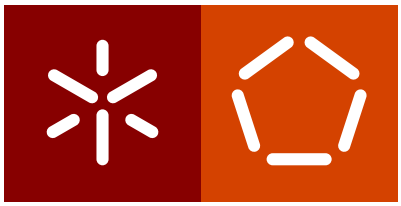
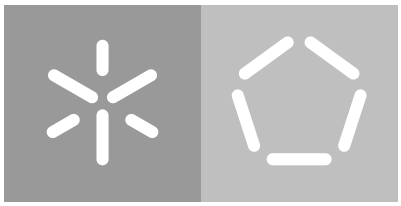


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Flávio Joel Martins Peixoto

Classificador da Condição do Piso para um Sistema de Condução Autónoma

Outubro de 2020



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Flávio Joel Martins Peixoto

Classificador da Condição do Piso para um Sistema de Condução Autónoma

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho efetuado sob a orientação do
Professor António Joaquim André Esteves

Outubro de 2020

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

AGRADECIMENTOS

Em primeiro lugar, quero agradecer ao Departamento de Informática e à Universidade, pois, sem todo o conhecimento e experiência que me proporcionaram ao longo do Mestrado em Engenharia Informática, não seria possível iniciar um projeto ambicioso e de contexto real.

Ao meu orientador Professor António Joaquim André Esteves, por toda a dedicação, empenho, prontidão e tempo disponibilizado no auxílio e acompanhamento de todas as fases de desenvolvimento desta dissertação

Ao Professor Jorge Cabral e Sérgio Branco por me terem acolhido no contexto do projeto *Sensible Car*, tanto pelo esclarecimento de dúvidas, como também ao fornecerem-me todas as informações necessárias ao desenvolvimento deste trabalho.

Ao Alberto Pereira, Almeno Soares e Ricardo Sousa, que sempre demonstraram interesse e preocupação pelo meu sucesso nesta fase da minha experiência académica e apoio contínuo.

À Helena Ribeiro, ao acompanhar-me todos os dias desde o início, por me motivar e incentivar a procurar soluções nos momentos menos bons e por ser um pilar nos meus dias. Ao João Gomes, Alexandre Teixeira, Tiago Fraga e Daniela Gomes, por me terem dado momentos incríveis durante esta fase da minha vida académica e principalmente por terem um grande impacto na minha vida pessoal.

Aos meus pais e à minha irmã que desde sempre me apoiaram incondicionalmente e que sempre me proporcionaram todas as condições necessárias ao meu sucesso, tanto nas vertentes profissionais e académicas, como também pessoais.

RESUMO

O presente trabalho de dissertação surgiu no contexto do projeto `Sensible Car`, uma parceria entre a `Bosch` e a Universidade do Minho (UM), onde se está a desenvolver um sensor da condição do piso (RCS) em que circula um veículo automóvel. Com esta dissertação pretendia-se verificar se a aplicação de dados meteorológicos aliados à informação ótica apresenta vantagens na classificação da condição do piso, para além da utilização da informação ótica. Também se pretendia demonstrar se, com recursos computacionais limitados, é possível implementar um classificador de piso com a fiabilidade e a capacidade de resposta exigidas. Para atingir os objetivos propostos aplicou-se aprendizagem automática com supervisão e utilizaram-se dados de treino que combinam (i) os rácios da intensidade da luz recebida (após reflexão no piso) sobre a intensidade da luz emitida pelos dispositivos óticos do RCS, para quatro comprimentos de onda distintos, com (ii) dados meteorológicos. Os dados óticos são essenciais para a circulação com segurança em veículos com condução autónoma. Isto porque a deteção da condição do piso onde se circula permite ao veículo tomar melhores decisões em tempo real. Para além de se comparar o desempenho de cada modelo treinado só com dados óticos, com o desempenho do mesmo modelo treinado com dados resultantes da fusão entre dados óticos e meteorológicos, testaram-se diversos modelos, para selecionar o que mais se adequa à classificação da condição do piso. Numa fase inicial, selecionaram-se os modelos que apresentaram melhor desempenho, i.e. melhor precisão e `recall`, na classificação de amostras dos vários tipos de piso. Os modelos aqui selecionados foram `SVM` Gaussiano (0.96 de precisão e 0.93 de `recall`), Regressão Logística (0.91 e 0.88), Árvore de Decisão (0.91 e 0.85) e `XGBoost` (0.94 e 0.94). Posteriormente, implementaram-se e testaram-se os melhores modelos no dispositivo `Nvidia Jetson Nano`. Nesta fase, além de se confirmar as percentagens de acerto dos modelos a classificar a condição do piso, verificou-se se eram capazes de classificar as amostras ao ritmo a que o sensor de condição de piso gera os dados óticos. Os resultados obtidos mostraram que os modelos desenvolvidos são capazes de acompanhar o ritmo de geração de dados do sensor da condição do piso, em que o modelo SVM faz 1040 classificações por segundo, a Regressão Logística faz 2080, a Árvore de Decisão efetua 1950 e o `XGBoost` faz 223. O modelo selecionado no fim foi o SVM Gaussiano, pois apesar de não ser o modelo com maior número de classificações por segundo, é o que possui o melhor desempenho geral na classificação da condição do piso.

Palavras-chave: aprendizagem automática, sensor da condição do piso, dispositivo *edge*, fusão sensorial, condições meteorológicas.

ABSTRACT

This dissertation arose in the context of the `Sensible Car` project, a partnership between Bosch and University of Minho, where a road condition sensor (RCS) is being developed. With this dissertation, it was intended to verify if the application of meteorological data allied to optical information presents advantages in the classification of the road condition, besides the use of optical information. It was also intended to demonstrate whether, with limited computing resources, it is possible to implement a road condition classifier with the required reliability and response capacity. To achieve the proposed objectives, supervised machine learning was applied and it was used training data that combines (i) the ratio of the received light intensity (after reflection on the pavement) over the light intensity emitted by the RCS optical device, for four different wavelengths, with (ii) meteorological data. Optical data is essential for safe autonomous driving. This is because detecting the condition of the road surface enables the vehicle to make better decisions in real time. In addition to comparing the performance of each model trained with optical data only, with the performance of the same model trained with data resulting from the fusion between optical and meteorological data, several models have been tested in order to select the one that best fits the tread condition classification. Initially, the models with the best performance, in terms of precision and recall, were selected to classify samples of the different road conditions, namely the SVM Gaussian model (precision of 0.96 and recall of 0.93), the Logistic Regression (0.91 and 0.88), the Decision Tree (0.91 and 0.85) and XGBoost (0.94 and 0.94). Subsequently, the best models were implemented and tested on the Nvidia Jetson Nano device. At this stage, in addition to confirming the models' accuracy to classify the road condition, it was checked whether they were able to classify samples at the rate that the road condition sensor generates the optical data. The results showed that the developed models are capable of keeping up with the pace of data generation from the road condition sensor, where the SVM model makes 1040 classifications per second, the Logistic Regression does 2080, the Decision Tree makes 1950 and the XGBoost 223. The final model that was selected is the Gaussian SVM. Although it is not the model with the highest number of classifications per second, it reached the best overall performance in road condition classification.

Keywords: machine learning, road condition sensor, edge device, sensor fusion, weather conditions.

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE LISTAGENS

SIGLAS

ANN Redes Neuronais Artificiais. 1, 33, 34, 71, 86

EDA Análise e Exploração de Dados. 1, 16, 47, 51, 54

GPS Sistema de Posicionamento Global. 1, 14, 20, 21, 23

K-NN K-Nearest Neighbours. 1, 28, 64, 68, 74, 83, 86

LiDAR Light Detection And Ranging. 1, 20–23

MAR Valor em falta por influência de outras variáveis. 1, 52
MCAR Valor em falta completamente aleatório. 1, 52, 58
MEI Mestrado em Engenharia Informática. 1, 13
MNAR Valor em falta não aleatório. 1, 52

NDA Non Disclosure Agreement. 1, 48, 95

RADAR Radio Detection And Ranging. 1, 20–23
RCS Sensor da Condição do Piso. 1, 13, 15, 16, 50, 86, 87, 89, 91, 95

SPA Single Page Application. 1, 44, 45
SVM Máquinas de Vetor de Suporte. 1, 29, 65, 68, 75, 83, 86–88, 92, 95

TPU Tensor Processing Unit. 1, 39

UM Universidade do Minho. 1, 4, 13

1

INTRODUÇÃO

Neste capítulo de introdução descreve-se o contexto e a motivação do trabalho a realizar, os objetivos que se esperam atingir no final da dissertação, bem como os resultados esperados dos mesmos. Será ainda descrita a metodologia a adotar no desenvolvimento do trabalho e o planeamento inicial. No final, é apresentada a estrutura do documento.

1.1 CONTEXTO E MOTIVAÇÃO

O presente trabalho enquadra-se no projeto `Sensible Car`, uma parceria entre a `Bosch` e a UM, e é desenvolvido no âmbito da unidade curricular de Dissertação, do segundo ano do Mestrado em Engenharia Informática (MEI) da Escola de Engenharia da Universidade do Minho.

O projeto `Sensible Car` é constituído por vários sub-projetos, que têm como principal objetivo responder às necessidades exigidas pelos veículos no contexto da condução autónoma. Estes sub-projetos contemplam o desenvolvimento de sistemas de perceção do meio envolvente ao veículo, de localização em tempo real e dotados de capacidade de atuação. A presente dissertação foca-se no desenvolvimento de um sistema capaz de avaliar a condição do piso onde um veículo circula, não no sentido de identificar deformações na estrada, mas sim perceber se o piso se encontra seco, molhado, com neve, ou com gelo. O sistema a desenvolver faz parte do objetivo maior, que é a conceção de um Sensor da Condição do Piso (RCS). O RCS é visto como um sensor essencial para assegurar uma condução autónoma segura e estável. O facto de poder vir a ser possível aos veículos autónomos analisarem a condição do piso, permite uma melhor regulação da condução, com base na determinação das condições de atrito existentes.

Desde o início, que a segurança da condução automóvel depende do comportamento adotado pelo condutor, mas em muitas situações também depende das condições meteorológicas e, por consequência da condição do piso onde o veículo circula. Deste modo, a conjugação destas duas componentes de elevado risco, comportamento humano e condições meteorológicas, tem um grande impacto na sinistralidade rodoviária nas estradas a nível mundial.

Com o propósito de tomar controlo e, de certa maneira prever o comportamento dos veículo, a condução autónoma representa uma mais valia no aumento da segurança e na diminuição das fatalidades nas estradas, pois a imprevisibilidade da conduta humana nas mais variadas condições será "anulada". No entanto, a avaliação das condições meteorológicas e, por conseguinte da condição do piso, continuam a ser elementos cruciais na segurança. Atualmente, as condições meteorológicas são classificadas utilizando os dados dos mais variados sensores existentes no automóvel, tais como as câmaras ou o Sistema de Posicionamento Global (GPS). Contudo, a classificação da condição de atrito no piso continua a ser uma variável bastante sensível na condução autónoma. Assim, dispor dum sensor dedicado à classificação do piso representa uma mais valia para uma condução autónoma mais segura e estável.

1.2 OBJETIVOS

Com este trabalho de dissertação pretende-se identificar o modelo de aprendizagem automática que melhor classifique a condição do piso, tendo por base um conjunto de características provenientes de fusão sensorial. Após identificar o melhor modelo, que neste caso será o modelo que requerer menos recursos computacionais e que ao mesmo tempo alcança o melhor desempenho possível na classificação do piso, procede-se à sua implementação e teste num dispositivo *edge*.

Para além de se avaliar os resultados da classificação em cada modelo treinado, tem que se ter em consideração quais os modelos que lidam melhor com as situações de erro e ruído que irão ocorrer em cenários de utilização real.

Apesar de os dados a utilizar já serem provenientes de vários sensores, deverá ser estudada a viabilidade de acrescentar na entrada do classificador dados provenientes de outros sensores existentes no automóvel, com o objetivo de melhorar a classificação do modelo.

1.3 RESULTADOS ESPERADOS

Um dos resultados esperados no final da dissertação é a identificação de um modelo de aprendizagem automática otimizado, de entre todos os modelos avaliados. Este modelo terá de ser capaz de classificar corretamente a condição do piso e em tempo útil. Para além disso, o modelo não deverá exigir um poder computacional muito elevado, dado que estará a ser executado num dispositivo com recursos computacionais bastante limitados e, para evitar assim ter que incluir no automóvel *hardware* com maior capacidade de processamento.

Outro resultado esperado consiste no registo e explicação dos diversos resultados obtidos da implementação e teste dos modelos de aprendizagem automática no dispositivo *edge*,

`NVIDIA Jetson Nano` ou `Google Coral Edge TPU`, de maneira a expor o comportamento dos mesmo no contexto da classificação da condição do piso.

1.4 MÉTODOS E TÉCNICAS

Inicialmente, o trabalho de dissertação contempla a leitura e análise da bibliografia recomendada na proposta de dissertação. Posteriormente, é feita uma pesquisa para obter novas referências bibliográficas relativas ao tema da dissertação, acompanhadas de uma análise crítica do seu conteúdo. De seguida, efetua-se a análise e descrição dos atributos presentes no conjunto de dados a utilizar no trabalho de dissertação, seguida da exploração e tratamento dos mesmos para extração de caraterísticas úteis ao sucesso do classificador a desenvolver.

A próxima etapa consiste em implementar diversos modelos de aprendizagem automática, onde cada modelo é testado e posteriormente validado com novos dados. Após esta fase, é realizada uma análise dos modelos mais promissores, avaliando o esforço computacional requerido e o desempenho obtido na classificação da condição do piso, já com a utilização do dispositivo *edge*, de modo a reproduzir um cenário próximo daquele em que o RCS irá funcionar.

Todas estas fases são acompanhadas pelo orientador, recorrendo a reuniões semanais, uma vez que elas facilitam o cumprimento do planeamento do trabalho estabelecido e um *feedback* atempado sobre o trabalho desenvolvido. No final, todo o trabalho desenvolvido será validado pelos investigadores do projeto *Sensible Car*.

1.5 PLANEAMENTO DO TRABALHO

O desenvolvimento da presente dissertação inicia-se com a leitura e análise da bibliografia recomendada. Segue-se a pesquisa e avaliação de novas referências relativas ao estado da arte do problema abordado. Esta pesquisa incide principalmente sobre (i) os sistemas de condução autónoma, (ii) os sensores envolvidos nestes sistemas, (iii) o impacto das condições meteorológicas sobre a condução, convencional ou autónoma, e de que maneira estas condições afetam o bom desempenho dos sensores envolvidos nas tomadas de decisão. Para além disto, é necessário focar a pesquisa no processos processos de fusão sensorial e na computação *edge*.

Depois de efetuar o levantamento do estado da arte será feita a análise, documentação e tratamento da informação contida no conjunto de dados fornecido pelos investigadores do projeto `Sensible Car`. Posteriormente procede-se à implementação, teste, e avaliação de modelos de aprendizagem automática na classificação da condição do piso, de modo a identificar os mais apropriados para o problema a resolver. Numa segunda fase da avaliação

de modelos, estes serão implementados no dispositivo *edge* (`Google Coral Edge TPU` ou `NVIDIA Jetson Nano`), de modo a perceber o seu comportamento em condições o mais aproximadas de um contexto real, a condução automóvel.

Todas as fases do processo de desenvolvimento desta dissertação são acompanhadas pelo orientador com recurso a reuniões semanais, de modo a avaliar o progresso das tarefas definidas durante o planeamento e corrigir com eficácia quaisquer desvios na planificação.

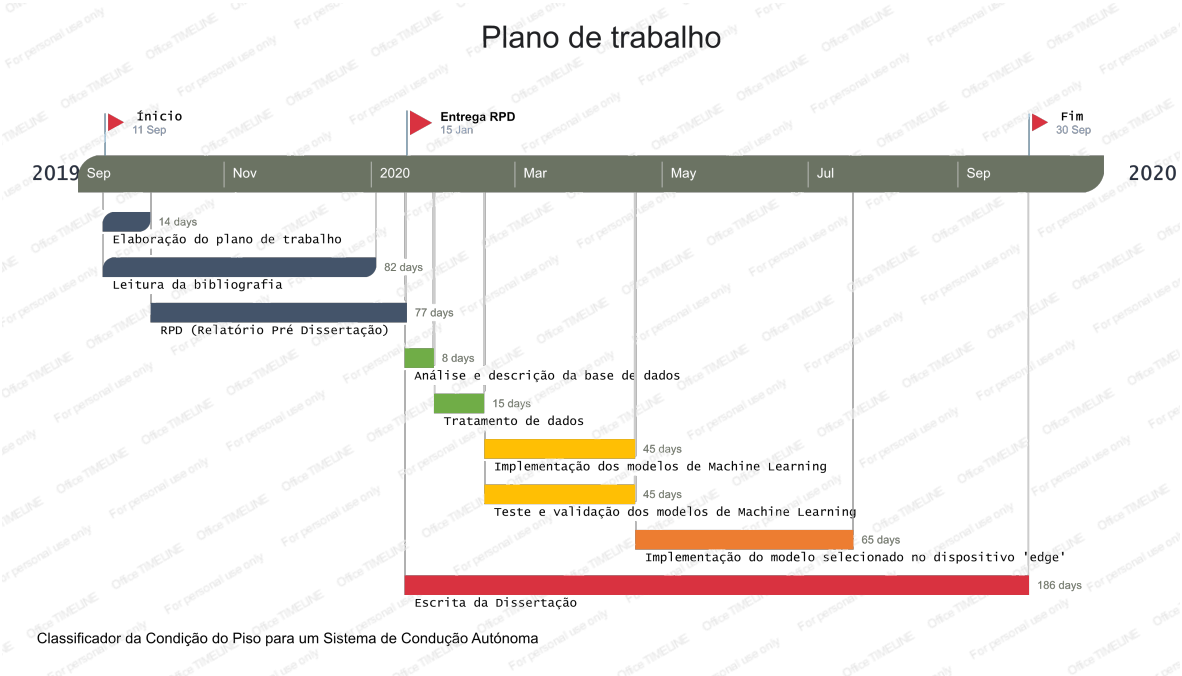


Figura 1: Planeamento do trabalho.

