uma medida de quão infiel é g ao se aproximar de f na localidade definida porπ_x, a interpretabilidade do modelo é garantida através da minimização da perda, $L(f, g, \pi_x)$.

Na figura X podemos perceber de forma gráfica todo o processo envolvido e mencionado em cima.

A função de decisão do modelo original é representada pelo fundo azul / rosa claramente não é linear. A cruz vermelha brilhante é a instância que está sendo explicada (vamos chamá-la de X). Mostramos instâncias perturbadas em torno de X e ponderamos de acordo com sua proximidade de X (o peso aqui é representado pelo tamanho). Obtemos a previsão do modelo original nessas instâncias perturbadas e, em seguida, aprendemos um modelo linear (linha tracejada) que se aproxima bem do modelo na vizinhança de X. A partir desta imagem podemos concluir que a explicação neste caso não é fiel globalmente, mas é fiel localmente em torno de X.

2.3.2.3 Utilização em Redes Neuronais - Explicação para imagens

A implementação do algoritmo em imagens, caso em estudo neste projeto, o processo de criação dos modelos substitutos locais, enunciado no ponto anterior, e a criação de um modelo linear para explicação das previsões, é possível através dos dois seguintes passos:

1. Segmentação da instância de interesse em Super-Píxeis.

   Visto que a instância em estudo se trata de uma imagem, acabaria por não ser uma boa prática perturbar píxes individualmente, visto que a partir de um, não se conseguiria determinar a que classe pertenceria a instância. Isto acontece porque a perturbação individual aleatória de píxeis não iria ter revelância suficiente para alterar muito as previsões do modelo. Essas perturbações apenas são conseguidas através da segmentação da instância específica em super-píxeis.

   Um Super-Pixel (SP) consiste num conjunto de píxeis vizinhos que partilham entre si um gradiente/intensidade de cor e brilho semelhante o bastante para se diferenciarem dos píxeis pertencentes aos super-píxeis vizinhos.

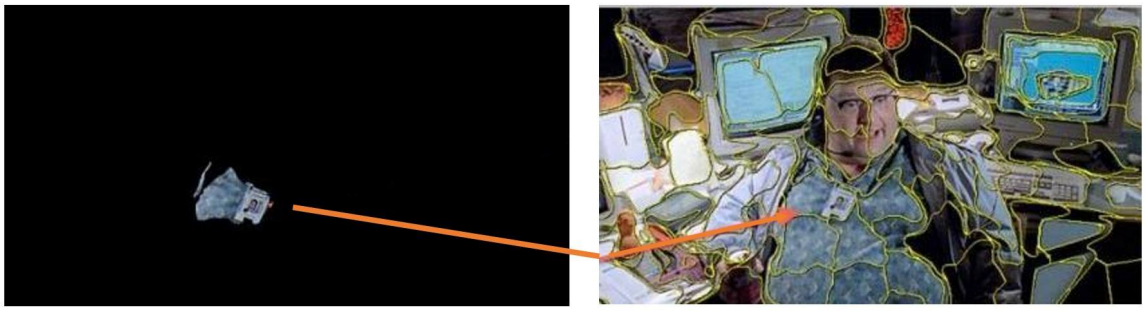

Figura 2.4: Imagem de superpixeis, componentes interpretáveis (superpixels contíguos[10]).

2. Criação do conjunto de dados de entrada do modelo - Instâncias perturbadas.

   Depois de criados os componentes interpretáveis, super-píxeis, são geradas várias perturbações que irão constituir o novo conjunto de entrada de dados no modelo. Estas perturbações consistem na desativação, ou perturbação, de super-píxeis, aleatoriamente, substituíndo a área correspondente destes por ruído.

   Além disso são registados vetores de perturbação para cada instância perturbada criada. Estes são constituídos por 0's e 1's, sendo que os 0's correspondem a super-píxeis desativados e 1's a super-píxeis ativados. O tamanho deste vetor é o numero de super-píxeis em que a imagem for fragmentada.

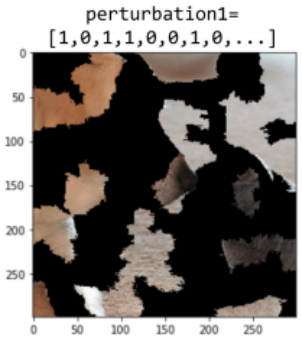

Figura 2.5: Exemplo de perturbação.[13]

3. Obter previsões para cada perturbação.

Testar na rede pré treinada as pertubações criadas em relação à instância de interesse(imagens pertencentes à mesma classe), de forma a obter o resultado das previsões.

4. Obter pesos para cada SP.

   O cálculo destes pesos é calculado através de uma função de distância do kernel. Essa função utiliza os vetores de perturbação criados anteriormente em relação à imagem original.

5. Construção de um modelo linear.

   O modelo linear será construído, tendo por base parâmetros calculados anteriormente. Sendo assim, este modelo é ajustado através dos valores do peso para cada SP, e também pelas perturbações e respetivas previsões com a instância de interesse.

6. Classificação dos super-píxeis

   Por último, para obter os super-píxeis mais determinantes para a explicação da previsão utilizamos os coeficientes retornados pelo modelo linear criado para obter os n primeiros super-píxeis com mais preponderância.

2.3.3 Conclusão

O AL é um ótimo algoritmo para depurar modelos de ML onde é útil ter todos os motivos da explicação da previsão em vez de alguns.

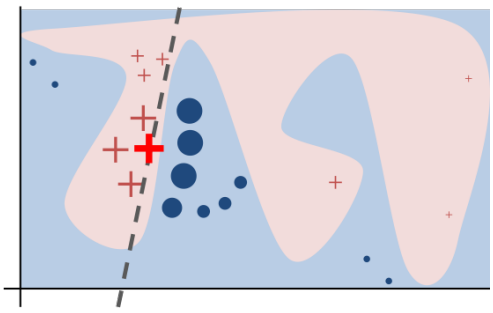

Figura 2.6: Fideldade local do LIME[13].

A fideldade demonstrada na imagem 2.3 mostra quão bem o modelo interpretável se aproxima das previsões da caixa preta do modelo de ML. Isto dá uma boa ideia de quão confiável o modelo interpretável é ao explicar as previsões da caixa preta na vizinhança da instância de dados de interesse.