1.6 Organização do documento

A presente dissertação encontra-se dividida em cinco capítulos. O capítulo de Introdução tem como objetivo apresentar o trabalho proposto, acompanhado da sua contextualização e motivação, quais os objetivos e resultados esperados, bem como os métodos e técnicas utilizadas na sua realização.

O segundo capítulo, Estado da Arte, apresenta a revisão da literatura efetuada ao longo da dissertação, onde se destacam todos os pontos relevantes para compreender a motivação do presente trabalho e o estado atual da área da condução autónoma e seus componentes.

No terceiro capítulo, Metodologia, resumem-se as tecnologias utilizadas ao longo do desenvolvimento do trabalho. De seguida, descrevem-se os processos de recolha dos dados meteorológicos e de fusão dos mesmos com os dados provenientes do RCS, os rácios dos sinais infravermelhos, a Análise e Exploração de Dados (EDA) aplicada ao conjunto de

dados provenientes da fusão e a preparação dos mesmos para a aplicação dos modelos de classificação. São ainda explicados os passos necessários à configuração do dispositivo *edge* utilizado.

No quarto capítulo, Discussão de Resultados, interpretam-se os resultados obtidos com a aplicação dos vários classificadores aos conjuntos de dados descritos no capítulo anterior. Por último, faz-se uma reflexão sobre a metodologia adotada para selecionar o melhor modelo de classificação da condição do piso.

No capítulo de Conclusão serão discutidos os objetivos atingidos face aos propostos, acompanhados da identificação das limitações e de ideias para trabalho futuro.

2

ESTADO DA ARTE

O presente capítulo tem como objetivo principal contextualizar o trabalho a desenvolver nesta dissertação, através da apresentação e análise do estado da arte na área aplicacional da condução autónoma. Juntamente com a exposição da bibliografia relacionada, apresentam-se os pontos fortes e os pontos fracos de cada abordagem analisada. Deste modo, ficarão claras as motivações do trabalho a desenvolver na dissertação.

2.1 CONDUÇÃO AUTÓNOMA E CONDIÇÕES METEOROLÓGICAS

Para uma melhor compreensão e organização do conhecimento adquirido, a revisão da literatura será organizada de acordo com as temáticas relacionadas com o tema proposto na presente dissertação. Assim sendo, a revisão começa com o impacto das condições meteorológicas na condução dita convencional, seguida do impacto das mesmas condições na condução autónoma. A partir daqui, são descritos os sensores presentes nos veículos de condução autónoma, bem como o grau de sensibilidade destes perante condições meteorológicas adversas. Por fim, é abordado o tema da fusão sensorial, num contexto em que que procura minimizar os problemas causados à condução pelas condições meteorológicas adversas, assim como a explicação sobre como se podem utilizar variáveis meteorológicas para enriquecer e melhorar a perceção das condições meteorológicas do momento.

2.1.1 *Condições meteorológicas na condução convencional*

Segundo dados do Departamento de Transportes dos Estados Unidos da América, dum total aproximado de 6 milhões de acidentes anuais, 21% estão relacionados com a presença de condições meteorológicas adversas no momento em que ocorreu o evento. Com o intuito de demonstrar o grau de impacto destas condições no quotidiano, este tipo de sinistralidade rodoviária (à parte a negligência dos condutores) resulta num número de fatalidades maior do que os desastres naturais, como por exemplo os tornados. Estas fatalidades ocorrem sobretudo durante períodos de precipitação (intensa, moderada e em alguns casos fraca),

e dependem também da sua duração, da existência de nevoeiro, de granizo, de neve, e de fenómenos bruscos de alteração das condições meteorológicas.

No entanto, o papel do condutor nestas condições é fundamental, não só para a sua segurança e para a integridade do seu veículo, mas também para a segurança das pessoas e bens que rodeiam o veículo no momento em que se dão os fenómenos. Aqui, é crucial o conhecimento que o condutor possui do veículo e a avaliação precisa que faz do comportamento do mesmo perante as atuais condições meteorológicas, como por exemplo a tração perante o estado atual do piso em que se circula. Acontece que em muitas situações as condições meteorológicas afetam de uma maneira tão rápida o pavimento, que impossibilitam o condutor de realizar uma avaliação atempada da condição do piso e de tomar a decisão adequada. Um elevado número de variáveis a ter em conta no momento em que se dá um evento meteorológica, aliado ao fator "medo" que é inato à condição humana, são em boa parte os fatores de risco que provocam este tipo de sinistralidade.

De acordo com os resultados de um estudo relativo ao impacto do comportamento humano na perceção de situações de risco para a condução, resultantes de condições meteorológicas adversas, quando a visibilidade diminui o condutor tem tendência a adaptar a sua condução ao comportamento dos outros veículos à sua volta. Por outro lado, o comportamento do condutor varia de acordo com a conjugação entre o estado do piso e a inclinação do mesmo. O condutor tende a ter mais cuidado no movimento descendente do veículo do que ascendente, devido à probabilidade de despiste ser maior Chen et al. (2019). Contudo, no nosso dia-à-dia como utilizadores das vias públicas podemos constatar a falta de adoção de medidas de condução preventiva por parte de alguns condutores, o que no caso da condução convencional, torna impossível o controlo e previsão do comportamento humano nas mais diversas situações.

2.1.2 *Condições meteorológicas na condução autónoma*

Ao longo dos anos, as marcas automóveis têm produzido e comercializado veículos com um número cada vez maior de sistemas de apoio à condução, como por exemplo o *cruise control*, a travagem automática e a assistência ao estacionamento. Estes sistemas de apoio induzem o desenvolvimento de veículos com capacidade de condução autónoma. No entanto, para que todos os sistemas tenham um comportamento correto e preciso um número elevado de sensores, que permitem ao veículo recolher uma considerável quantidade de dados relativos à sua área circundante, acompanhada do seu estado.

Para o sucesso das decisões, tomadas pelo vasto número de sistemas existentes num veículo, é imperativo que a informação recolhida não esteja corrompida, nem apresente erros. Para além das possíveis falhas que os sensores podem sofrer, um fator determinante para a qualidade da informação obtida é a condição meteorológica no momento da sua

recolha. Em condições atmosféricas favoráveis, com céu limpo e piso seco, todos os sistemas envolvidos na tomada de decisão apresentam bons resultados de operação. Porém as condições atmosféricas não são constantes, bem pelo contrário, são bastante imprevisíveis e de difícil abordagem.

As condições meteorológicas adversas constituem-se como um risco para o sucesso da condução autónoma, visto que afetam gravemente o correto funcionamento dos sensores envolvidos, seja porque se obtém informação de baixa qualidade (como por exemplo, as imagens das câmaras ou a localização do veículo), seja porque ocorrem danos nos sensores. Deste modo, pode afirmar-se que um dos desafios da condução autónoma reside na execução de testes em várias condições meteorológicas consideradas adversas, de maneira a apurar a forma de contornar as limitações provocadas pelo estado do tempo, assim como a transmitir à sociedade um bom nível de confiança nesta nova realidade e a evidenciar as sua vantagens.

2.1.3 *Sensores na condução autónoma*

Um veículo de condução autónoma divide-se em 4 blocos distintos: sensores, perceção, planeamento e controlo. Cada um destes blocos efetua ações específicas: os sensores recolhem dados do meio onde o veículo se encontra no momento, o bloco de perceção transforma esses dados em informação útil para o bloco de planeamento, que por sua vez utiliza essa informação para decidir o comportamento do veículo mediante essas condições, enquanto que o bloco de controlo se assegura de que o veículo respeita esse mesmo comportamento delineado Kocic et al. (2018).

Os sensores são a chave essencial para uma boa tomada de decisão por parte dos veículos de condução autónoma, pois é a partir deles que se inicia todo o processo de transferência de informação até ao último bloco. Os sensores que é comum incluir neste bloco são as câmaras, o GPS, o Light Detection And Ranging (LiDAR) e o Radio Detection And Ranging (RADAR).

A principal função das câmaras é a recolha de informação visual do meio que rodeia o veículo e, também, a monitorização do estado do habitáculo. As câmaras são o tipo de sensor mais comuns nos veículos de condução autónoma, quer porque podem ser utilizadas num vasto leque de aplicações, ou porque existe grande oferta no mercado e a preços acessíveis. No entanto, as câmaras exigem um elevado poder computacional porque recolhem uma grande quantidade de dados, dados estes que requerem uma boa capacidade de processamento de maneira a extrair o máximo de informação útil possível. As câmaras podem ser classificadas mediantes as seguintes características: i) a localização do sensor, podendo neste caso ser câmaras frontais, traseiras ou, em alguns casos, localizadas na lateral do veículo, ii) o espetro de cor e a perceção do espaço, podendo neste caso ser câmaras *mono* ou *estéreo* Taraba et al. (2018).

O GPS, um sensor que é altamente utilizado no quotidiano pela sociedade, fornece em tempo real a localização do veículo. Este tanto pode ser instalado no interior, como no exterior do automóvel.

Para além de detetar a presença de objetos na proximidade do veículo, também é necessário determinar a distância dos mesmos ao veículo. Esta é a função do LiDAR, que utiliza um feixe na gama do infravermelho para determinar a distância entre o sensor e os objetos mais próximos Kocic et al. (2018). O método de funcionamento do LiDAR consiste na emissão de pulsos, que por sua vez serão refletidos pelos objetos que se encontrem no meio circundante ao veículo, o que permite criar uma imagem 3D da ocupação do espaço por parte de objectos ou pessoas Taraba et al. (2018). Na figura 2, encontra-se um exemplo de sensor LiDAR.

Figura 2: Exemplo de sensor LiDAR.

Fonte: www.futurecar.com/235/Velodyne-adds-a-high-res-version-of-its-LiDAR-sensor-for-autonomous-driving

O RADAR, representado na figura 3, desempenha uma importante função na condução autónoma, ao determinar o alcance e velocidade de objetos em movimento Zang et al. (2019). Enquanto outros sensores, como LiDAR, necessitam de duas medidas para calcular a velocidade, tendo por base a diferença entre o instante da emissão de um pulso e o instante da receção do sinal refletido, o RADAR tira proveito do efeito de *Doppler* para medir diretamente a velocidade Kocic et al. (2018).

Figura 3: Exemplo de sensor Radar.

Fonte: phys.org/news/2019-06-radar-sensor-module-added-safety.html

2.1.4 *Condições meteorológicas vs Sensores*

Um dos principais obstáculos no desenvolvimento de veículos aptos para condução autónoma, reside na diminuição do desempenho dos sistemas de apoio ao condutor, perante condições meteorológicas adversas. Tal como já foi referido, o fraco desempenho distorce os dados recolhidos do meio envolvente ao veículo, o que provoca uma má tomada de decisão e, consequentemente acidentes. Para tal, o teste exaustivo destes sensores em condições meteorológicas adversas apresenta um cariz prioritário, pois apenas desta forma é possível encontrar novas metodologias para contornar os efeitos nefastos destas variáveis no processo de tomada de decisão e controlo do veículo.

Relativamente ao sensor LiDAR, os trabalhos apresentados em Peynot et al. (2009) e Heinzler et al. (2019) demonstram que as condições ambientais adversas implicam uma forte redução da perceção da existência de objetos em redor do veículo. Ora, esta redução do desempenho do sensor LiDAR, por exemplo em condições meteorológicas de precipitação, basta que uma gota de água se encontre demasiado próxima do foco emissor do feixe para que provoque uma deteção errada de um obstáculo, pois irá gerar uma reflexão idêntica a um objeto que se encontrasse perto do veículo Zang et al. (2019).

As câmaras fornecem um grande conjunto de dados essenciais à tomada de decisão, tais como a identificação de sinais de trânsito, a condição do pavimento, a deteção de obstáculos, entre outros. No entanto, perante condições adversas, as câmara têm dificuldade em reconhecer padrões e recolher dados fiáveis. Isto deve-se principalmente ao fato de, em casos de precipitação, a presença de um grande número de partículas corromper as imagens e os vídeos que a mesma está a recolher no momento. Para além disso, os fenómenos de condensação também prejudicam a obtenção de dados, pois provocam uma obstrução da visão nas lentes Zang et al. (2019). Em condições favoráveis ao bom desempenho dos sensores, como por exemplo um dia de céu limpo, ter luz solar a incidir diretamente na lente da câmara também corrompe as imagens e vídeos recolhidos, impedindo a obtenção duma boa perceção do espaço Sundararajan and Zohdy (2016).

Como já foi referido, o RADAR apresenta uma vasta utilização nos sistemas de apoio à condução, nomeadamente a deteção de objetos à volta do veículo e a distância e velocidade relativa dos mesmos. Porém, também este sensor quando colocado perante condições meteorológicas adversas, possui uma menor precisão e gera menos dados do que outros sensores Kocic et al. (2018). Uma vez que o RADAR possui uma frequência de aproximadamente *77GHz*, a maior ameaça ao bom desempenho é a precipitação, pois em alguns casos a dimensão das partículas é semelhante ao comprimento de onda utilizado. Resultados obtidos com simulação permitiram concluir que o desempenho do RADAR é degradado em 45% perante condições meteorológicas adversas Zang et al. (2019). O GPS não é tão afetado pelas condições meteorológicas como os outros sensores. Em condições de precipitação, o

GPS sofre uma ligeira degradação de desempenho, pois a chuva afeta a receção de sinal quando o recetor está instalado no exterior do veículo Zang et al. (2019).

2.1.5 *Fusão sensorial e fusão de dados*

Tal como foi referido na secção anterior, as condições meteorológicas adversas degradam gravemente o desempenho dos sensores que se encontram instalados no veículo. No entanto, estas consequências podem ser mitigadas se se complementar a informação recolhida por um sensor, com dados obtidos por outro. Este processo designa-se por fusão sensorial, e consiste em combinar dados de vários sensores distintos, de maneira a transformar o conjunto de dados recolhidos em informação relevante para uma boa tomada de decisão. Com esta abordagem é possível complementar os pontos fracos de um sensor perante condições adversas, com os pontos fortes de um outro, o que contribui para uma solução mais robusta e confiável. Alguns exemplos de fusão sensorial nos veículos com capacidade de condução autónoma são (i) a deteção em 3D de objetos, onde se utiliza o sensor LiDAR para deteção de obstáculos e a câmara para a visualização do ambiente; (ii) o mapeamento de uma grelha de ocupação, utilizada para auxílio ao movimento dos veículos em ambientes dinâmicos, em que são utilizados novamento o LiDAR e a câmara; e (iii) a deteção e rastreamento de objetos em movimento, onde se utilizam três sensores, o RADAR, o LiDAR e a câmara Kocic et al. (2018).

Dada a natureza e função distintas de cada um dos sensores usados na fusão sensorial, estes produzem diferentes tipos de saída e a utilização direta dos dados por eles recolhidos não permite obter informação precisa. Assim sendo, é necessário integrar e normalizar os dados provenientes das diversas fontes envolvidas no processo de fusão sensorial, de maneira a obter uma melhor representação do ambiente em monitorização Panicker et al. (2016). Uma vez obtida uma uma boa combinação dos dados, é possível obter uma boa previsão ou classificação do estado alvo de estudo Dannheim et al. (2014).