2.4 Partial Dependence Plots

2.4.1 Breve Definição

O gráfico de dependência parcial (PDP ou gráfico PD) mostra o efeito que um ou dois recursos têm sobre o resultado previsto de um ML (JH Friedman, 2001). Um gráfico de dependência parcial pode mostrar, se a relação entre o alvo e um recurso é linear, monotónica ou mais complexa. Por exemplo, quando aplicado a um modelo de regressão linear, os gráficos de dependência parcial mostram uma relação linear.

A função de dependência parcial para regressão é definida como:

$$\hat{f}_{x_S}(x_S) = E_{x_C}\left[\hat{f}(x_S, x_C)\right] = \int \hat{f}(x_S, x_C) d\mathbb{P}(x_C)$$

Figura 2.7: Função de dependência parcial para regressão

O xs são os recursos para os quais a função de dependência parcial deve ser traçada e xc são os outros recursos usados no modelo ML f. Normalmente há apenas um ou dois recursos no conjunto S. O(s) recurso(s) em S são aqueles para os quais desejamos saber o efeito sobre a previsão. Os vetores de recursos xs e xc combinados constituem o espaço total de recursos x. A dependência parcial funciona marginalizando a saída do modelo ML sobre a distribuição dos recursos no conjunto C, de modo a que a função mostre a relação entre os recursos no conjunto S . Ao marginalizar sobre os outros recursos, obtém-se uma função que depende apenas dos recursos em S, interações com outros recursos incluídos. A função parcial fxs é estimada calculando médias nos dados de treinamento, também conhecido como método de Monte Carlo:

$$\hat{f}_{x_S}(x_S) = E_{x_C}\left[\hat{f}(x_S, x_C)\right] = \int \hat{f}(x_S, x_C) d\mathbb{P}(x_C)$$

Figura 2.8: Função representativa do método de Monte Carlo

A função parcial diz-nos que, para dado valor(es) de características S qual é o efeito marginal médio na previsão. Nesta fórmula, Xc^{Eu} são valores reais de recursos do conjunto de dados e n é o número de instâncias no conjunto de dados. Uma suposição do PDP é que os recursos em C não estão

correlacionados com os recursos em *S*. Se essa suposição for violada, as médias calculadas para o gráfico de dependência parcial incluirão pontos de dados que são muito improváveis ou mesmo impossíveis. Para classificação, em que o modelo ML gera probabilidades, o gráfico de dependência parcial exibe a probabilidade para uma determinada classe, dados diferentes valores para recurso(s) em *S*. Uma maneira fácil de lidar com várias classes é desenhar uma linha ou gráfico por classe.

O gráfico de dependência parcial é um método global: o método considera todas as instâncias e fornece uma declaração sobre a relação geral de um recurso com o resultado previsto (Molnar, 2020).

2.4.2 Exemplo

Por exemplo, observando um conjunto de dados de aluguer de bicicletas e o gráfico de dependência parcial para as quatro estações, obtém-se quatro números, um para cada estação. Para calcular o valor de "verão", substituí-se a temporada de todas as instâncias de dados por "verão"e calcula-se a média das previsões.

Na prática, o conjunto de recursos S geralmente contém apenas um recurso ou no máximo dois, porque um recurso produz gráficos 2D e dois recursos produzem gráficos 3D. Tudo além disso é bastante complicado.

De volta ao exemplo, no qual se prevê o número de bicicletas que serão alugadas em um determinado dia. Primeiramente, ajusta-se um modelo ML e, em seguida, analisa-se as dependências parciais. Nesse caso, ajusta-se uma floresta aleatória para prever o número de bicicletas e usa-se o gráfico de dependência parcial para visualizar as relações que o modelo apreendeu. A influência das características do clima nas contagens de bicicletas previstas é visualizada na figura a seguir (Molnar, 2020).

A seguinte figura demonstra PDPs para o modelo de previsão da contagem de bicicletas e temperatura, humidade e velocidade do vento. As maiores diferenças são na temperatura. Quanto mais quente, mais bicicletas são alugadas. Esta tendência sobe até 20 graus Celsius e, de seguida, nivela-se e cai ligeiramente a 30. As marcas no eixo x indicam a distribuição dos dados.

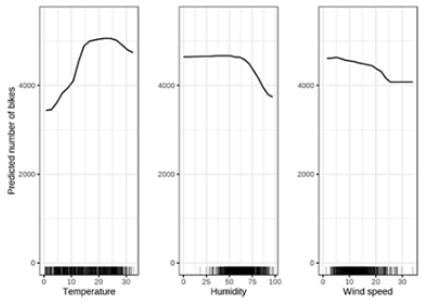

Figura 2.9: PDPs relativos ao modelo de previsão da contagem de bicicletas e temperatura[2]

Para clima quente, mas não muito quente, o modelo prevê, em média, um grande número de bicicletas alugadas. Os motociclistas em potencial são cada vez mais inibidos de alugar uma bicicleta quando a humidade ultrapassa 60 por cento. Além disso, quanto mais vento, menos pessoas gostam de andar de bicicleta, o que faz sentido. Curiosamente, o número previsto de aluguer de bicicletas não diminui quando a velocidade do vento aumenta de 25 para 35 km / h, mas não há muitos dados de treinamento, então o modelo ML provavelmente não poderia apreender uma previsão significativa para esta faixa. Pelo menos intuitivamente, seria de esperar que o número de bicicletas diminuísse com o aumento da velocidade do vento, especialmente quando a velocidade do vento é muito alta (Molnar, 2020).

Para ilustrar um gráfico de dependência parcial com uma característica categórica é examinado o efeito da característica nas estações nos alugueres de bicicletas previstos.

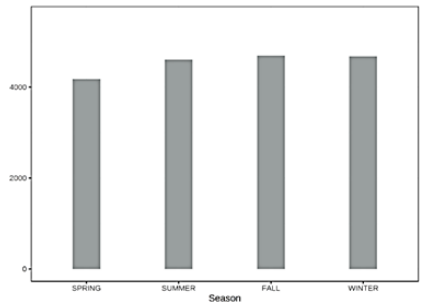

Figura 2.10: Gráfico com o efeito da característica nas estações[2]

2.4.3 Sinopse

O cálculo de gráficos de dependência parcial é intuitivo : a função de dependência parcial em um valor de recurso específico representa a previsão média, se forem forçados todos os pontos de dados a assumir esse valor de recurso.

Se o recurso para o qual for calculado o PDP não estiver correlacionado com os outros recursos, os PDPs representam como o recurso influencia, em média, a previsão. No caso não correlacionado, a interpretação é clara : o gráfico de dependência parcial mostra como a previsão média no seu conjunto de dados muda quando o jésimo recurso é alterado (Zhao et al.,2017).

2.5 Conclusão

Neste capítulo apresenta-se exposta a componente teórica básica para a realização deste projeto, logo não está representado todo o conceito de ML e nem são referenciados os modelos interpretativos específicos ou métodos interpretativos.

Sendo o conceito de ML tão popular da informática contemporânea, este ao apresentar um mecanismo de caixa negra, sem explicação das previsões obtidas, traz novos desafios, como por exemplo a sua interpretabilidade.

Este capítulo serve também como base de auxílio para os restantes capítulos, apresentados numa componente mais prática da implementação do algoritmo LIME.

Finalizando, podem ser apresentadas novos conceitos no decorrer dos próximos capítulos ausentes neste.

Capítulo

3

Tecnologias e Ferramentas Utilizadas

3.1 Introdução

No decorrer deste projeto, várias ferramentas foram utilizadas para a obtenção do trabalho final.

Essas ferramentas serão expostas no decorrer deste capítulo, bem como, os seus componentes.

3.2 LaTeX

Figura 3.1: Logotipo LaTeX

Ferramenta a partir da qual foi escrito o presente relatório de projeto. Consiste num sistema de criação de documentos de texto simples com aparência e formatação bastante profissional implícita no próprio sistema. Com isto possibilita ao utilizador centrar o seu trabalho na produção e distribuição ló-

gica do conteúdo do documento. Utilizado maioritariamente na produção de livros, artigos ou até mesmo para cartas.

Desenvolvido em 1983 por Leslie Lamport.

3.3 Overleaf

Figura 3.2: Logotipo Overleaf

Desenvolvido pelos matemáticos John Hammersley e John Lees-Miller em 2012, Overleaf é um editor de LaTeX, ferramenta enunciada em cima, e que conjuntamente com esta, possibilitou a escrita deste relatório. Foi utilizado o Overleaf (3.2) e um editor de LaTeX(3.1).

3.4 Python

Figura 3.3: Logotipo Python

Linguagem de programação de alto nível, orientada a objetos, de forte e dinâmica tipagem. Esta apresenta um modelo de desenvolvimento comunitário

e foi lançada por Guido Van Rossum em 1991. Conhecida pela facilidade na produção e leitura de código, bem como, na típica implementação em problemas de IA. Linguagem utilizada nos diferentes aspetos programáticos relativos a este projeto. Foi utilizado o Python (2.7).

3.5 Pycharm

Figura 3.4: Logotipo PyCharm

Desenvolvido pela empresa JetBrains, Pycharm é um Ambiente de Desenvolvimento Integrado (IDE), utilizado especificamente para programação na linguagem Python, que fornece uma ampla gama de ferramentas integradas, tais como, análise de código, depurador gráfico, sistema de controlo de versão, shell integrada, entre outros e ainda suporta desenvolvimento web ou de *Data Science* (DS).

Toda a parte de programação foi desenvolvida neste ambiente computacional, Pycharm (3.4).

3.6 TensorFlow

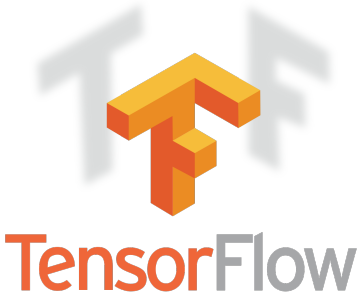

Figura 3.5: Logotipo Tensorflow

Biblioteca de ML, de código aberto compatível com Python, desenvolvida pela equipa Google Brain lançada em 2015. Esta reúne uma série de modelos e algoritmos de ML e de redes neuronais. A partir da utilização desta ferramente pode-se treinar e executar redes neuronais, como por exemplo, para processamento de linguagem natural, classificação manuscrita de dígitos, simulações baseadas em equações diferenciais parciais, modelos seqüência a sequência para tradução automática, reconhecimento de imagens, entre outros.

No âmbito deste projeto foi utlizado o TensorFlow (9.0.1).

3.7 Keras

Figura 3.6: Logotipo Keras

Keras é uma biblioteca de rede neuronal de código aberto, escrita em Python. É normalmente executado aquando do Tensorflow. Projetado para permitir a experiência com redes neurais profundas. Desenvolvido como parte do esforço de pesquisa do projeto *Open-ended Neuro-Electronic Intelligent Robot Operating System* (ONEIROS).

No âmbito deste projeto foi utilizado o Keras (2.1.6)

3.8 Conclusões

Todas as ferramentas enunciadas ao longo deste capítulos , foram essenciais para a conclusão deste projeto, tanto na parte programática como teórica. Importante denotar que nem sempre as versões das ferramentas utilizadas foram as mais utilizadas, devido ao manuseamento de um modelo pré-treinado ao longo da realização do projeto, mantendo-se assim as versões na qual este foi originalmente desenvolvido.

Capítulo

4

Implementação e Testes

4.1 Introdução

Neste capítulo será documentado todo o trabalho implementado a fim de cumprir o objetivo do projeto bem como a especificação do **MRPPI!** (**MRPPI!**) utilizado para obter os resultados finais.

4.2 Modelo de Verificação Perioocular de Pares de Imagens

Foi utilizado um **MRPPI!** ResNet50 pré treinado. Este modelo recebe pares de imagens, sendo que é um modelo apenas para verificação (1:1), ou seja, verifica se as imagens pertencem ou não à mesma classe. Um par de imagens só é considerado genuíno, pertencente à mesma classe, se pertencer à mesma pessoa e ao mesmo lado da face, se forem lados opostos e mesmo que seja a mesma pessoa é designado impostor.