2.1.6 *Utilização de variáveis meteorológicas na condução autónoma*

Os veículos com capacidade de condução autónoma permitem diferentes tomadas de decisão, dependendo do intervalo temporal disponível para as tomar. O intervalo temporal pode ir de uma fração de segundo a várias horas. Neste intervalo, o veículo tem de avaliar as implicações das variáveis que afectam a segurança e a mobilidade. Por exemplo, as condições meteorológicas afetam a escolha da velocidade de navegação de modo a evitar os obstáculos, ou a escolha de uma rota em que as condições da via não coloque em causa a segurança do veículo e dos passageiros Sundararajan and Zohdy (2016). A utilização de variáveis meteorológicas ajuda a contornar alguns destes desafios. Estas variáveis dividem-se

em variáveis meteorológicas gerais, como a temperatura do ar e a humidade relativa, e variáveis meteorológicas da via, como a temperatura da superfície da estrada e as condições da mesma.

Recorrendo a resultados obtidos em diversos estudos, a utilização de variáveis meteorológicas revelou-se útil para classificar as condições da via e para medir indiretamente outros fatores que influenciam o estado de circulação na estrada. Num estudo levado a cabo por Lu Junhui and Wang Jianqiang (2010) está demonstrado que é possível a classificação da condição do piso com a utilização de variáveis como a temperatura do ar, a humidade relativa no ar e a temperatura da estrada, esta diretamente relacionada com a radiação solar. Um outro estudo, documentado em Costa et al., faz uso de variáveis meteorológicas como a temperatura do ar e a humidade relativa no ar, para prever a formação de gelo na estrada.

2.2 COMPUTAÇÃO EDGE

A quantidade de dados produzida aumenta constantemente, dado que qualquer interação que tenhamos com um dispositivo "inteligente" cria um grande volume de dados para descrever as nossas ações. Estes dados necessitam de ser transformados em informação útil para que possam ser corretamente utilizados. Em grande parte, o processamento destes dados é efetuado na nuvem *(cloud)*. Apesar de a nuvem fornecer um serviço bastante eficiente, à velocidade com que os dados são produzidos, transportar esses dados para o local de processamento é inviável em muitas situações. Como os dados são produzidos fora de nuvem, uma possibilidade interessante consiste em processá-los onde são gerados. Seguindo esta abordagem, na computação *edge* várias tarefas de computação são executadas fora da rede Shi et al. (2016).

A possibilidade de executar as tarefas de processamento próximo da fonte que gera os dados, permite diminuir a latência e aumentar a capacidade de manutenção dos dispositivos. Estes aspetos são importantes quando as aplicações são sensíveis a atrasos, como por exemplo na condução autónoma, em que a rápida perceção do ambiente e consequentes tomadas de decisão são fulcrais para a integridade do veículo, do condutor e do meio que o rodeia. Para além da diminuição da latência em aplicações sensíveis, a computação *edge* permite que serviços como a nuvem diminuam o seu tempo de resposta aos pedidos efetuados pelos utilizadores, pois não se encontra sobrecarregado com tarefas de computação que são distribuídas pelos dispositivos *edge* Satyanarayanan (2017). Daqui, surgiu a visão da computação *edge*, que consiste na aproximação do utilizador e das suas aplicações aos serviços fornecidos pela nuvem, de maneira a executar tarefas de elevada complexidade em dispositivos ricos em recursos computacionais.

Como representado na figura 4, quase todos os dispositivos com os quais interagimos diariamente produzem uma elevada quantidade de dados. Porém, grande parte desses

dados gerados não são utilizados e, mesmo assim, são tradicionalmente guardados na nuvem, o que provoca uma sobrecarga desnecessária de recursos computacionais e de armazenamento. Com isto, a utilização de dispositivos *edge* para realizar o tratamento destes dados na fonte, ou próximos dela, representa uma mais valia para o bom desempenho dos serviços. A condução autónoma está constantemente a produzir novos dados e a exigir tomadas de decisão. Delegar as tarefas computacionais, associadas à tomada de decisão e baseadas na avaliação de todas as variáveis que rodeiam o veículo, num servidor remoto não é um opção viável nem segura. Deste modo, a utilização de computação *edge*, para auxiliar na tomada de decisão em tempo real, é essencial.

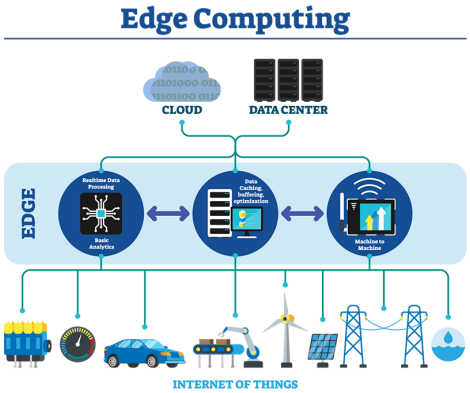


Figura 4: Computação *edge*.

Fonte: innovationatwork.ieee.org/real-life-edge-computing-use-cases/

2.2.1 *Dispositivos edge*

Atualmente, existem no mercado diversos dispositivos *edge*, mas de todos eles há dois que foram especificamente concebidos para aplicações na área da inteligência artificial. Estes dispositivos são o *Google Coral Dev Board* e o *NVIDIA Jetson Nano*. A figura 5 mostra os dois dispositivos.

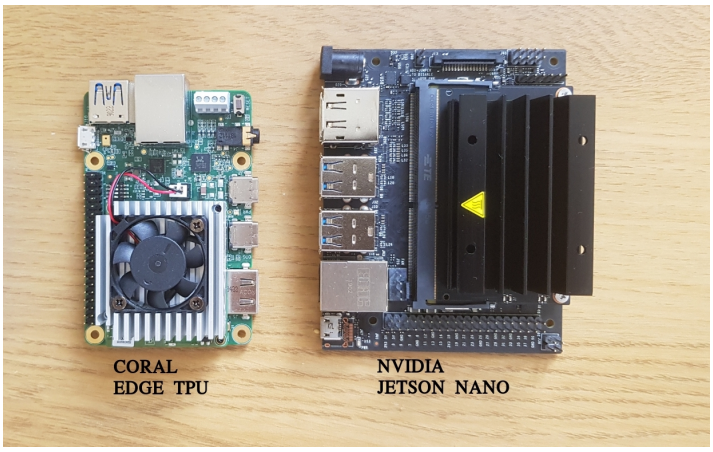


Figura 5: Dispositivos *edge*.

Fonte: www.sooner.ai/2019/05/07/battle-of-edge-ai

Apesar de ambos os dispositivos mencionados permitirem bons resultados, a escolha de um deles para a resolução de um determinado problema deve ser bem ponderada e estudada. Para o problema abordado na presente dissertação, a tabela 1 mostra as especificações a ter em conta em cada um dos dispositivos *edge* considerados.

Tabela 1: Especificações dos dispositivos *edge*.

	Google Coral Dev Board	NVIDIA Jetson Nano
CPU	NXP i.MX 8M SOC (quad Cortex-A53, Cortex-M4F)	Quad-core ARM A57 @ 1.43 GHz
GPU	Integrated GC7000 Lite Graphics	128-core Maxwell
Acelerador ML	Google Edge TPU coprocessor	
RAM	1GB LPDDR4	4GB
Memória Flash	8GB	16GB
WiFi	Sim	Wi-Fi requer um chip externo
Bluetooth	Sim	Não
Dimensões	48mm x 40mm x 5mm	69 mm x 45 mm, 260-pin conector edge

A maioria das avaliações comparativas realizadas após o lançamento destes dispositivos, focam-se em tarefas de classificação e de deteção de objetos. As figuras 6 e 7 mostram, respetivamente, o tempo de inferência por dispositivo e a precisão face ao tempo de inferência. Estes resultados foram obtidos pelo autor Pablo (2019), que utilizou um conjunto de dados constituído aproximadamente por dez mil imagens, divididas em mil categorias. Na realização dos testes foram utilizadas *frameworks* como o `TensorFlow` ou o `PyTorch`, e modelos como `ResNet-50` ou `EfficientNet-S`, para obter resultados mais abrangentes sobre a capacidade dos dispositivos. Esta avaliação colocou à prova mais dispositivos para além

do `Google Coral Dev Board` e o `NVIDIA Jetson Nano`, mas para a presente dissertação estes dois são os de maior interesse.

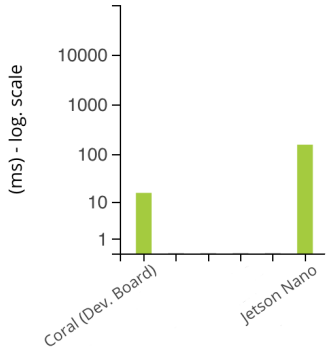


Figura 6: Tempo médio de inferência por dispositivo.

Fonte: tryolabs.com/blog/machine-learning-on-edge-devices-benchmark-report/

De acordo com a figura 6 e para as diversas combinações de *frameworks* utilizadas, o `NVIDIA Jetson Nano` é o dispositivo que apresenta melhor tempo de resposta nas tarefas que lhe foram testadas. Para além de apresentar um melhor tempo de resposta, o *NVIDIA Jetson Nano* possui também uma melhor precisão nos resultados obtidos, como se verifica na figura 7. Porém, os valores obtidos nos testes realizados podem também depender do conjunto de dados utilizado, em conjunto com a *framework* utilizada, que em alguns casos são diferentes de um dispositivo para o outro.

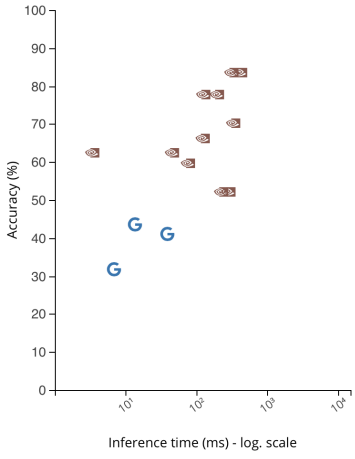


Figura 7: Precisão vs. tempo de inferência.

Fonte: tryolabs.com/blog/machine-learning-on-edge-devices-benchmark-report/

2.3 ALGORITMOS DE CLASSIFICAÇÃO

Ao contrário da regressão, em que se efetua a previsão de um valor contínuo, na classificação é prevista uma classe ou categoria. Um algoritmo de classificação é visto como uma função que atribui pesos às variáveis de entrada, para que no momento em que seja fornecida ao modelo uma nova entrada, este seja capaz de identificar a classe representativa das características dessa entrada. A atribuição de pesos ocorre durante a fase de treino do modelo.

Uma vez que a presente dissertação aborda um problema de classificação da condição do piso, apresentam-se a seguir os algoritmos considerados mais promissores para a sua resolução.

2.3.1 *Regressão Logística*

A regressão logística pode ser dividida em dois tipos, binária e ordinal. A regressão logística binária requer que a variável dependente seja binária, ao contrário da regressão logística ordinal onde a variável dependente deve ser ordinária. A vantagem deste algoritmo de classificação é fornecer informação estatística sobre a importância das variáveis independentes em relação à variável dependente. No entanto, a quantidade de pressupostos definidos por este tipo de classificação representa uma desvantagem. Na regressão logística, as variáveis independentes não devem apresentar multicolinariedade entre si, ou seja não devem ser altamente correlacionadas. As variáveis independentes não devem resultar de medições repetidas ou criadas a partir de dados combinados, devem ser independentes umas das outras. Estes pressupostos são justificados pelo fato de a regressão logística ser um modelo linear.

2.3.2 *K-Nearest Neighbours*

K-Nearest Neighbours (K-NN) é um modelo de aprendizagem supervisionada, não linear, que pode ser utilizado tanto em problemas de classificação, como em problemas de regressão. Este modelo depende do pressuposto de todos os pontos semelhantes se encontrarem próximos uns dos outros. Com isto, o K-NN calcula a similaridade ou proximidade dos pontos, com recurso a métricas tais como a distância de *Minkowski*, a distância de *Manhattan*, a distância euclidiana ou a distância termo cosseno. Apesar de ser um algoritmo fácil de compreender e implementar, o seu desempenho depende do volume de dados que lhe é aplicado. Com o aumento da quantidade de dados, o tempo de resposta do algoritmo será cada vez maior. Para além disto, também é necessário escolher o valor de k, ou seja o número de vizinhos, que melhor se adequa ao problema em questão.

2.3.3 *Máquinas de Vetores de Suporte*

As Máquinas de Vetor de Suporte (SVM) são um modelo de classificação linear, capaz de abordar problemas de classificação, regressão e deteção de valores discrepantes. Porém, este algoritmo pode ser convertido num modelo não linear, com a aplicação de um *kernel*. O principal objetivo das SVM é encontrar um hiperplano, num espaço de dimensão N, em que N é o número de atributos de entrada, que seja capaz de classificar corretamente os pontos do conjunto de dados fornecido. O hiperplano funciona como o limite de decisão para a classificação dos dados, em que a classe associada a cada um dos pontos dependerá da posição que ele ocupe em relação ao hiperplano definido. A orientação do hiperplano será alterada de acordo com os pontos que se encontram mais próximos dele. Estes pontos mais próximos constituem os vetores de suporte, e a partir deles é possível definir as margens que orientam o hiperplano calculado. O algoritmo SVM é adequado para resolver problemas de classificação onde o conjunto de dados utilizado não seja de elevada complexidade, ou seja, onde o número de atributos de entrada não seja muito elevado. Uma vantagem do modelo é ser imune à presença de valores discrepantes, uma vez que os pontos de maior interesse são os que se encontram mais próximos do hiperplano, e que constituem os vetores de suporte.

Na figura 8 é possível observar a margem definida pelos pontos que constituem os vetores de suporte e a maneira como estes influenciam a orientação do hiperplano em que, x é a entrada do modelo, w^T representa os pesos do modelo e b o *bias*.

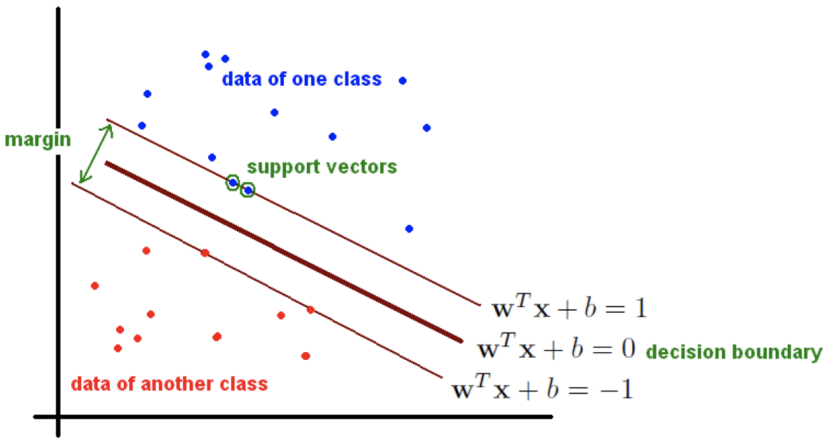


Figura 8: Máquinas de Vetores de Suporte.

Fonte: fderyckel.github.io/machinelearningwithr/svm.html

2.3.4 *Árvores de Decisão*

As árvores de decisão são conhecidas pela sua versatilidade em lidar tanto com problemas de regressão, como de classificação. Para além desta flexibilidade, possuem um elevado grau de interpretabilidade, têm a capacidade de abordar problemas lineares e não lineares, e não exigem que se escalem os atributos de entrada. No entanto, é um algoritmo bastante sensível à ocorrência de *overfitting* e obtém resultados pouco precisos em conjuntos de dados pequenos.

A construção de uma árvore de decisão baseia-se na redução do desvio padrão das classificações, ou seja, a construção de novos ramos na árvore tem como objetivo encontrar os atributos que promovem a redução do desvio padrão no novo nível da árvore. Assim, é possível criar novos ramos mais homogéneos. Esta redução do desvio padrão pode também ser interpretada como um ganho de informação, que é calculado a partir do grau de diminuição do desvio padrão, depois de efetuada a ramificação. Logo, quanto maior a redução do desvio padrão, maior o ganho de informação.

2.3.5 *Bagging*

O método de *bagging* consiste na implementação de diversos algoritmos de aprendizagem automática do mesmo tipo, por exemplo árvores de decisão, que por sua vez constituirão uma *random forest*. Num problema de regressão, a decisão final baseia-se na média dos resultados dos vários algoritmos e, num problema de classificação, é selecionada a classe mais votada (*hard voting*). Todos os modelos utilizados para agregação são treinados num conjunto de dados diferente. Esta diversidade advém da aplicação de *bootstraping* no conjunto de dados original, em que deste são criadas réplicas para o treino individual de cada um dos modelos. O objetivo desta abordagem é a criação de instabilidade no método de classificação ou regressão, o que melhora substancialmente os resultados das previsões (Breiman, 1996). A figura 9 ilustra o método de *bagging*, em que a partir de um conjunto de dados inicial são geradas réplicas, que por sua vez cada uma delas será utilizada para treinar um algoritmo de aprendizagem automática. No final, agregam-se os resultados dos modelos e decide-se o resultado.

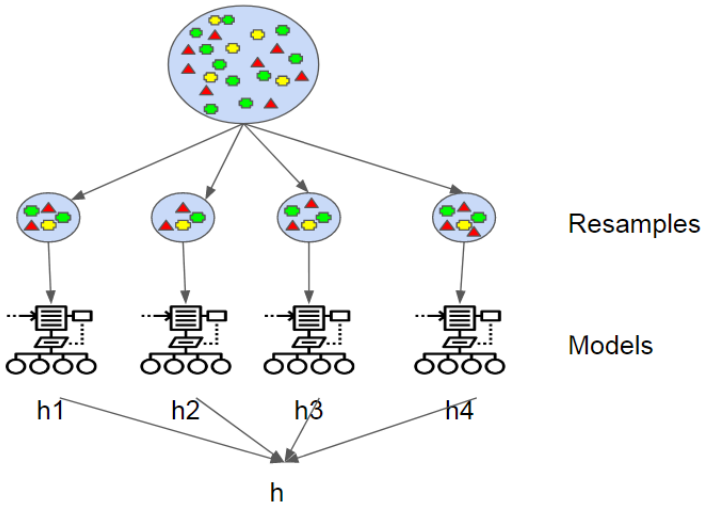


Figura 9: Método de *bagging*

Fonte: http://www.differencebetween.net/technology/difference-between-bagging-and-random-forest/

2.3.6 *Random Forests*

O algoritmo *Random Forest* utiliza um conjunto de árvores de decisão, comummente treinado com o método de *bagging*. Este classificador possui todos os hiperparâmetros presentes nas árvores de decisão, para controlar a forma como as árvores são construídas, em conjunto com os hiperparâmetros presentes no método de *bagging*, para controlar o agrupamento das árvores de decisão. Uma *random forest* fornece uma maior aleatoriedade na construção das árvores de decisão que a constituem. Uma vez que o objetivo da construção das árvores de decisão é encontrar o melhor atributo para efetuar a divisão em cada nó, numa *random forest* procura-se o melhor sub-conjunto de atributos aleatórios.

Apesar deste algoritmo ser bastante robusto, preciso e com bom desempenho, tanto em problemas lineares como não lineares, sem uma escolha correta dos hiperparâmetros, como por exemplo o número de árvores de decisão, é fácil ocorrer *overfitting*. Para além da elevada sensibilidade ao *overfitting*, o algoritmo tem uma interpretabilidade bastante reduzida.

2.3.7 *Boosting*

Os algoritmos baseados em *boosting*, são algoritmos genéricos e não um modelo específico de aprendizagem automática. Estes algoritmos procuram melhorar a classificação das previsões ao treinar uma sequência de modelos, por sua vez mais fracos, mas que a cada modelo treinado, este irá compensar os erros cometidos pelos modelos já treinados Geron (2017).

Nesta categoria de algoritmos baseados em *boosting*, existem dois que se destacam, *Adaboost* e *Gradient Boosting*.

O algoritmo *Adaboost* foi desenvolvido para abordar problemas de classifição, em que se foca principalmente nas instâncias que o seu antecessor não se ajustou tanto, ou seja, as fraquezas dos antecessores são identificadas pelas suas taxas de erro. Aqui, em cada iteração, este algoritmo identifica as instâncias mal classificadas e aumenta os pesos das mesmas e, por consequência diminui os pesos das que foram classificadas corretamente, de modo a que o classificador seguinte tome mais atenção em classificá-las corretamente Tharwat (2018).

Enquanto que o *Adaboost* ajusta em cada iteração os pesos associados a cada instância do conjunto de dados, o *Gradient Boosting* foca-se na diferença entre a previsão e o valor real. Com isto, este algoritmo tenta ajustar os novos modelos treinados aos erros residuais cometidos pelos seus antecessores.

2.3.8 *Stacking Classifier*

Enquanto que a maioria dos *ensemble methods* se baseiam em *hard voting* ou seja, agregar todas as previsões e no final selecionar a classe que obteve mais votos, o *stacking classifier* diferencia-se ao treinar um modelo para efetuar esta agregação. Ora, uma vez que este é um *ensemble method*, é constituído por dois ou mais modelos de aprendizagem automática. No entanto, este classificador em específico não requer que estes modelos base sejam homogéneos, por exemplo apenas árvores de decisão, mas permite que sejam usados modelos distintos uns dos outros. Aliás, esta diversidade é imperativa, pois como o classificador final tentará compensar os erros de uns modelos com as classificações corretas de outros, os modelos base não devem cometer os mesmos erros. Com isto, é possível que este classificador obtenha uma melhor classificação do que o melhor modelo base utilizado, pois irá compensar os seus erros com os sucessos de todos os outros classificadores. Na figura 10 encontra-se ilustrado um diagrama do classificador em questão, em que a verde estão representados os classificadores base, que por sua vez efetuam previsões, e que no final serão agregadas por um único classificador, também chamado de *meta*, que tomará a última decisão.

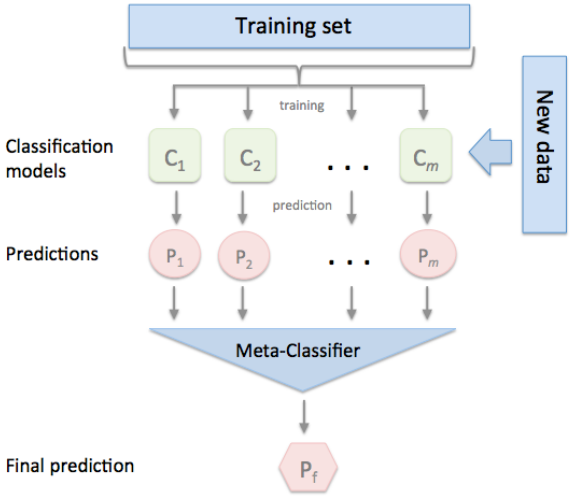


Figura 10: Método de *stacking*

Fonte: http://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/

2.3.9 *Pairwise Classification*

Este método de classificação, para além de se denominar *pairwise classification*, também é apresentado como *round robin classifier*, *class binarization* ou n^2 *classifier*. Esta abordagem especializa-se em problemas de classificação com um número de classes superior a dois e tem como principal objetivo a decomposição de um problema multi classe, num conjunto de sub problemas com apenas duas classes, de modo a colmatar os limites de decisão não lineares. Um problema inicial, por exemplo com quatro classes, será decomposto em $(n^2 - n)/2$ sub problemas ou seja, seis sub problemas. Para uma instância nunca antes vista, a classificação final resulta de uma agregação dos resultados obtidos por todos os classificadores base. No entanto, esta classificação pode ser simples, o que acaba por ser semelhante ao conceito de *hard voting* em que a classe mais votada é a escolhida, ou uma classificação baseada nos pesos de cada uma das previsões, onde a credibilidade dos classificadores é determinante para a escolha final. Esta credibilidade é definida ainda durante a fase de treino dos classificadores base Jelonek and Stefanowski (1998).

2.3.10 *Redes Neuronais Artificiais*

As Redes Neuronais Artificiais (ANN) constituem-se como um modelo extremamente versátil, escalável e com ótimo desempenho na realização de diversas e complexas tarefas de aprendizagem automática, como por exemplo a classificação de milhões de imagens ou de séries temporais.

Uma ANN utiliza diferentes camadas de processamento de modelos matemáticos, de modo a retirar significado da informação utilizada durante o treino. O número de camadas utilizadas nas redes ANN depende muito da complexidade do problema a resolver e o valor pode ir até às centenas e incluir dezenas de milhões de nós (ou neurónios ou unidades). As ANN são constituídas por uma camada de entrada, uma ou mais camadas escondidas e uma camada de saída. A camada de entrada recebe dados nas mais variadas formas, como por exemplo texto, imagens, ou áudio. A partir daqui, os dados seguem para os nós presentes nas camadas escondidas, onde os dados introduzidos na rede são convertidos em informação útil para a aprendizagem. Nas camadas *fully connected* estes nós encontram-se todos ligados entre si e a cada uma destas ligações está associado um peso (w). O cálculo dos pesos ótimos das ligações (w_{ij}) é realizado com base num algoritmo de otimização, como o gradiente descendente, em que em cada iteração de treino, os pesos associados às ligações são atualizados no sentido de minimizar a função de perda (*loss function*), que mede o erro entre as previsões e as classes alvo. A otimização pára ao fim de um número de iterações pré-definido ou quando a perda for inferior ao valor fixado.

A fase de treino de uma ANN itera sobre a totalidade do conjunto de dados um número de vezes que se designa por número de épocas. Normalmente, o treino de uma ANN termina quando se executar este número de épocas. No entanto, existe a possibilidade de aplicar *early stopping*, que consiste em verificar se a perda durante o treino da rede começa a ser demasiado pequena. Neste caso, o treino termina antes de se atingir o número máximo de épocas definidas para o treino.

2.4 TRABALHOS RELACIONADOS

Até à data foram publicadas diversas abordagens ao problema da classificação da condição do piso, na sua maioria recorrendo a imagens e vídeos provenientes de câmaras instaladas no veículo. No entanto, já existem diversos estudos que cobrem as vantagens de utilização de outros sensores e a fusão dos mesmos, bem como novos métodos e sistemas de classificação. O trabalho documentado em (Yamada, 2003) apresenta uma técnica de classificação da condição do piso, que utiliza dados recolhidos por uma câmara instalada no veículo. A técnica baseia-se no princípio da polarização da luz refletida na superfície a avaliar, que é diferente consoante esteja presente água ou gelo nessa superfície. Contudo, torna-se complicado aplicar esta técnica durante a noite. Dada esta limitação, a introdução de informação complementar à obtida com a câmara, e independente da presença de iluminação, é uma mais valia.

Num estudo levado a cabo pelos autores Casselgren et al. (2007), a abordagem utilizada para classificar a condição do piso consiste em explorar as variações de luz refletida pela superfície, mediante diversas condições, como sejam asfalto seco, molhado, com gelo e neve. Utilizam um espetrómetro que efetua a medição da luz refletida pela superfície a classificar, com o qual foram realizadas experiências utilizando primeiramente dois comprimentos de onda e, dadas as limitações na classificação, foi acrescentado um terceiro comprimento de onda. Uma vez que se revelou difícil diferenciar a superfície com gelo da superfície com água, utilizando fusão dos dados do espetrómetro com a temperatura da estrada, foi possível diminuir o número de classificações erradas. Num estudo posterior por parte de Casselgren et al. (2016), os autores investigam a utilização de uma câmara que atua no infravermelho próximo, projetando na superfície do piso três comprimentos de onda. De modo a complementar a classificação, todas as superfícies testadas possuem na sua proximidade uma fonte de iluminação auxiliar, de maneira a simular um ambiente de utilização real. Uma outra abordagem, documentada em (Lu Junhui and Wang Jianqiang, 2010), baseia-se na classificação do piso na relação entre a temperatura da superfície da estrada e a radiação solar, aplicando uma rede neuronal (*backpropagation neural network*). A utilização de mais variáveis meteorológicas tornou-se uma mais valia, pois em condições noturnas a temperatura da superfície da estrada e do ar são semelhantes.

Na tentativa de encontrar uma relação entre a condição do piso, as imagens obtidas por uma câmara instalada num veículo e as variáveis meteorológicas, Jonsson (2011a) efetua fusão destes mesmos dados. Demonstram assim que a utilização das variáveis meteorológicas melhora o método de classificação, em contraste com a utilização singular das imagens da câmara. Os mesmos autores desenvolveram também um sensor, constituído por três detetores de infravermelhos, para ser instalado à face da estrada Jonsson (2011b). Desta forma e, aproveitando o facto de a água absorver a radiação de maneira diferente consoante

o estado em que se encontra, foi possível classificar a condição do piso num segmento da estrada.

Numa outra abordagem, os autores Yang et al. (2013) implementaram um algoritmo que utiliza máquinas de vetores de suporte e *Wavelet Packet Transform* para extrair as zonas de maior frequência das imagens utilizadas, sendo estas diferentes consoante as características da condição do piso. Para detetar a presença de água no piso, os autores Silion and Fosalau (2014) instalaram dois sensores para medir a temperatura e a humidade, um colocado no topo do carro e outro próximo da superfície do piso. Com isto, os autores pretendiam determinar a diferença de humidade em dois pontos distintos, pois como demonstram no estudo, nos casos em que a via está húmida, a humidade relativa no topo do carro é inferior à registada próximo da superfície da estrada.

No artigo publicado por Dannheim et al. (2014), os autores propõem um método onde é efetuada a fusão das imagens provenientes das câmaras instaladas no veículo com dados recolhidos do sensor *LiDAR*. O objetivo consistia em detetar o estado do tempo e, posteriormente, classificar a condição do piso, mediante a informação recolhida. A utilização do *radar* para classificar a condição do piso é sugerida por Viikari et al. (2008), onde se avalia a influência das propriedades de retro-difusão da água líquida e do gelo, na identificação da presença da mesma na superfície da via. No entanto, as variáveis meteorológicas condicionam a aplicação desta metodologia, daí que a fusão destas variáveis com os dados obtidos do radar poderá melhorar os resultados da classificação.

Os autores Jonsson et al. (2015) sugerem que a classificação da condição do piso deve ser realizada num segmento de estrada e não mediante a medição num único ponto. No final, o método implementado consistiu em utilizar uma câmara instalada à face da estrada, a operar no infravermelho próximo, e equipada com vários filtros óticos, para avaliar os comprimentos de onda mais adequados a uma correta classificação do piso. Com o objetivo de generalizar ao máximo a obtenção de informação ótica, útil para a classificação da condição do piso, no trabalho descrito em (Qian et al., 2016) as imagens obtidas do ambiente são fornecidas por uma câmara não calibrada instalada no interior do veículo, em diversas condições meteorológicas e em diversos tipos de piso. No entanto, este método é afetado pelas limitações provocadas pelas variáveis meteorológicas.

Na metodologia de classificação da condição do piso descrita por Bystrov et al. (2016), avalia-se a viabilidade de utilizar ultrassom em conjunto com a sua reflexão. Deste estudo é possível concluir que variáveis como a temperatura e a humidade relativa têm impacto na devolução do sinal, pelo que a fusão das condições meteorológicas torna-se essencial para obtenção de melhores resultados. Os mesmos autores procederam à avaliação de diversas métodos de classificação supervisionada, baseado na fusão de dados do sonar e de um radar polarimétrico Bystrov et al. (2015). Posteriormente, é aplicada uma rede neuronal modular para classificar a condição do piso Bystrov et al. (2017). Para além das vantagens associadas

ao paralelismo e ao rápido treino do modelo, a fusão destes dois tipos de dados traduz-se em bons resultados de classificação. Com o objetivo de obter uma clara compreensão do desempenho dos vários sensores presentes no veículo, quando aplicados na classificação do piso, os autores avaliaram o desempenho individual de cada sensor em condições de teste reais Bystrov et al. (2018). Deste trabalho, foi possível concluir que a utilização individual destes sensores na determinação da condição do piso é insuficiente para a obtenção de bons resultados. A fusão de variáveis meteorológicas é que permite atenuar as limitações apresentadas por cada sensor isoladamente.

A metodologia descrita por Galanis et al. (2018) utiliza variáveis meteorológicas, como a temperatura, a humidade e a sensação térmica, para classificar a condição do piso e, também para estimar o *Road Surface Index (RSI)*. O RSI quantifica a influência das condições meteorológicas no estado da superfície da estrada. Os autores de (Gui et al., 2019) optaram por construir um dispositivo, que incluía um sensor piezoelétrico e um sensor ótico, para classificar a condição do piso e para inferir a espessura de água ou gelo na estrada. A integração de mais sensores no dispositivo, como por exemplo um termómetro, poderia tornar a classificação mais precisa e confiável.

3

METODOLOGIA

No presente capítulo, é descrito todo o processo de desenvolvimento desta dissertação, desde a fase de recolha de dados necessários à resolução do problema proposto, até à aplicação de modelos de aprendizagem automática aos mesmos dados. Aqui, são também enunciadas todas as tecnologias e bibliotecas utilizadas em cada fase deste processo.

3.1 TECNOLOGIAS UTILIZADAS

Ao longo do desenvolvimento da presente dissertação e à medida que os desafios se revelavam, a necessidade de adotar novas estratégias e a rápida adaptação aos mesmos levou à aprendizagem e utilização de novas tecnologias que ajudariam a abordar estes problemas. Nesta secção são apresentadas as tecnologias utilizadas neste trabalho. No entanto, não estão descritas aqui bibliotecas tais como `Scikit Learn` e `Numpy`, uma vez que são vastamente utilizadas, bem como o dispositivo *edge* `Nvidia Jetson Nano`, pois já se encontra descrito na secção 2.2.

3.1.1 *Scrapy*

`Scrapy` é uma *framework* desenvolvida em `Python`, que permite efetuar *web scraping* e *web crawling* a páginas *web*, de modo a ser possível extrair informação dessa mesma página, ou para monitorizar cenários de teste automáticos Scrapy (2020). A utilização do `Scrapy` roda em torno da utilização de *Spiders*, que não são mais do que classes autónomas, que após a definição de instruções por parte do utilizador e execução do *script*, a classe assume o controlo das mesmas e inicia o *crawling* da página *web*.