A arquitetura do modelo está presente na figura:

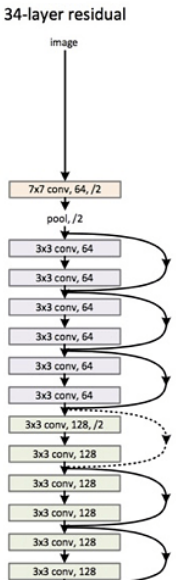

Figura 4.1: Arquitetura Resnet

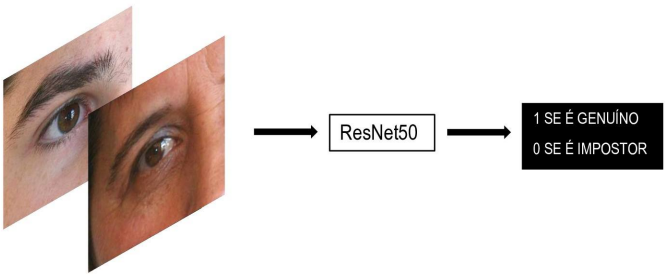

Figura 4.2: Modelo Resnet

O ficheiro de saída, predição, deste modelo consiste numa comparação todos contra todos, em que todas as imagens do conjunto de teste são comparadas entre si. Por cada comparação o modelo fornece um valor entre 0 e 1, score de saída da rede, em que 0 significa que não há probabilidade de as imagens serem da mesma classe e 1 que representa a certeza absoluta por parte do modelo em que ambas as imagens pertencem à mesma classe.

O modelo para obter as suas previsões faz uso de uma base de dados de imagens denominada "UBIPr". Esta, tem fotos de 344 pessoas diferentes, sendo que para cada pessoa foram tiradas várias fotos, de ângulos e luminosidades diferentes, e divididas na parte esquerda e direita da face. No total, é

constituída por 10 250 imagens, 5125 do lado esquerdo da face e 5125 do lado direito. O conjunto de teste consiste nas imagens da pessoa 0 até e inclusive à 172, enquanto que o conjunto de teste consiste nas imagens das pessoas 173 até à 344.

Figura 4.3: Exemplos de imagens da base de dados UBIPr

Denotar que o modelo foi treinado com imagens sem perturbações e testado, como vai ser explicado em baixo, com imagens com ruído ou perturbadas.

4.3 Perturbação do Conjunto de Teste

O AL, referido e explicado no capítulo 2, torna-se toda a base para o desenvolvimento do projeto. A aplicação deste envolve várias etapas, sendo que a primeira do processo passa pela perturbação do conjunto de teste, a instância em estudo, de forma a tornar as previsões do modelo interpretáveis ao utilizador.

4.3.1 Algoritmo SLIC

4.3.1.1 Introdução

Segmentação de imagem é o processo de particionar uma imagem em vários segmentos, segmentos esse que são um conjunto de píxeis ou um SP). O objetivo ao fazer esta segmentação é representar a imagem como algo mais significativo e fácil de analisar.

Simple Linear Iterative Clustering (SLIC) é um algoritmo a partir do qual são gerados super-píxeis com um custo computacional mais baixo a outros métodos comuns. Estes são formados a partir do agrupamento de píxeis, que partilham entre si similaridade de cores e proximidade no plano da imagem. Este processo é efetuado num espaço de cinco dimensões [labxy], onde xy é

a posição do pixel e [lab] representa o vetor de cor do pixel no espaço de cor CIELAB.

O espaço de cor CIELAB foi definifo pela Comissão Internacional de Iluminação (CIE), em 1976. Este expressa cor como três valores numéricos, L para a leveza e a e b para os componentes de cor verde-vermelho e azul-verde. CIELAB foi criado com o intuito de recriar a mudança de perceção da vista humana, sendo que a mesma quantidade de mudança numérica nos valores referidos corresponde aproximadamente à mesma quantidade da mudança percebida visualmente.

Voltando de novo ao algoritmo SLIC é a partir deste, sendo que a distância máxima possível entre duas cores no espaço CIELAB é limitada e a distância espacial no plano xy depende do tamanho da imagem, que conseguimos agrupar píxeis neste espaço 5D.

Este algoritmo utiliza como único parâmetro de entrada o número desejado de super-píxeis. Número desejado porque o algoritmo cria super-píxeis com uma área similar mas também de acordo com as propriedades mencionadas anteriormente, o que faz com que nem sempre seja possível ter o número de super-píxeis desejados.

4.3.1.2 Código Implementado e Respetiva Representação

Na figura está representado o código implementado e na figura o resultado desse trecho de código numa imagem exemplo pertencente ao conjunto de teste.

```
image_sample = cv2.imread(path_original + first_image)
#segmentacao
segments = slic(img_as_float(image_sample), n_segments=numSegments, sigma=5)
fig = plt.figure("Superpixels -- %d segments" % numSegments)
ax = fig.add_subplot(1, 1, 1)
ax.imshow(mark_boundaries(img_as_float(cv2.cvtColor(image_sample, cv2.COLOR_BGR2RGB)), segments)
plt.axis("off")
plt.savefig('segmented.png')
```

Figura 4.4: Código implementado

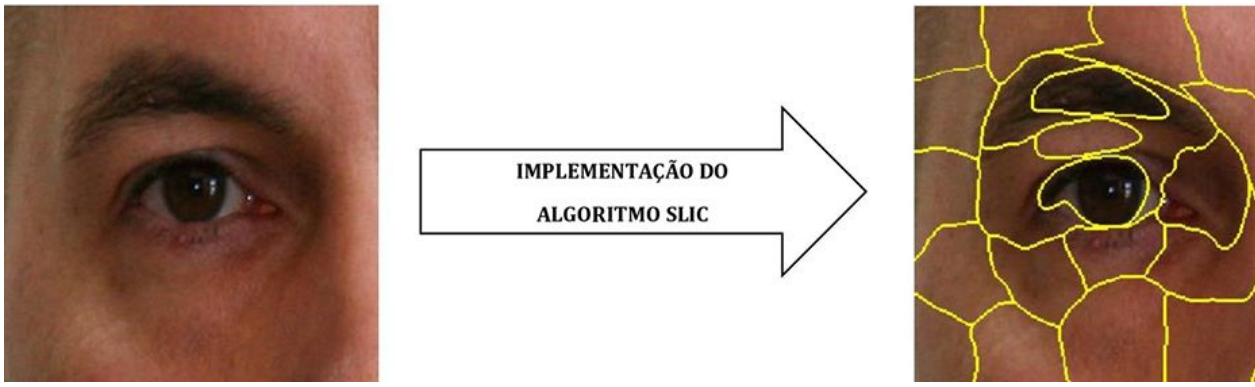

Figura 4.5: Resultado da segmentação de uma imagem com o algoritmo SLIC

4.3.2 Implementação de Ruído

Esta constitui a segunda parte da perturbação do conjunto de teste. Depois de efetuada a extração dos superpixeis, seguiu-se a criação da fonte se ruído para criar as perturbações.