3.1.2 *Selenium*

O `Selenium` é um conjunto de ferramentas e bibliotecas que permitem a automação de testes no navegador *web* Selenium (2020). Apesar do seu principal objetivo não ser o *scraping*

de páginas *web*, este processo é facilitado, uma vez que replica todas as ações que um utilizador humano desencadeia ao interagir com um *website*, tais como cliques ou *screenshots*. O `Selenium` em conjunto com o *Scrapy*, permite que seja feito o *crawling* a páginas *web* dinâmicas, pois podemos comunicar ao *crawler* a instrução de aguardar que o conteúdo de um certo elemento esteja presente. Contudo, com todas estas ações, o `Selenium` demora mais tempo a executar do que o `Scrapy`.

3.1.3 *Jupyter Notebook*

O `Jupyter Notebook` é uma aplicação *web*, o qual permite a criação de documentos estruturados em formato *JSON*, onde é possível desenvolver código numa célula e ao mesmo tempo acompanhar os *outputs* dessa mesma célula. A grande interação entre *inputs* e *outputs* e a capacidade de em determinado momento poder executar uma porção de código, por exemplo apenas uma célula, torna-se bastante proveitoso no caso de grandes linhas de raciocínio.

3.1.4 *Colaboratory*

`Colaboratory`, ou simplesmente `Colab`, é um produto `Google` que fornece um ambiente `Jupyter Notebook` para desenvolver código tal como num `Jupyter Notebook` comum. No entanto, ao passo que numa instância *Jupyter Notebook* comum, esta é executada utilizando os recursos da máquina remota, o `Colab` disponibiliza gratuitamente o acesso a *hardware* como *CPU's* e *GPU's*. Desta maneira, a execução do código desenvolvido é acelerada, o que representa uma vantagem para as tarefas mais morosas do ponto de vista cronológico e tecnológico.

3.1.5 *TensorFlow*

O `TensorFlow` é uma biblioteca *open-source* desenvolvida pela *Google Brain Team*, dedicada ao desenvolvimento de modelos de aprendizagem automática TensorFlow (2020). Esta biblioteca pode ser executada em diversos *CPU's* e *GPU's*, bem como em diversos sistemas operativos e dispositivos móveis. Para além disso, possui também uma unidade de processamento dedicada a esta tecnologia, o Tensor Processing Unit (TPU). O `TensorFlow` é também reconhecido pela sua flexibilidade no *deployment* nos mais diversos dispositivos e arquiteturas, desde *cluster*, até dispositivos móveis e dispositivos *edge*, como é o caso da presente dissertação.

3.2 RECOLHA DE DADOS

Para a realização desta dissertação são necessários dados relativos às condições meteorológicas, devidamente acompanhados da respetiva condição do piso e rácios dos comprimentos de onda em determinado momento. No entanto, nesta fase do projeto *Sensible Car*, os dados disponibilizados pelos investigadores para a presente dissertação, as amostras constituintes deste conjunto de dados, não estão associadas às respetivas variáveis meteorológicas no momento da medição. Deste modo, a abordagem ao problema da classificação do piso com a utilização de fusão sensorial entre variáveis meteorológicas e comprimentos de onda teve que ser reformulada. Para colmatar a ausência dos dados meteorológicos foi necessário encontrar conjuntos de dados, que contivessem não só as variáveis meteorológicas, mas também a condição do piso. Apesar de existirem diversos conjuntos de dados relativos a condições meteorológicas, nenhum dos que foram encontrados incluiam a condição do piso.

3.2.1 *Dados meteorológicos*

Com uma pesquisa exaustiva, foi possível encontrar um conjunto de dados relativo a acidentes rodoviários ocorridos no Reino Unido, em que uma das variáveis presentes é a condição do piso no momento em que ocorreu o acidente Fisher-Hickey (2017). Este conjunto de dados é constituído por aproximadamente 1.6 milhões de acidentes ocorridos durante os anos de 2000 a 2016. Todos os eventos são relativos aos países que formam o Reino Unido, Inglaterra, Escócia, Irlanda do Norte e País de Gales. Os registos neste conjunto de dados possuem a condição do piso, mas não incluem as condições meteorológicas. No entanto, estão presentes outras variáveis que ajudam a obter os dados em falta. Cada acidente tem associadas as coordenadas geográficas, a data, a hora, a descrição por extenso do estado do tempo, a condição do piso e a existência ou não de condições extraordinárias no local, como por exemplo a presença de óleo na via. Esta última variável tem como principal objetivo a deteção de valores discrepantes, uma vez que o problema a tratar aborda a classificação da condição do piso em consequência das condições meteorológicas e não devido a fatores adicionais não contemplados. Com as variáveis selecionadas, é retirado do conjunto de dados original um conjunto de amostras de aproximadamente 25000 acidentes, numa proporção igual à do conjunto de dados completo. Por exemplo, se no conjunto de dados completo os acidentes ocorridos em condições em que o piso se encontrava molhado representam 45% dos dados, o conjunto de acidentes recolhidos em piso molhado representarão também 45% da totalidade dos dados. Para tal, são importados os dados do ficheiro que contém os acidentes mais recentes, de modo a aumentar a probabilidade de naquelas zonas já existirem sensores meteorológicos para recolha de dados. Uma vez importado o ficheiro, é utilizado o método `StratifiedShuffleSplit` da biblioteca `scikit-learn`, que permite obter

um conjunto de amostras aleatórias e estratificadas, em que é preservada a percentagem de amostras que possui um determinado atributo no conjunto de dados original, neste caso a condição do piso.

Em 3.1 encontra-se o excerto de código utilizado para a extrair do conjunto de dados original um conjunto de amostras. Uma vez que o ficheiro importado possui aproximadamente 470000 registos de acidentes, o que requer um elevado poder computacional para os processar, optou-se por retirar uma pequena percentagem dos dados. Esta percentagem é definida pelo parâmetro `test_size`, o qual é inicializado com o valor 0.05, ou seja 5% do conjunto de dados original, que equivalem a 24000 acidentes. Ora, este conjunto de amostras é estratificado utilizando a variável condição do piso, `Road Surface Conditions`. Neste caso, o conceito de estratificação tem como objetivo obter no conjunto de amostras recolhidas uma percentagem de classes igual ao conjunto de dados original. Como já foi referido, se o conjunto de dados original possuir 45% de classes do tipo `Dry`, no conjunto de amostras retiradas a percentagem de classes do tipo `Dry`, deverá também ser 45%.

```
1 from sklearn.model_selection import StratifiedShuffleSplit
2
3 split = StratifiedShuffleSplit(n_splits=1, test_size=0.05, random_state=42)
4 for train_index, test_index in split.split(all_incidents, all_incidents["Road_Surface_Conditions
      "]):
5     strat_train_set = all_incidents.iloc[train_index]
6     uk_incidents = all_incidents.iloc[test_index]
```

Listagem 3.1: Obtenção do conjunto de amostras com `StratifiedShuffleSplit`.

Como se pode verificar na tabela 2, apenas 8 dos 33 atributos disponíveis no conjunto de dados original foram selecionados, visto que os restantes servem apenas para caracterizar o acidente.

Tabela 2: Extrato do conjunto de amostras recolhidas.

Accident Index	201006G028673	2.01107E+12	200904AN09047	200991NJ03785	2.00953E+12
Longitude	-2.115175	-2.772961	-3.017468	-4.972872	-2.225434
Latitude	53.456099	53.371135	53.822340	58.088384	51.850615
Date	25/02/2010	17/06/2011	03/06/2009	21/03/2009	04/09/2009
Time	15:00	18:26	18:14	16:50	07:25
Light Conditions	Daylight: Street light present	Daylight: Street light present	Daylight: Street light present	Daylight: Street light present	Daylight: Street light present
Weather Conditions	Unknown	Fine without high winds	Fine without high winds	Fine without high winds	Fine without high winds
Road Surface Conditions	Dry	Dry	Dry	Dry	Dry
Special Conditions at Site	None	None	None	None	None

Na tabela 3, encontram-se descritas as variáveis que compõem o conjunto de amostras retiradas. As variáveis `Latitude` e `Longitude` correspondem à localização geográfica do acidente, `Date` e `Time` referem-se ao dia e hora do mesmo, `Light_Conditions` e `Weather_Conditions` consistem nas condições ambientais no momento do evento, `Special_Conditions_at_Site`

sinalizam efeitos extra que poderão ter influenciado o sinistro e `Road_Surface_Conditions` corresponde à condição da superfície do piso.

Tabela 3: Descrição das variáveis dos registos selecionados.

Variável	Descrição
Latitude	Latitude a que ocorreu o acidente
Longitude	Longitude a que ocorreu o acidente
Date	Dia do acidente
Time	Hora do acidente
Light_Conditions	Condições de luminosidade no momento
Weather_Conditions	Condições atmosféricas no momento
Road_Surface_Conditions	Condição do piso no momento
Special_Conditions_at_Site	Situações adicionais relativas ao acidente

Numa primeira fase, optou-se pela remoção dos valores discrepantes presentes no conjuntos de amostras. Neste caso, entende-se por valor discrepante todo o registo de incidente em que as condições atmosféricas se encontrem nulas ou com o valor `Other` e em que existam condições extraordinárias ao acidente ou seja, qualquer registo com um valor diferente de `None`. Após remover todos os valores discrepantes, é possível proceder à obtenção da localização mais detalhada, a partir do uso das coordenadas geográficas obtidas na recolha do conjunto de dados. Para isso foi utilizado o serviço web `Geopy` que, dadas as coordenadas geográficas dum local, devolve uma descrição detalhada do local em questão. De seguida, os dados relativos à latitude e longitude são agrupados, para posteriormente serem utilizados como parâmetro na chamada da `API` utilizada.

Em 3.2 apresenta-se o excerto de código utilizado para obter a informação da zona onde ocorreu um acidente. Para o efeito definiu-se a função `getRoadWithCoordinates`, que recebe como parâmetros as coordenadas geográficas de um determinado acidente. Nesta função é realizada uma chamada ao método `reverse` do `Geopy`, que para cada par de coordenadas (latitude e longitude), devolve o endereço das mesmas, neste caso a cidade. Com isto, é definido o método `geocode`, ao qual se aplica a função `getRoadWithCoordinates` e um atraso de 1 segundo, uma vez que este serviço apresenta um número máximo de pedidos por período de tempo. Por fim, é anexada uma nova coluna ao conjunto das amostras original, onde constará para cada incidente a cidade onde ocorreu.

```
1 def getRoadWithCoordinates(coordinates):
2     try:
3         location = geolocator.reverse(coordinates)
4         return location.raw['address']['city']
5     except KeyError:
6         return 'None'
7 
8 from geopy.extra.rate_limiter import RateLimiter
9 geocode = RateLimiter(getRoadWithCoordinates, min_delay_seconds=1)
```

```
10 uk_incidents['location'] = uk_incidents['coordinates'].apply(geocode)
```

Listagem 3.2: Obtenção da localização.

Com o conjunto de dados atualizado com o local onde ocorreu cada acidente, é possível pesquisar os dados meteorológicos associados a cada acidente. Após uma pesquisa sobre possíveis aplicações e sítios Web para extrair estes dados, optou-se pelo sítio Web *Weather Underground*, que contém dados históricos de diversas cidades, obtidos a partir de estações meteorológicas públicas e privadas. Ao introduzir a data e o local pretendido para pesquisa neste sítio Web, a interface disponibiliza ao utilizador uma tabela de dados meteorológicos atualizados com a periodicidade de 30 minutos. Na tabela 4 encontra-se um exemplo da representação dos dados para uma determinada pesquisa.

Tabela 4: Dados meteorológicos obtidos de Weather Underground.

Time	Temperature	Dew Point	Humidity	Wind	Wind Speed	Wind Gust	Pressure	Precip.	Condition
1:20 AM	54 F	48 F	82 %	SW	15 mph	0 mph	29.57 in	0.0 in	Partly Cloudy
1:50 AM	54 F	48 F	82 %	WSW	12 mph	0 mph	29.57 in	0.0 in	Mostly Cloudy
2:20 AM	54 F	48 F	82 %	SW	10 mph	0 mph	29.57 in	0.0 in	Partly Cloudy
2:50 AM	54 F	48 F	82 %	WSW	13 mph	0 mph	29.57 in	0.0 in	Partly Cloudy

Aqui, os dados de interesse são a temperatura, o ponto de orvalho (*Dew Point*), a pressão atmosférica e a humidade. Constata-se que tratar manualmente os 24000 acidentes do conjunto de dados, e registar as condições meteorológicas associadas a cada um, não é viável. Como a estrutura do `URL` de consulta dos dados históricos não se altera, apenas mudam os campos data e cidade do mesmo, é possível aplicar *web scrapping* para extrair os dados meteorológicos. Antes de se conseguir gerar dinamicamente o `URL` correspondente a cada incidente, é necessário adaptar os dados que farão parte dos campos dinâmicos: a localização e a data. No caso da localização, o nome da cidade não pode estar capitalizado, por isso aplicou-se a função `lower` para colocar todas as instâncias em minúsculas. No campo data é necessário alterar o formato da mesma, para obedecer à configuração "ano-mês-dia", em vez de "dia/mês/ano" (excerto 3.3).

```
1 uk_incidents['Date'] = uk_incidents['Date'].apply(lambda x: datetime.strptime(x, '%d/%m/%Y').
      strftime("%Y-%m-%d"))
```

Listagem 3.3: Modificação do campo data.

Uma vez cumpridos todos os requisitos, é definida uma função que gera dinamicamente, para cada incidente, o respetivo `URL` para consultar os dados meteorológicos. No excerto de código 3.4, encontra-se a função criada para gerar os `URL`.

```
1 def createUrl(date, location):
2     url = 'https://www.wunderground.com/history/daily/gb/' + location +'/date/' + date
3     return url
4
```

```
uk_incidents['URL'] = uk_incidents.apply(lambda row: createUrl(row['Date'], row['location']),
    axis=1)
```

Listagem 3.4: Geração dinâmica de URL.

Uma vez gerado o URL para cada acidente do conjunto de dados, procedeu-se à instalação da *framework* `Scrapy` e é implementado o `Spider` que irá percorrer todas as páginas Web associadas aos *links* criados.

Inicialmente, a abordagem de *scraping* utilizada consistiu apenas na utilização de um `Spider` para recolher os dados. No entanto, após uma análise e execução de um `Spider` simples para obter a tabela pretendida de uma das páginas alvo, o resultado devolvido era sempre nulo. Com uma inspeção mais rigorosa, reparou-se que a mesma está implementada com recurso à *framework Angular* ou seja, estas páginas são Single Page Application (SPA). Com isto, a utilização com `Spider` não é viável, uma vez que apenas permite o *scraping* a páginas *HTML* não dinâmicas. Contudo, é possível associar o `Spider` com `Selenium`, uma ferramenta que simula um *browser*. Como o `Selenium` efetua a simulação do *browser*, permite que o `Spider` efetue o *crawl* e recolha a informação pretendida dos respetivos elementos da página, que anteriormente não estavam visíveis. Com estas limitações tidas em conta, procedeu-se à instalação dos restantes módulos e dependências necessárias à implementação da nova abordagem, nomeadamente a instalação do `Selenium` e de um *web driver*, que neste caso o *driver* utilizado é o *chromedriver*.

A classe `Spider` do `Scrapy` é definida essencialmente por três elementos, o atributo `name` e os métodos `start_requests()` e `parse()`. O atributo `name` identifica o `Spider` em questão e necessita de ser único em todo o projeto. O método `start_requests()` retorna um iterável de pedidos, pelos quais o `Spider` iniciará a sua procura. Relativamente ao método `parse()`, será chamado para processar as respostas respetivas aos pedidos que são realizados. Este método possui um parâmetro `response`, uma instância do tipo `TextResponse`, que armazena o conteúdo da página para depois a processar, recorrendo a métodos próprios desta instância ou definidos pelo utilizador. Na classe `Spider`, utilizada para recolher os dados meteorológicos, antes de implementar os métodos `start_requests()` e `parse()`, definiu-se o método `__init__()`.

No excerto 3.5 apresenta-se o método `__init__()` utilizado. Aqui, o `Spider` consegue aceder aos argumentos passados pela linha de comandos e, posteriormente copiá-los como atributos. Uma vez que os argumentos do `Spider` são tratados como `strings`, e como só é necessário enviar o nome do ficheiro para iniciar o *crawling*, não é essencial especificar instruções auxiliares. Para além disto, também é iniciado o `webdriver` pretendido (neste caso o *webdriver* do `Chrome`), o conjunto de dados e a filtragem das colunas necessárias ao funcionamento do `Spider`, bem como algumas variáveis auxiliares.

```
def __init__(self, **kwargs):
        super(ObservationsSpyder, self).__init__(**kwargs)
```

```
        self.driver = webdriver.Chrome()
        self.uk_crash = pd.read_csv('Real_Files/'+self.fileName+'.csv', sep=',')
        self.uk_url_incidents = self.uk_crash.iloc[:,[1,3,7]].copy()
        self.uk_crash['temperature'] = 0.0
        self.uk_crash['humidity'] = 0.0
        self.uk_crash['pressure'] = 0.0
        self.uk_crash['dewPoint'] = 0.0
```

Listagem 3.5: Método `__init__()` da classe `Spider`.