4.3.2.1 Criação da Fonte de Ruído

O ruído que será implementado nas imagens segmentadas é proveniente de um ficheiro, ficheiro esse que é guardado como uma própria imagem. Este contém a média de valores RGB para cada posição ou píxel de todas as imagens da base de dados em uso. Nas figura estás representado o código implementado para a obtenção do mesmo e na figura a sua representação.

```
num_pixeis = 65536
width = 256
height = 256
red = [0.0] * num_pixeis
green = [0.0] * num_pixeis
blue = [0.0] * num_pixeis
num_fotos = len(dirs)


def media_train():
    for item in dirs:
        count = 0
        im = Image.open(path_images + item)
        new_img_list = list(im.getdata())
        for a in new_img_list:
            red[count] += a[0]
            green[count] += a[1]
            blue[count] += a[2]
            count += 1

    mediaarray = np.zeros(shape=(width, height, 3), dtype=np.uint8)
    count = 0
    for x in range(width):
        for y in range(height):
            mediaarray[x,y,0] = red[count]/num_fotos
            mediaarray[x,y,1] = green[count]/num_fotos
            mediaarray[x,y,2] = blue[count]/num_fotos
            count += 1

    mediaimage = Image.fromarray(mediaarray)
    mediaimage.save('media1.png')
```

Figura 4.6: Código implementado para a imagem de ruído

Figura 4.7: Imagem de ruído

4.3.2.2 Perturbação dos Super-Píxeis

A partir da imagem com a media dos valores RGB, foi possível implementar ruído no conjunto de teste, combinando-a com o passo de segmentação das imagens em superpixeis. No processo são escolhidos, para cada perturbação a ser criada um número aleatório de super-píxeis a serem perturbados, sendo que depois é utilizada uma máscara com o intuito de perceber as fronteiras dos super-píxeis que vão conter ruído e respetivamente essa(s) áreas são substituídas pela mesma área correspondente à imagem fonte de ruído.

Para melhor perceção do mecanismo, seguem-se as seguintes figuras, figura onde se encontra implementado o codigo e figuras onde se perturba 1 e 3 super-píxeis respetivamente.

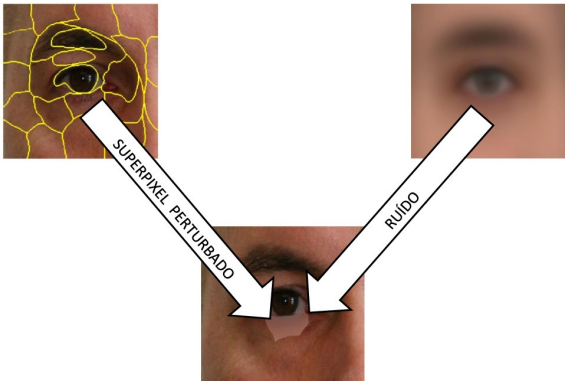

Figura 4.8: Imagem com um segmento perturbado

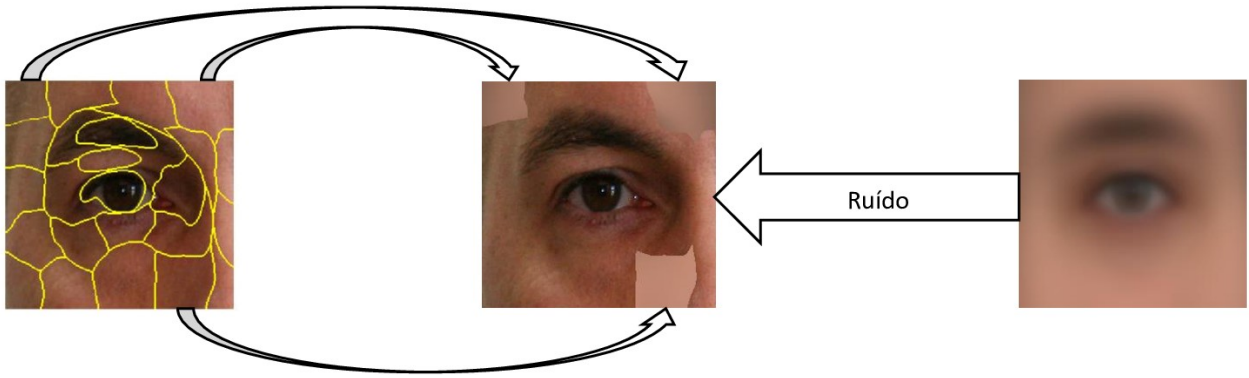

Figura 4.9: Imagem com três segmentos perturbados

4.3.3 Criação de Perturbações Aleatórias

4.3.3.1 Introdução

A partir da implementação dos mecanismos anteriores torna-se possível a criação de perturbações aleatórias sobre a instância desejada. Para cada perturbação criada é originado o respetivo vetor de perturbação. Estes vetores são constituídos por 0's e 1's, sendo que os 0's correspondem a super-píxeis com ruído e os 1's a super-píxeis não perturbados. O seu tamanho será do numero de super-píxeis em que as imagens foram fragmentadas.

4.3.3.2 Código Implementado

Na figura está representado o código implementado para a criação das perturbações, bem como, dos respetivos vetores de perturbação.

```
n_perturbacoes = random.randint(3, 10)
perturbations = np.ones((num_superpixeis,), dtype=int)
segVal = random.sample((np.unique(segments)), n_perturbacoes)
for indexes in segVal:
    perturbations[indexes] = 0

mask = np.zeros(image_sample.shape[:2], dtype="uint8")
mask[segments != segVal[0]] = 255
for x in range(1, n_perturbacoes):
    mask[segments == segVal[x]] = 0

# substitute mask for the correspondent mediaimage area
imgMask = Image.fromarray(mask)
result = Image.composite(image_open, media_open, imgMask)
result.save(path_disturbed + (first_image.split('.')[0]) + '(1).jpg')
```

Figura 4.10: Criação de vetores de perturbação, com entre 3 e 10 superpixeis perturbados e as respetivas imagens

4.3.3.3 Exemplos de Perturbações

Nas figuras estão representados um exemplo de perturbação bem como o seu vetor de perturbação para uma melhor compreensão do resultado final.