Com este método implementado, procedeu-se à criação do método `start_requests()`. No excerto 3.6 está definido o método `start_requests()`. Para cada *url* que se encontre no conjunto de dados importado, será gerado um objeto do tipo `Request`, o qual irá conter o *url* do incidente e um dicionário de dados necessários à filtragem da tabela recebida do sítio Web, com recurso ao parâmetro `meta`. Para além destes parâmetros, também se define a função `callback` que irá receber e tratar a resposta obtida deste pedido. Alguns *url* gerados são iguais, pois como estão presentes no conjunto de dados incidentes que ocorreram no mesmo dia e na mesma cidade, o *url* que será enviado no pedido é duplicado. Por omissão esta funcionalidade está ativada, mas neste caso foi necessário desativá-la. Esta desativação é realizada ao definir o valor do parâmetro `dont_filter` como `true`, de modo a ignorar pedidos duplicdos. Este método é chamado uma única vez, ao iniciar o `scraping`.

```
def start_requests(self):
        for index, row in self.uk_url_incidents.iterrows():
            yield scrapy.Request(url=row['URL'], callback=self.parse, meta={'time':row['Time'],'
    accident_index':row['Accident_Index']},
                dont_filter=True)
```

Listagem 3.6: Método `start_requests()` da classe `Spider`.

Com os métodos `__init__()` e `start_requests()` implementados, foi possível iniciar a implementação do método `parse()`, que irá tratar as respostas obtidas pelos pedidos efetuados ao *website*. A listagem 3.7 mostra a inicialização do método `parse`. Na variável `tmp` é guardado o nome do elemento alvo do *scrapping*. Para isso utiliza-se a classe `By` do `Selenium`, que fornece diversos mecanismos para obter elementos presentes no `HTML` da resposta. Neste caso, é utilizado o mecanismo `CLASS_NAME`, que irá procurar na resposta o elemento com o nome de classe `observation-table`. De seguida, o `driver` irá aceder ao `url` fornecido pelo objeto do tipo `Response` enviado como argumento, pelo que vai aguardar até que a página esteja completamente disponível. No entanto, como a página a processar é uma SPA, é necessário que o `driver` aguarde que o elemento alvo da extração de dados esteja presente. Para isto recorre-se à utilização de uma espera explícita, que permite controlar o tempo necessário para que uma determinada condição seja verificada, neste caso a presença da tabela com o nome de classe `observation-table` e com um tempo de espera até 10

segundos. Já com a página disponível e a tabela presente, são retiradas da página fonte todas as linhas desse mesmo elemento, com recurso ao `xpath`.

```
def parse(self, response):
        tmp = (By.CLASS_NAME, "observation-table")
        self.driver.get(response.url)
        WebDriverWait(self.driver, 10).until(EC.presence_of_element_located(tmp))
        page_source = self.driver.page_source
        scrapy_selector = Selector(text = page_source)
        rows = scrapy_selector.xpath('//*[@class="mat-table ng-star-inserted"]//tbody//tr')

        partOfDayAndHour = self.getHourAndPartOfDay(response.meta.get('time'))
        partOfDay = partOfDayAndHour[0]
        hour = partOfDayAndHour[1]
        splittedHour = list(hour)

        temperature = [0, 0]
        humidity = [0, 0]
        dewPoint = [0, 0]
        pressure = [0, 0]
```

Listagem 3.7: Inicialização do método `parse()` da classe `Spider`.

De modo a recolher a informação pretendida, é necessário saber a parte do dia do acidente (`AM` ou `PM`) e a hora a que ocorreu. Estes dados foram anteriormente enviados através do atributo `meta`, do objeto `Request` na listagem 3.6. O método utilizado para identificar o dia e a hora do acidente encontra-se na listagem 3.8.

```
def getHourAndPartOfDay(self, time):
        splitHourAndPart = time.split()
        partOfDay = splitHourAndPart[1]
        splitHourAndMinute = splitHourAndPart[0].split(':')
        hour = splitHourAndMinute[0]
        return partOfDay, hour
```

Listagem 3.8: Método para obter o dia e a hora dum acidente.

As variáveis `temperature`, `humidity`, `dewPoint` e `pressure` têm como principal objetivo registar a soma dos valores das respetivas variáveis meteorológicas recolhidas e a sua frequência. A frequência é necessária, pois há casos em que numa hora só existe uma observação e outros em que há duas observações e, uma vez que é pretendido o cálculo da média dos valores das variáveis meteorológicas nesse momento, esta medida é necessária. Com a hora e a parte do dia a que o incidente ocorreu, já é possível tratar as linhas relativas a estas restrições.

Na listagem 3.9, encontra-se o excerto de código responsável pela extração da informação das linhas da tabela. Para cada linha presente na tabela, é verificada a presença ou não da hora e da parte do dia nessa entrada. Caso essa entrada respeite estas condições, são

retirados os valores das respetivas colunas das variáveis pretendidas. Estes valores são adicionados às variáveis `temperature`, `humidity`, `dewPoint` e `pressure` representados na listagem 3.7 e incrementada a sua ocorrência.

```
if splittedHour[0] == '0':
            for row in rows:
                tmpHour = row.xpath('td[1]//text()').extract_first()
                tmpHour = tmpHour.split(':')
                if splittedHour[1] in tmpHour[0] and partOfDay in row.xpath('td[1]//text()').
    extract_first():
                    temperature[0] += float(row.xpath('td[2]//text()').extract_first())
                    temperature[1] += 1
                    humidity[0] += float(row.xpath('td[4]//text()').extract_first())
                    humidity[1] += 1
                    dewPoint[0] += float(row.xpath('td[3]//text()').extract_first())
                    dewPoint[1] += 1
                    pressure[0] += float(row.xpath('td[8]//text()').extract_first())
                    pressure[1] += 1
        else:
            for row in rows:
                tmpHour = row.xpath('td[1]//text()').extract_first()
                tmpHour = tmpHour.split(':')
                if hour in tmpHour[0] and partOfDay in row.xpath('td[1]//text()').extract_first
    ():
                    temperature[0] += float(row.xpath('td[2]//text()').extract_first())
                    temperature[1] += 1
                    humidity[0] += float(row.xpath('td[4]//text()').extract_first())
                    humidity[1] += 1
                    dewPoint[0] += float(row.xpath('td[3]//text()').extract_first())
                    dewPoint[1] += 1
                    pressure[0] += float(row.xpath('td[8]//text()').extract_first())
                    pressure[1] += 1
```

Listagem 3.9: Extração da informação da tabela.

Após extrair esta informação, procedeu-se ao cálculo da média dos valores retirados da tabela, para cada uma das variáveis meteorológicas. Uma vez calculada a média, estes valores têm de ser atribuídos ao incidente que lhes faz referência. De modo a ser possível atribuir os dados ao incidente correto, no momento em que é realizada a chamada ao método `parse()`, no parâmetro `meta` é enviado o índice do incidente para fazer a atribuição. Quando terminada esta fase, os novos dados são guardados num novo ficheiro `CSV`, para ser possível iniciar a fusão destes dados com os rácios dos comprimentos de onda e posteriormente dar início à fase de EDA. A listagem 3.10 apresenta o código desta última fase.

```
conditions = self.calculateAverageConditions(temperature, humidity, pressure, dewPoint)
        self.associateValuesToAccidentIndex(response.meta.get('accident_index'), conditions)

```

```
4         self.uk_crash.to_csv('Scrapped_Files/'+self.fileName+'.csv', header=True)
```

Listagem 3.10: Finalização do `scrapping`.

3.2.2 *Fusão dos sinais de infravermelhos com os dados meteorológicos*

Como o problema da falta de dados referentes às condições meteorológicas está agora colmatado, é necessário relacionar estes dados com os rácios dos sinais infravermelhos. No entanto, uma vez que o conjunto de dados relativos aos rácios dos sinais infravermelhos apenas possuem uma referência à condição do piso no momento da medição, e nenhuma referência às condições meteorológicas, a abordagem tem que ser ajustada.

Ao contrário do conjunto de dados utilizado como base para recolher as condições meteorológicas, o qual possui atributos como localização geográfica, dia e hora, o conjunto de dados relativo aos rácios dos sinais infravermelhos apenas contém os valores associados a cada feixe utilizado e a respetiva condição do piso. Na tabela 5 encontra-se um fragmento do conjunto de dados referido. Aqui, as gamas de infravermelho utilizadas para obtenção dos rácios são omitidas e substituídas respetivamente pelos termos `Lambda1`, `Lambda2`, `Lambda3` e `Lambda4`, de modo respeitar o Non Disclosure Agreement (NDA) assinado antes do desenvolvimento da dissertação.

Tabela 5: Exemplo dos dados relativos aos rácios dos sinais infravermelhos.

Class	Lambda1	Lambda2	Lambda3	Lambda4
Dry	0.077587	0.090570	0.093068	0.100581
Dry	0.083475	0.098579	0.100855	0.108881
Dry	0.074316	0.084427	0.086365	0.091904

Tal como referido, a inexistência de qualquer referência que sirva de elo de ligação entre os dois conjuntos de dados torna a fusão dos mesmos mais sensível e sujeita a incoerências. Porém, os dois conjuntos de dados possuem um atributo em comum, a condição da superfície do piso, o que possibilita estabelecer uma ligação. A abordagem utilizada consistiu em gerar dados sintéticos a partir do conjunto de dados naturais cedidos pelos investigadores do projeto `Sensible Car`. No entanto, a geração destes dados sintéticos não pode ser realizada de uma maneira aleatória. Para cada classe presente no conjunto de dados com os rácios dos sinais infravermelhos (`Dry`, `Water` e `Ice`), extrai-se a média e o desvio padrão de cada sinal, ou seja, de cada um dos `lambdas`. Desta forma, é possível replicar a distribuição dos dados de cada `lambda` nas diferentes condições do piso. Na listagem 3.11 apresenta-se o código relativo à extração da média e desvio padrão para a classe `Dry`. O processo é o mesmo para as restantes classes.

```
def extractMeanAndStdDry():
    # Mean and Standard Deviation from 'Lambda1'
    wavelength_dry_lambda1_mean = wavelength_dry['Lambda1'].mean()
    wavelength_dry_lambda1_std = wavelength_dry['Lambda1'].std()
    # Mean and Standard Deviation from 'Lambda2'
    wavelength_dry_lambda2_mean = wavelength_dry['Lambda2'].mean()
    wavelength_dry_lambda2_std = wavelength_dry['Lambda2'].std()
    # Mean and Standard Deviation from 'Lambda3'
    wavelength_dry_lambda3_mean = wavelength_dry['Lambda3'].mean()
    wavelength_dry_lambda3_std = wavelength_dry['Lambda3'].std()
    # Mean and Standard Deviation from 'Lambda4'
    wavelength_dry_lambda4_mean = wavelength_dry['Lambda4'].mean()
    wavelength_dry_lambda4_std = wavelength_dry['Lambda4'].std()

    return wavelength_dry_lambda1_mean, wavelength_dry_lambda1_std,
           wavelength_dry_lambda2_mean, wavelength_dry_lambda2_std,
           wavelength_dry_lambda3_mean, wavelength_dry_lambda3_std,
           wavelength_dry_lambda4_mean, wavelength_dry_lambda4_std
```

Listagem 3.11: Extração da média e desvio padrão para a classe Dry.

Após extrair as médias e os desvios padrão para cada uma das classes, já é possível gerar os dados sintéticos para cada uma das amostras do conjunto de dados meteorológicos. Para isso, utiliza-se uma função que gera dados aleatórios, respeitando a média e o desvio padrão de um determinado tipo de piso, e seguindo uma distribuição Gaussiana ou normal. Esta função está representada na listagem 3.12.

```
# function that generates the random wavelength reflection based in
# the mean and standard deviation of that road surface condition
def generateRandomGaussianValue(mean, std):
  return np.random.normal(loc=mean, scale=std)
```

Listagem 3.12: Função para gerar dados sintéticos com distribuição normal.

Um dos problemas encontrados na geração de dados sintéticos para cada classe de piso, resultou do fato de o conjunto de dados cedido pelos investigadores do projeto `Sensible Car` apenas possuir as classes `Dry`, `Water` (correspondente a piso molhado) e `Ice`, enquanto que o conjunto de dados meteorológicos possui também a classe `Snow`, além das três mencionadas. De acordo com Casselgren et al. (2007), o asfalto seco e o asfalto com neve são superfícies refletoras difusas. Deste modo, para compensar a ausência da classe `Snow` nos rácios de comprimentos de onda, os dados sintéticos gerados para a classe `Snow` seguem a mesma distribuição da classe `Dry`. Para cada condição de piso, a função apresentada na listagem 3.13 gera valores diferentes tendo em conta a média e desvio padrão retirados anteriormente. Este processo repete-se para os restantes sinais infravermelhos utilizados.

```
def inputLambda1(road_surface_condition):
  if ((road_surface_condition == 'Dry') | (road_surface_condition == 'Snow')):
```

```
    return generateRandomGaussianValue(wavelength_dry_lambda1_mean, wavelength_dry_lambda1_std)
  elif (road_surface_condition == 'Wet'):
    return generateRandomGaussianValue(wavelength_water_lambda1_mean,
    wavelength_water_lambda1_std)
  elif (road_surface_condition == 'Ice'):
    return generateRandomGaussianValue(wavelength_ice_lambda1_mean, wavelength_ice_lambda1_std)
  else:
    return 0.0
```

Listagem 3.13: Geração de dados sintéticos.

Uma vez que se dispõe de todas as funções necessárias à geração de dados sintéticos, já é possível anexar os rácios de comprimentos de onda ao conjunto de dados referente aos valores meteorológicos. Dado um *dataframe* com os dados meteorológicos já importados, a função representada na listagem 3.14 tem por objetivo anexar quatro novas colunas, referentes aos rácios dos sinais infravermelhos, geradas com base nos dados reais.

```
def inputWavelengthData(meteo_data):
    meteo_data['Lambda1'] = meteo_data.apply(lambda row : inputLambda1(row['
    Road_Surface_Conditions']), axis = 1)
    meteo_data['Lambda2'] = meteo_data.apply(lambda row : inputLambda2(row['
    Road_Surface_Conditions']), axis = 1)
    meteo_data['Lambda3'] = meteo_data.apply(lambda row : inputLambda3(row['
    Road_Surface_Conditions']), axis = 1)
    meteo_data['Lambda4'] = meteo_data.apply(lambda row : inputLambda4(row['
    Road_Surface_Conditions']), axis = 1)

    return meteo_data
```

Listagem 3.14: Anexação dos dados relativos aos sinais infravermelhos.

Quando estiverem concluídos todos estes passos, possui-se o conjunto de dados pretendido, o qual representa a fusão dos dados provenientes de estações meteorológicas e do RCS. Na tabela 6 encontra-se um excerto do conjunto de dados após a fusão.

Tabela 6: Conjunto de dados após a fusão.

Atributos	**Valores**		
Weather_Conditions	Fine	Snow	Fine
temperature	45.5	35.0	37.0
humidity	84.0	90.0	87.0
pressure	29.78	29.52	29.46
dewPoint	41.0	34.0	34.0
Lambda1	0.056205	0.023744	0.083427
Lambda2	0.075471	0.009376	0.114120
Lambda3	0.019486	0.009787	0.093949
Lambda4	0.109250	0.008492	0.122716
Road_Surface_Conditions	Dry	Wet	Snow

3.3 ANÁLISE E EXPLORAÇÃO DE DADOS

EDA é um processo de pesquisa e descoberta de conhecimento, que o olho humano não consegue identificar, pelo menos em tempo útil e sem um grande esforço por parte da pessoa. Assim, o principal objetivo da EDA é facilitar a identificação de padrões no conjunto de dados, realçar as suas principais caraterísticas e, por meio de elementos visuais, revelar o comportamentos dos dados e das suas variáveis num contexto singular, ou enaltecer relações entre outras variáveis. Esta fase, permite também a determinação da qualidade do conjunto de dados em questão e, sendo este um processo iterativo, a constante recorrência à EDA revela-se de grande importância, pois faz diferença na compreensão dos resultados obtidos da aplicação, neste caso de algoritmos de aprendizagem automática.

Uma vez que o conjunto de dados utilizado foi criado a partir da fusão dos dados recolhidos pela técnica de `Web scrapping`, aplicada ao sítio Web que contém o histórico de dados meteorológicos em diversas localizações, com os rácios dos sinais infravermelhos, gerados com base nos dados reais cedidos pelos investigadores do projeto `Sensible Car`, é importante ter uma noção da qualidade do mesmo e do comportamento das variáveis.

Numa primeira fase, começa-se com uma visualização superficial do conjunto de dados. Utilizando o método `info()` é possível obter esta visão geral dos dados, que neste caso constituem um total de aproximadamente `20000` registos e `9` colunas, sem contabilizar a variável dependente. Estas colunas correspondem aos valores de temperatura, humidade, pressão atmosférica, ponto de orvalho, condições atmosféricas e os quatro valores referentes aos rácios dos comprimentos de onda utilizados. Os valores relativos às variáveis temperatura e ponto de orvalho estão representados em `Fahrenheit`, a humidade está em percentagem, a pressão atmosférica em `Hg`, ou seja, é a pressão acima do nível médio da água do mar, e cada `Lambda` representa o rácio entre o sinal refletido e emitido para um dado comprimento de onda. No início, a amostra total dos dados correspondia a `25000` registos, mas como em determinadas datas ainda não existiam sensores de recolha de dados meteorológicos nas respetivas localizações, não foi possível extrair essa informação. Daqui, procedeu-se à eliminação destas entradas, que no final contabilizam um total de `20000` registos válidos. Destes `20000` registos, aproximadamente `6000` são duplicados. Neste caso, como se trata de variáveis meteorológicas, os registos duplicados entendem-se como dias que possuem os mesmos valores de temperatura e humidade por exemplo. A tabela 6 contém um excerto do conjunto de dados a utilizar.

Neste conjunto de dados não existem valores nulos e há dois tipos de dados: `float64`, nos valores das colunas `temperature`, `humidity`, `pressure`, `dewPoint`, `Lambda1`, `Lambda2`, `Lambda3` e `Lambda4`, e `object`, na representaçãoo da variável `Weather_Conditions`. Como a variável `Weather_Conditions` possui um número limitado de valores, `Fine`, `Rain`, `Fog` e `Snow`, foi convertida para o tipo `category`, recorrendo à função `asType()`.

Para obter informação sobre as características das variáveis numéricas, a tabela 7 apresenta a distribuição destas variáveis. É possível verificar que não existem valores negativos no conjunto de dados, a não ser nos rácios dos sinais infravermelhos. No entanto, é possível constatar que existem valores discrepantes, uma vez que os valores mínimos de todas as variáveis meteorológicas estão bastante afastados do 25º percentil.

Tabela 7: Distribuição das variáveis numéricas.

	count	mean	std	min	25%	50%	75%	max
temperature	18726.0	49.404092	13.237617	0.000000	40.000000	51.000000	59.000000	84.500000
humidity	18726.0	77.024917	16.057772	0.000000	67.678571	79.854167	88.000000	100.000000
pressure	18726.0	29.382888	3.286700	0.000000	29.480000	29.760000	30.010000	30.780000
dewPoint	18726.0	39.849358	14.538982	0.000000	34.000000	44.000000	50.000000	67.000000
Lambda1	18726.0	0.066054	0.037487	-0.098945	0.034413	0.062823	0.093701	0.210298
Lambda2	18726.0	0.070632	0.043386	-0.145528	0.034593	0.069227	0.103461	0.230219
Lambda3	18726.0	0.065009	0.050720	-0.100549	0.015190	0.070602	0.105519	0.224803
Lambda4	18726.0	0.068267	0.054131	-0.126686	0.017436	0.073133	0.111484	0.238261

Com o objetivo de identificar visualmente a ocorrência de valores discrepantes, são utilizados diagramas de caixa para cada uma das variáveis numéricas. A partir da figura 11, relativa aos diagramas de caixas das variáveis meteorológicas, é possível verificar que existem valores discrepantes em todas as variáveis numéricas. Ora, alguns destes registos representam apenas situações extraordinárias de condições meteorológicas, ao passo que outras, nomeadamente as instâncias que contêm o valor `0`, referem-se a valores em falta. Existem muitas variações deste tipo de valores e nestes casos, convém primeiro detetar o tipo dos dados em falta. Estes dados podem dividir-se em três tipos: Valor em falta completamente aleatório (MCAR), Valor em falta por influência de outras variáveis (MAR) ou Valor em falta não aleatório (MNAR). Após analisar os diagramas de caixa, em conjunto com algumas amostras do conjunto de dados, conclui-se que os valores em falta são do tipo MCAR. Isto resulta de situações em que alguns sensores estão inativos/inoperacionais, em determinadas horas do dia, ou quando ocorreu um erro na transmissão de dados entre o emissor (estação meteorológica) e o recetor (sítio Web). O único caso em que o valor em falta poderia ser do tipo MAR será o `Dew Point`, pois é uma característica que depende inteiramente dos valores da temperatura e humidade. No entanto, há amostras em que o valor do `Dew Point` está presente, mas o valor da temperatura está a `0`. Para este caso há duas soluções possíveis: (i) remover as amostras que incluem valores a `0`, ou (ii) calcular os valores em falta recorrendo às variáveis preenchidas.

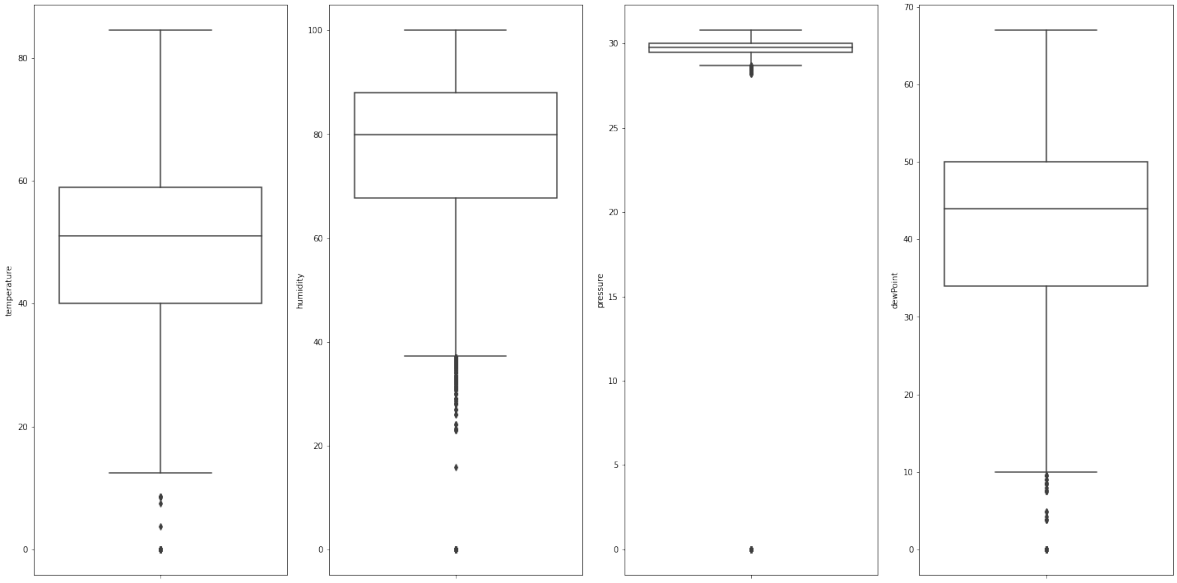


Figura 11: Diagrama de caixa das variáveis numéricas meteorológicas.

Os diagramas de caixa, incluídos na figura 12, dizem respeito aos rácios dos sinais infravermelhos e demonstram que existem poucos valores discrepantes.

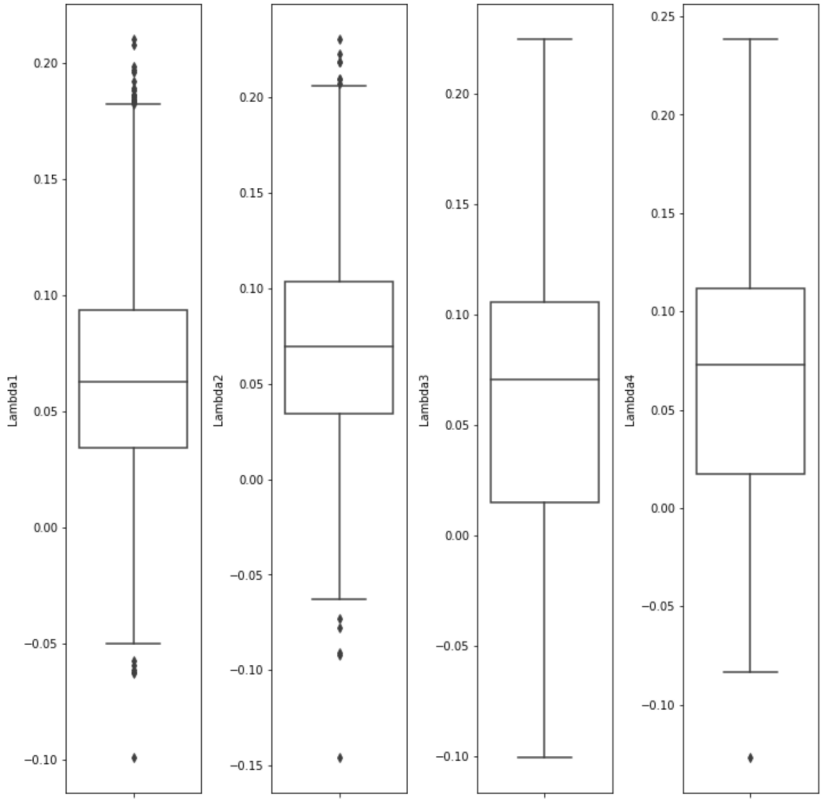


Figura 12: Diagrama de caixa dos rácios dos sinais infravermelhos.

Relativamente às duas variáveis categóricas presentes no conjunto de dados utilizado, `Weather_Conditions` e `Road_Surface_Conditions`, um dos principais objetivos de efetuar a EDA sobre este tipo de atributos é identificar valores únicos e a respetiva contagem. Na figura 13 estão representados os quatro valores possíveis para a variável que faz referência às condições atmosféricas, `Weather_Conditions`. Estes valores dividem-se em: (i) `Fine`, correspondente a condições atmosféricas com céu limpo, (ii) `Fog`, que indica a presença de nevoeiro, (iii) `Rain`, que sinaliza a ocorrência de chuva e (iv) `Snow`, que indica a presença de neve. Pela figura é possível verificar a elevada quantidade de amostras para a condição `Fine` e uma quantidade quase residual de amostras em que o acidente ocorreu com nevoeiro.

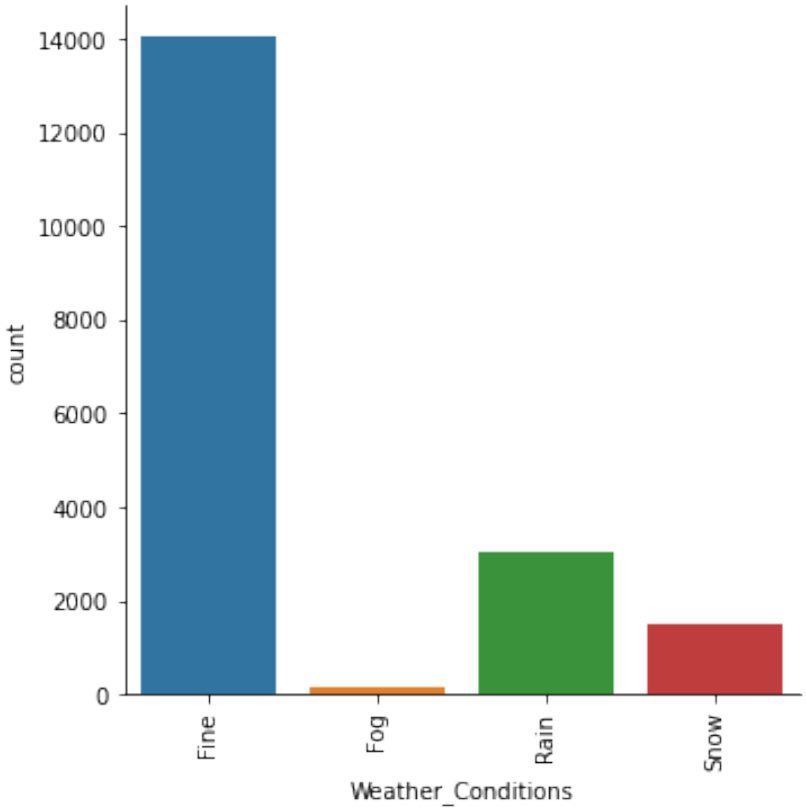


Figura 13: Contagem de valores únicos de `Weather_Conditions`.

Relativamente à variável que representa a condição do piso, apresentada na figura 14, mais de 10000 acidentes ocorreram em condições de piso seco, enquanto aproximadamente 5000 ocorreram em piso molhado. Em comparação com o gráfico de barras da figura 13, apesar de a maioria dos acidentes terem ocorrido em condições de atmosféricas não adversas, o número de amostras que registam piso seco, não igualam o número de amostras de condições atmosféricas favoráveis.

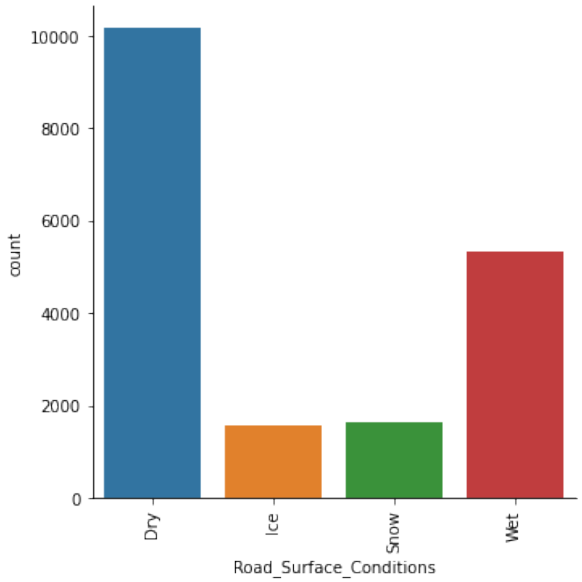


Figura 14: Contagem de valores únicos de `Road_Surface_Conditions`.

Com isto, é possível concluir que apesar de as condições estarem favoráveis, não quer dizer que o piso não esteja por exemplo molhado ou com gelo. A figura 15 confirma esta conclusão.

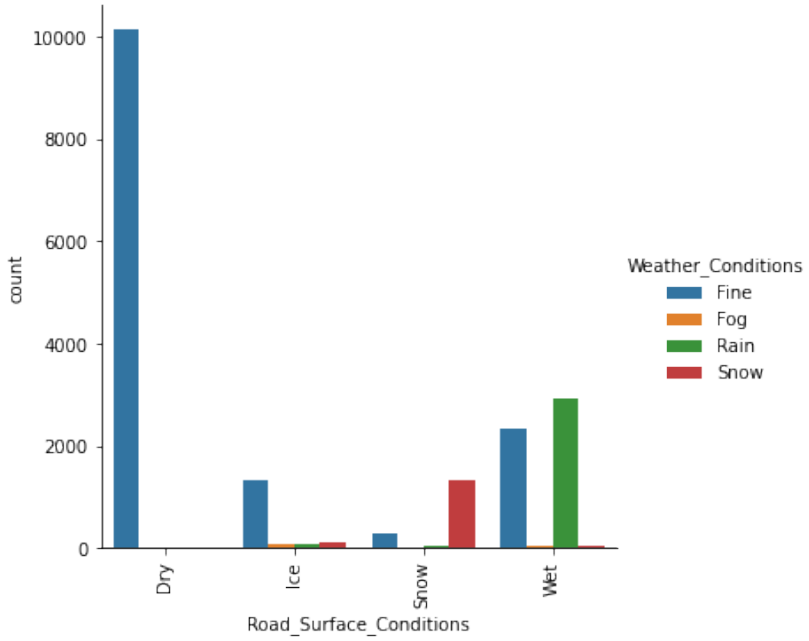


Figura 15: `Road_Surface_Conditions` vs `Weather_Conditions`.

Como é possível observar na figura 15, praticamente todas as amostras relativas a piso seco correspondem a condições atmosféricas favoráveis, ao passo que nas restantes condições

de piso, como por exemplo em piso molhado, o piso pode encontrar-se molhado e não estar a chover nesse momento. Com isto, dada esta similaridade dos valores meteorológicos para condições de piso seco e molhado por exemplo, poderá dificultar o bom desempenho no treino e posterior capacidade de classificação, por parte do algoritmo de aprendizagem automática.