[1,0,0,0,0,1,....,1,0]

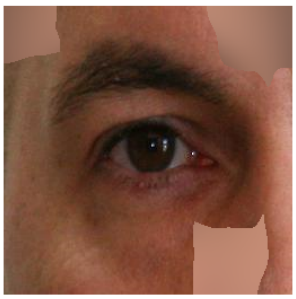

Figura 4.11: Exemplo de perturbação e respetivo vetor.

4.3.3.4 Conclusão

A criação das perturbações significa que foi criado o conjunto de teste. Com a sua utilização na rede podemos obter valores importantes e necessários para a implementação do AL, que são, o score das comparações entre estas imagens retornadas pelo modelo e, uma métrica de distância para avaliar quão longe está cada perturbação da instância original, obtida através dos vetores de perturbação originados.

4.3.4 Resultados na Rede

4.3.4.1 Introdução

Neste ponto do relatório serão analisados os resultados na rede de diferentes conjuntos de teste. Para efeitos de observação na divergência de métricas retornadas, além da utilização do conjunto de teste original sem perturbações, foram criados três conjuntos de teste perturbados. As três estruturas de entrada perturbadas tem correspondentemente em cada uma das suas imagens, um dois e três superpixeis perturbados respetivamente.

4.3.4.2 Resultados

Nas figuras seguintes serão apresentados os resultados das métricas retornadas sobre o conjunto de teste original e de três estruturas de entrada perturbadas, que têm em cada uma das suas imagens, um dois e três superpixeis perturbados respetivamente.

```
lazjunior@socialab-System-Product-Name:~/Desktop/proj_lime_artur$ python2 evaluate.py
Genuine: 22012, Impostor: 6246232
EER = 0.8778418963988 with Threshold 0.0226851348881 -- AUC = 0.97655121419  DEC = 3.27910475546
== EER: 0.8778418963988 +- 0.0, AUC: 0.97655121419 +- 0.0, DEC: 3.27910475546 +- 0.0
lazjunior@socialab-System-Product-Name:~/Desktop/proj_lime_artur$
```

Figura 4.12: Resultado das métricas com imagens com um super-píxel perturbado

```
lazjunior@socialab-System-Product-Name:~/Desktop/proj_lime_artur$ python2 evaluate.py
Genuine: 22012, Impostor: 6246232
EER = 0.108763220862 with Threshold 0.00658678682582 -- AUC = 0.964874985834  DEC = 2.56309595046
== EER: 0.108763220862 +- 0.0, AUC: 0.964874985834 +- 0.0, DEC: 2.56309595046 +- 0.0
lazjunior@socialab-System-Product-Name:~/Desktop/proj_lime_artur$
```

Figura 4.13: Resultado das métricas com imagens com dois super-píxeis perturbado

```
lazjunior@socialab-System-Product-Name:~/Desktop/proj_lime_artur$ python2 evaluate.py
Genuine: 22012, Impostor: 6246232
EER = 0.124093853702 with Threshold 0.00246611842891 -- AUC = 0.949341512971  DEC = 2.89275782638
== EER: 0.124093853702 +- 0.0, AUC: 0.949341512971 +- 0.0, DEC: 2.89275782638 +- 0.0
lazjunior@socialab-System-Product-Name:~/Desktop/proj_lime_artur$
```

Figura 4.14: Resultado das métricas com imagens com três super-píxeis perturbado

4.3.4.3 Análise

A partir da observação dos resultados da rede correspondentes aos diferentes conjuntos fez-se notar uma descida do valor da curva AUC, o que indica que quanto mais super-píxeis perturbados em relação às instância originais, maior aptência para um aparecimento de falsos positivos.

Também se notou uma notável descida da métrica decidabilidade(DEC), o que indica que as feature são menos discriminantes à medida que o ruído aumenta em comparação com as instâncias originais. Esta métrica mede o quão separados estão as duas distribuições (intra-classe e inter-classe) e quanto maior este valor mais capacidade de discriminação terá o nosso modelo sobre o conjunto de teste.

4.4 Modelo Linear

4.4.1 Introdução

Neste ponto será demonstrada a implementação necessária para ajustar um modelo linear interpretável usando as perturbações criadas, previsões e pesos para cada perturbação criada.

4.4.2 Recolha das Previsões Pertencentes à classe

4.4.2.1 Introdução

Para aplicar o algoritmo LIME, primeiro têm de se perceber qual a classe pertencente da instância em estudo, para assim poder comparar todas as imagens que lhe pertecem e saber os super-píxeis mais relevantes. Na figura seguinte está representado o trecho de código para extração da classe. Para extração é lido o ficheiro class.txt que contém a classe de cada imagem.

4.4.2.2 Código Implementado

Nas figuras está representado os trechos de código para extração da classe da imagem em estudo e de todas as imagens pertencentes a esta.

```
for item in dirs:

    test_image = original_images_path+item

    #abre a perturbacao sem superpixeis perturbados, imagem original
    cv2_image = cv2.imread(test_image)
    pillow_image = Image.open(test_image)

    #saber a classe da imagem
    with open(class_file_path, 'r') as fclass:
        all_classes = np.loadtxt(fclass, dtype=str)
    for r in all_classes:
        if r[0] == item:
            study_class = r[1]
            print (study_class)
    all_class_images = []
```

Figura 4.15: Trecho de código para extração da classe da imagem de instância

Em complemento com o trecho de código anterior, na figura seguinte está representado como extrair todas as imagens pertencentes a uma classe.

```
for r in all_classes:
    if r[1] == study_class:
        all_class_images.append(r[0])
print(all_class_images)
test_class_images = list(set(original_set).intersection(all_class_images))
print(test_class_images)
```

Figura 4.16: Trecho de código para extração de todas as imagens pertencentes à classe de interesse

4.4.3 Extração dos vetores de perturbação das imagens em estudo

4.4.3.1 Introdução

Para o cálculo do modelo linear precisamos dos vetores de perturbação, criados aquando da perturbação do conjunto de teste, das imagens em estudo. Estes vetores encontram-se em ficheiros originados quando se criou o conjunto de teste.

4.4.3.2 Código Implementado

Na figura seguinte está o trecho de código utilizado para a extração dos vetores de perturbação das imagens em estudo.

```
for r in all_classes:
    if r[1] == study_class:
        all_class_images.append(r[0])
print(all_class_images)
test_class_images = list(set(original_set).intersection(all_class_images))
print(test_class_images)
```

Figura 4.17: Implementação para extração dos vetores de perturbação

4.4.4 Extração das predições originadas

4.4.4.1 Introdução

Para a criação do modelo linear também é preciso das predições retornadas pelo modelo sobre o conjunto de teste perturbado. De recordar que para originar essas previsões criou-se um ficheiro auxiliar para fazer comparações todos contra todos.

4.4.4.2 Código Implementado

Na seguinte figura está presente o código implementado para recolha das previsões.

```
# -------------GET PREDICTIONS OF EACH IMAGE COMPARISON--------------#
predictions = np.empty(((len(test_class_images)*30), 1), dtype=float)
print(predictions.shape)
with open(results_path, 'r') as fr:
    all_predictions = np.loadtxt(fr, dtype=str, usecols=(0, 1, 3))
for r in all_predictions:
    for z in test_class_images:
        if r[0] == z.split('.')[0]:
            numbers = re.findall(r'\d+', r[1])
            predictions[int(numbers[0])-1] = r[2]
print(predictions)
```

Figura 4.18: Implementação para recolha das previsões necessários ao modelo linear

4.4.5 Cálculo de Pesos Para Cada Perturbação

4.4.5.1 Introdução

Para avaliar quão longe está cada perturbação da instância original, foi calculada uma métrica de distância, um peso, importante parâmetro para a execução do AL. Sendo que a imagem original é uma perturbação com todos os super-píxeis ativos e que as pertubações são vetores multidimensionais. A distância do cosseno é uma métrica que pode ser usada para esse fim. Depois

deste valor calculado, uma função do kernel é usada para traduzir esse valor entre 0 e 1.

4.4.5.2 Código Implementado

Na figura está representado o código implementado para obter um peso para cada perturbação.

```
#---------------GET PERTURBATIONS AND WEIGHTS OF EACH IMAGE COMPARISON----------------#
perturbations = []
for z in test_class_images:
    with open(perturbations_path+'perturbations'+z.split('.')[0]+'.txt', 'r') as fd:
        for lines in fd:
            linha = [int(i) for i in lines.split()]
            perturbations.append(linha)
print (perturbations)

original_image = np.ones(numSegments)[np.newaxis,:]
distances = sklearn.metrics.pairwise_distances(perturbations, original_image, metric='cosine').
kernel_width = 0.25
weights = np.sqrt(np.exp(-(distances**2)/kernel_width**2))
```

Figura 4.19: Trecho de código onde é medida a distância de cada perturbação em relação à original(peso)

4.4.5.3 Conclusão

Ao final deste processo temos um peso (importância) para cada perturbação no conjunto de dados, uma distância à imagem original, que nos vai ajudar na próxima etapa no cálculo do coeficiente dos super-píxeis.

4.4.6 Criação do modelo linear

4.4.6.1 Introdução

Para cálculo do modelo linear foi utilizado um modelo de regressão linear que usa como parâmetros os vetores de perturbação, as predições e as distâncias de cada perturbação à imagem original, para o ajustar.

4.4.6.2 Código Implementado

Na figura seguinte está o código implementado para ajustar o modelo linear que nos vai possibilitar recolher futuramente os super-píxeis mais relevantes para a descriminação da classe.

```
simpler_model = LinearRegression()
simpler_model.fit(X=perturbations, y=predictions, sample_weight=weights)
coeff = simpler_model.coef_[0]
print coeff
```

Figura 4.20: Trecho de código onde é o modelo linear

4.4.7 Cálculo de Coeficiente Para Cada Super-Píxel

4.4.7.1 Introdução

A fim de obter um modelo linear interpretável, usamos as informações obtidas nas etapas anteriores, nomeadamente, os vetores de perturbação e métrica de distância de cada perturbação e as previsões fornecidas pela rede. Com isto é possível calcular um coeficiente para cada super-píxel, a fim de perceber quais os píxeis mais determinantes para a previsão da classe.

4.4.7.2 Código Implementado

Na figura está representado o código implementado para obter os super-píxeis com mais peso ou importância para a previsão da classe.

```
#---------------GET PERTURBATIONS AND WEIGHTS OF EACH IMAGE COMPARISON----------------#
perturbations = []
for z in test_class_images:
    with open(perturbations_path+'perturbations'+z.split('.')[0]+'.txt', 'r') as fd:
        for lines in fd:
            linha = [int(i) for i in lines.split()]
            perturbations.append(linha)
print (perturbations)

original_image = np.ones(numSegments)[np.newaxis,:]
distances = sklearn.metrics.pairwise_distances(perturbations, original_image, metric='cosine').ravel()
kernel_width = 0.25
weights = np.sqrt(np.exp(-(distances**2)/kernel_width**2))
```

Figura 4.21: Trecho de código onde é medido o peso de cada superpixel

4.4.8 Classificação dos super-píxeis mais relevantes

4.4.8.1 Introdução

A fim de obter os super-píxeis mais relevantes para a descriminação de uma instância sobre a sua classe são utilizados os quatro maiores coeficientes retornados pelo modelo linear. Depois disso é utilizada uma máscara para desativar os super-píxeis fora deste grupo. No final fica guardada a representação criada.

4.4.8.2 Código Implementado e Respetiva Representação

Nas figuras estão representados o código implementado para obter os 4 píxeis com mais peso e a respetiva representação.

```
#---------------GET TOP SUPERPIXELS----------------#
num_top_features = 4
top_features = np.argsort(coeff)[-num_top_features:]
print(top_features)
#Show only the superpixels corresponding to the top features
initial_mask = np.zeros(numSegments)
initial_mask[top_features] = True #Activate top superpixels
active_pixels = np.where(initial_mask == 1)[0]
#final_mask = np.zeros(segments.shape)
final_mask = np.zeros(segments.shape[:2],dtype="uint8")
for active in active_pixels:
    final_mask[segments == active] = 1
#skimage.io.imshow(perturb_image(Xi/2+0.5, mask, segments))

#substitute mask for the correspondent mediaimage area
imgMask = Image.fromarray(final_mask)
write_image = cv2.bitwise_and(cv2_image, cv2_image, mask = final_mask)

#result = Image.composite(pillow_image, pillow_image, imgMask)
cv2.imwrite('topsuperpixels'+item.split('.')[0]+'.jpg', write_image)
```

Figura 4.22: Código implementado para a extração dos 4 superpíxeis mais relevantes para uma instância em estudo.

Figura 4.23: Demonstração dos 4 superpíxeis mais relevantes para a discriminação.

4.4.8.3 Conclusão

É a partir desta métrica que é conseguido fazer a distinção de classificação de super-píxeis de acordo com a sua determinância para pertencer a determinada classe.

4.5 Conclusões

O estudo do AL permitiu observar que podemos tornar um modelo não interpretativo num interpretativo, porque a partir deste é retornada uma explicação das previsões fornecidas. Esta explicação é nos dada na forma de uma imagem em que a área (super-píxeis) que têm um peso/importância mais forte para a previsão aparecem ativas enquanto que as restantes áreas aparecem a negro. Este projeto mostra como este mecanismo pode ajudar a aumentar a confiança, por parte do utilizador, num modelo de ML.

Apesar de não terem sido estudados neste relatório, existem outros algoritmos para extração de super-píxeis, como o quickshift, que apresenta resultados igualmente satisfatórios mas foi optado o SLIC devido ao seu custo computacional mais baixo.

Capítulo

5

Conclusões e Trabalho Futuro

5.1 Conclusões Principais

Antes de começar a desenvolver a parte prática do trabalho, o conceito de Machine Learning pareceu-me algo bastante complexo. Isso veio a averiguar-se aquando do manuseamento do modelo pré-treinado que foi fornecido para a execução deste projeto. Apesar de tudo, e de alguma dificuldade inicial na percepção da rede, foi bastante interessante o desenrolar do trabalho para este projeto, principalmente a nível pessoal.

Penso que a aplicação destes métodos interpretativos agnósticos ao modelo fazem todo seintido na prática de aplicações de alto risco. Visto que, estes modelos de aprendizagem são muito utilizados nos dias de hoje, pode haver alguma relutância na parte de utilização destes quando as consequências podem ser graves. Logo, a utilização destes métodos vêm dar uma brisa de confiança aos modelos de ML, em que a partir da utilização destes, o utilizador tem um papel ativo tanto na aprendizagem de como o modelo fez tais previsões, bem como, de optar em escolher se a previsão dada é ou não válida para o problema em questão.

Ao longo do relatório foi dada uma enfatização à interpretabilidade dos modelos de ML, como se os não-interpretáveis já não fossem úteis. Porém, a utilização de modelos interpretáveis e não-interpretáveis tem de coexistir, na medida em que os não-interpretáveis deverão ser utilizados em aplicações de baixo risco e os interpretáveis nas de alto risco. Isto porque, seria um desperdício de esforço, a utilização de métodos interpretáveis em modelos que quando erram não apresentem consequências graves.

Hoje em dia vivemos num mundo que necessita da IA e que vive dela. Este tipo ramos dela, como o ML vêm acrescentar resolução para uma grande va-

riedade de problemas que sem isto seriam impossíveis, o que por si só já é fantástico, sendo que este tipo de métodos interpretáveis vem ainda acrescentar melhoria para estes mecanismos.

5.2 Trabalho Futuro

A técnica do AL apresenta grandes virtudes que podem ser aproveitadas, como referido, para aplicações críticas como a àrea de saúde por exemplo. Penso que a sua aplicação ao nível de exames complementares de diagnóstico, iria ter enorme sucesso a nível de percepção de sintomas para doenças.

Na realização deste projeto centrou-se a atenção para determinar regiões das imagens que são mais importantes para decidir uma comparação Genuina. Para trabalho futuro penso que faz todo o sentido perceber quais as regiões que possam ter mais preponderância entre pares de imagens impostores.

Também para trabalho futuro seria importante o trabalho de anotação dos dados do modelo, bem como, análises qualitativas e subjetivas sobre os resultados, o que não foi feito por falta de organização minha.

Outro ponto interessante para trabalho futuro passaria por perceber quais as imagens de entrada mais difíceis e a razão para tal.

Bibliografia

[1] Centro de Computação Gráfica (2017). Machine learning: o que é e para que serve?
http://www.ccg.pt/machine-learning-o-que-e/

[2] Christoph Molnar (2020). Interpretable Machine Learning. A Guide for Making Black Box Models Explainable.
https://christophm.github.io/interpretable-ml-book/index.html

[3] Composite two images according to a mask image with Python, Pillow (2019).
https://note.nkmk.me/en/python-pillow-composite/

[4] Friedman, Jerome H. "Greedy function approximation: A gradient boosting machine."Annals of statistics (2001)
https://projecteuclid.org/euclid.aos/1013203451#ui-tabs-1

[5] Hulstaert (2018). Understanding model predictions with LIME.
https://towardsdatascience.com/understanding-model-predictions-with-lime-a582fdff3a3b

[6] Jain (2019). Superpixels and SLIC. What is a Superpixel?
https://medium.com/@darshita1405/superpixels-and-slic-6b2d8a6e4f08

[7] R. Achanta et. al. (2012) "SLIC superpixels compared to state-of-the-art superpixel methods."
https://www.researchgate.net/publication/225069465_SLIC_Superpixels_Compared_to_State-of-the-Art_Superpixel_Methods

[8] Ribeiro, Singh and Guestrin (2016). Introduction to Local Interpretable Model-Agnostic Explanations (LIME).
https://www.kdnuggets.com/2016/08/introduction-local-interpretable-model-agnostic-explanations-lime.html

[9] Ribeiro, Singh and Guestrin (2016). "Why Should I Trust You?" Explaining the Predictions of Any Classifier. Universidade de Washington, Seattle.
https://arxiv.org/pdf/1602.04938.pdf

[10] Rosebrock (2014). Accessing Individual Superpixel Segmentations with Python.
https://www.pyimagesearch.com/2014/12/29/accessing-individual-superpixel-segmentations-python/

[11] Zhao, Qingyuan, and Trevor Hastie. "Causal interpretations of black-box models."
https://www.tandfonline.com/doi/full/10.1080/07350015.2019.1624293

[12] Ribeiro, Singh and Guestrin (2016). Local Interpretable Model-Agnostic Explanations (LIME): An Introduction. A technique to explain the predictions of any machine learning classifier.
https://www.oreilly.com/content/introduction-to-local-interpretable-model-agnostic-explanations-lime/

[13] Cristian Arteaga (2019). Interpretable Machine Learning for Image Classification with LIME.
https://towardsdatascience.com/interpretable-machine-learning-for-image-classification-with-lime-ea947e82ca13

Bibliografia