Outro aspeto importante a analisar é o comportamento das variáveis para as diferentes condições de piso, ou seja, verificar se uma determinada variável apresenta uma distribuição dos valores que varia com o tipo de piso. No diagrama inferior direito da figura 16, verifica-se que as distribuições dos valores da temperatura, paras pisos seco (azul) e molhado (vermelho), se concentram na mesma região. No entanto, é possível identificar que no piso molhado há uma concentração de amostras, para temperaturas inferiores, ligeiramente superior ao que ocorre no piso seco. Relativamente aos valores da humidade, apresentados no diagrama inferior esquerdo, é possível verificar que já ocorre uma diferença significativa na distribuição de valores entre as condições de piso seco (azul) e molhado (vermelho). Enquanto que em condições de piso seco os valores de humidade se encontram mais dispersos, para o piso molhado estes concentram-se na região acima de `70%` de humidade relativa. Quanto ao ponto de orvalho, representado no diagrama superior esquerdo, nota-se uma concentração de valores na mesma região nos pisos secos e molhados. Ainda relativamente aos pisos com neve e gelo, apesar de apresentarem praticamente a mesma distribuição que os outros 2 tipos de piso, há uma concentração de valores ligeiramente superior na temperatura de `30°F`. Relativamente à pressão atmosférica, visível no diagrama superior direito da figura 16, verifica-se que não existe variação significativa da distribuição dos valores com a condição do piso.

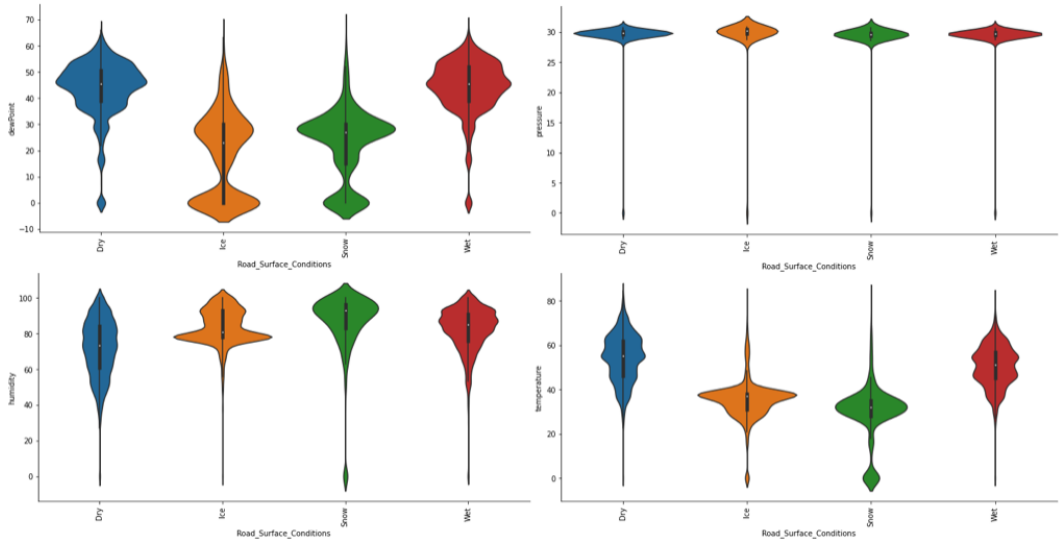


Figura 16: Distribuição dos valores das variáveis meteorológicas com a condição do piso.

Uma vez analisada a distribuição dos valores das variáveis meteorológicas para diferentes condições de piso, resta analisar a influência da condição do piso na distribuição dos valores dos rácios dos sinais infravermelhos. Na figura 17 estão representados os quatro sinais utilizados em relação à condição do piso. Verifica-se que todos os rácios apresentam o mesmo tipo de distribuição dos valores. Embora mínimas, é possível notar algumas diferenças. As distribuições dos valores de *Lambda1* e *Lambda3*, em condições de piso com neve, apresentam a mesma forma. Para a mesma condição de piso, *Lambda2* e *Lambda4* apresentam uma menor dispersão dos valores. O mesmo acontece nos casos de piso com gelo e seco. Em relação à condição de piso molhado, e para qualquer um dos *lambdas*, a distribuição de valores apresenta a mesma forma, a qual é bem distinta da que se verifica nas restantes condições, o que auxilia a classificação de novos casos.

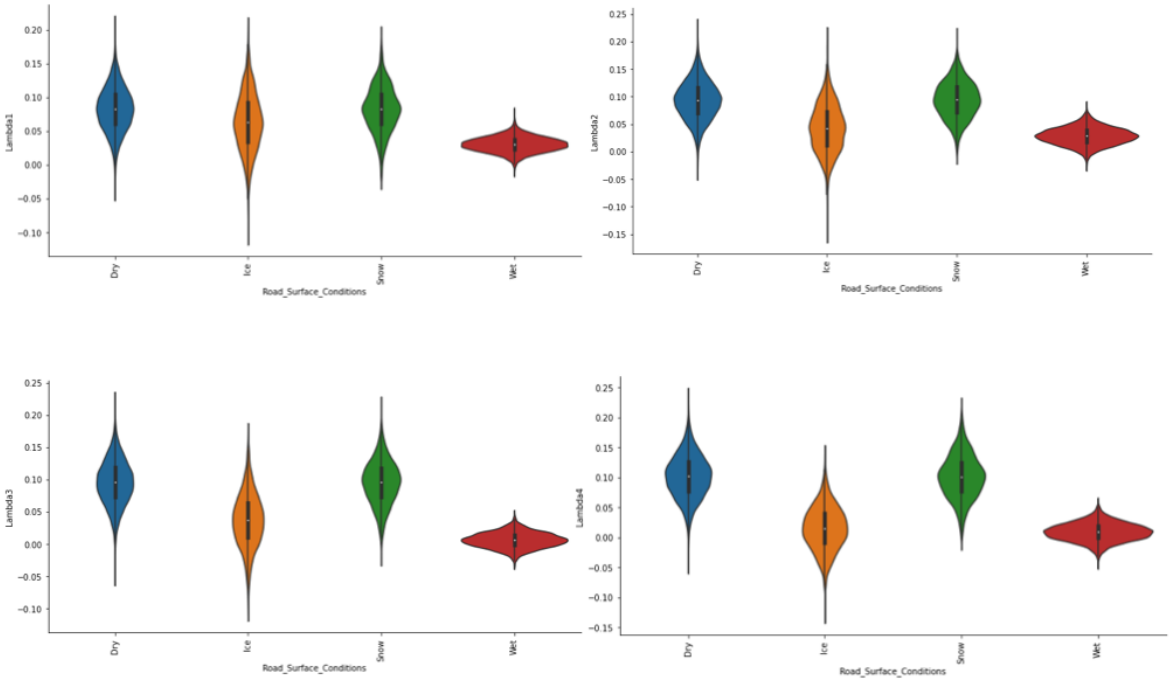


Figura 17: Distribuição dos valores dos rácios dos sinais infravermelhos com a condição do piso.

Para além do estudo da distribuição dos valores de todas as variáveis a um nível singular, também é necessário avaliar a distribuição dos valores das mesmas para com outras variáveis presentes no conjunto de dados. Para isso, na figura 18 está representado um conjunto de gráficos que têm como objetivo avaliar a relação entre cada par de variáveis numéricas presentes no conjunto de amostras recolhidas. A partir destes gráficos repara-se que existe uma relação positiva entre a temperatura e o ponto de orvalho. Esta relação é explicada com a fórmula de cálculo do ponto de orvalho, que por sua vez utiliza a temperatura. Uma outra relação que é possível identificar em dois dos gráficos, é a relação negativa existente

entre os valores da temperatura e da humidade, dado que ao aumentar a temperatura a humidade tende a diminuir. Na figura 18 verifica-se também a existência de valores em falta anteriormente referidos. Para além disto, é possível verificar que as variáveis correspondentes aos rácios dos sinais infravermelhos não apresentam qualquer tipo de correlação com as variáveis meteorológicas, o que representa uma vantagem relativamente ao treino dos algoritmos de classificação. Nas variáveis dos rácios dos sinais infravermelhos, nota-se também a existência de valores em falta, representados como `0`. Estes valores em falta, à semelhança do que foi referido para as variáveis meteorológicas, são do tipo MCAR. Verifica-se ainda que os rácios dos comprimentos de onda também não se correlacionam entre si, apesar de este aspeto não se encontrar representado na figura 18.

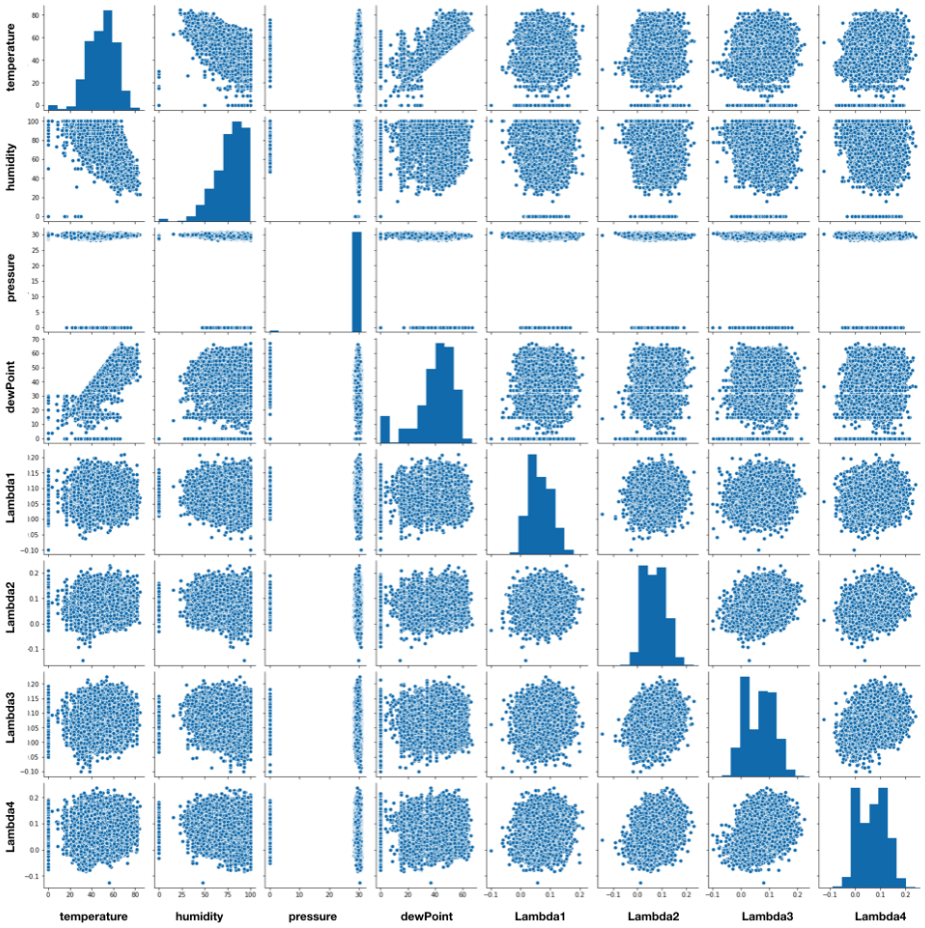


Figura 18: Distribuição entre pares.

3.4 PREPARAÇÃO DOS DADOS

Na seção 3.3 foi identificada a presença de vários valores em falta, em que neste caso são valores que as estações responsáveis pela recolha dos dados meteorológicos não estavam funcionais, ou não possuem o sensor que recolhe um certo tipo de variável, como por exemplo a temperatura. Estes valores, que estão identificados com `0.0`, podem ser calculados analiticamente. Por exemplo o ponto de orvalho pode ser calculado a partir dos valores da temperatura e da humidade. No entanto, há dois tipos de situações relativas aos dados em falta do conjunto de dados utilizado. Uma situação corresponde a amostras em que falta apenas o valor de uma variável, enquanto que a outra situação corresponde aos casos em que faltam duas ou mais variáveis. Nos casos em que apenas está em falta uma variável, utiliza-se a fórmula que a permite calcular, caso faltem duas variáveis ou mais, insere-se a média dos valores da coluna correspondente.

Numa primeira fase, retira-se a média de cada coluna onde existem valores identificados como `0.0`. No entanto, a média da humidade relativa em condições de piso seco e piso molhado muitas vezes não é a mesma e, tendo em conta esta característica, a média de um determinado atributo é calculada consoante a condição do piso. O excerto de código da listagem 3.15 extrai a média dos valores da temperatura. Para os restantes atributos a abordagem é a mesma, apenas é necessário alterar o nome do atributo de `temperature` para `humidity` ou `dewPoint`. Contudo, para calcular a média de um determinado atributo, as amostras que contém este valor em falta não entram nos cálculos.

```
1 mean_temp_dry = road_surface_meteo_cond.loc[(road_surface_meteo_cond['Road_Surface_Conditions']
      == 'Dry') & (road_surface_meteo_cond['temperature'] != 0.0)]['temperature'].mean()
2
3 mean_dewPoint_dry = road_surface_meteo_cond.loc[(road_surface_meteo_cond['
      Road_Surface_Conditions'] == 'Dry') & (road_surface_meteo_cond['dewPoint'] != 0.0)]['
      dewPoint'].mean()
4
5 mean_hum_dry = road_surface_meteo_cond.loc[(road_surface_meteo_cond['Road_Surface_Conditions']
      == 'Dry') & (road_surface_meteo_cond['humidity'] != 0.0)]['humidity'].mean()
```

Listagem 3.15: Média de atributos por condição de piso.

Uma vez calculadas as médias que vão ser inseridas nas amostras que possuem dois ou mais atributos em falta, pode-se calcular os valores dos atributos temperatura, humidade e ponto de orvalho, a inserir nas amostras em que apenas está em falta um dos atributos. A listagem 3.16 contém o código das funções responsáveis por esse mesmo cálculo.

```
1 # Calculo da temperatura
2 # rh - humidade relativa
3 # dp - ponto de orvalho (Celsius)
4 def calculateTemp(rh, dp):
```

```
    t = 243.04*(((17.625*dp)/(243.04+dp))-np.log(rh/100))/(17.625+np.log(rh/100)-((17.625*dp)
    /(243.04+dp)))
    return t
# Calculo do ponto de orvalho
# rh - humidade relativa
# t - temperatura (Celsius)
def calculateDewPoint(rh, t):
    dp = 243.04*(np.log(rh/100)+((17.625*t)/(243.04+t)))/(17.625-np.log(rh/100)-((17.625*t)
    /(243.04+t)))
    return dp
# Calculo da humidade relativa
# t - temperatura (Celsius)
# dp - ponto de orvalho
def calculateHumidity(t, dp):
    rh = 100*(np.exp((17.625*dp)/(243.04+dp))/np.exp((17.625*t)/(243.04+t)))
    return rh
```

Listagem 3.16: Funções para calcular valores das variáveis meteorológicas.

Os valores da temperatura e do ponto de orvalho, presentes no conjunto de dados, encontram-se em graus *Fahrenheit*. Como as fórmulas apresentadas na listagem 3.16 efetuam as operações em graus *Celsius*, foi necessário implementar a conversão entre os dois tipos de unidade (listagem 3.17).

```
# Fahrenheit to Celsius
def fahrenheitToCelsius(t):
    c = (t - 32) * 5.0/9.0
    return c
# Celsius to Fahrenheit
def celsiusToFahrenheit(t):
    f = 9.0/5.0 * t + 32
    return f
```

Listagem 3.17: Funções para conversão da unidade de temperatura.

O próximo passo consistiu em implementar uma função que, ao ser aplicada ao conjunto de amostras, insira corretamente os dados em falta, tanto nos casos em que exista apenas um dado em falta, como nos casos onde existem dois ou mais dados em falta. No excerto de código representado na listagem 3.18 estão implementadas as três funções responsáveis pela inserção dos dados em falta. São implementadas três funções diferentes, para se poder controlar a ordem pela qual são inseridos os dados. Uma vez que o ponto de orvalho depende da temperatura do ar e da humidade relativa, estes dois últimos são calculados primeiro e só depois se calcula o ponto de orvalho. Usando como exemplo a função `fillTempNullValues`, começa por verifica se o valor da temperatura do ar está realmente em falta. Se o valor for igual a `0.0` e os valores correspondentes ao ponto de orvalho e humidade relativa forem maiores que `0.0`, efetua-se o cálculo da mesma. Caso estes dois

últimos valores estejam também em falta, são substituídos pela média correspondente à condição do piso em questão. O processo é análogo nos métodos `filldewPointNullValues` e `fillHumidityNullValues`.

```
# Calculo da temperatura do ar consoante a humidade relativa e o ponto de orvalho
def fillTempNullValues(t, rh, dp, road_condition):
    if t == 0.0:
        if (rh > 0.0) & (dp > 0.0):
            dpCelsius = fahrenheitToCelsius(dp)
            tCelsius = calculateTemp(rh, dpCelsius)
            tFahrenheit = celsiusToFahrenheit(tCelsius)
            return tFahrenheit
        elif (rh == 0.0) | (dp == 0.0):
            value = checkTemperatureByCondition(road_condition)
            return value
        else:
            return t
    else:
        return t
# Calculo do ponto de orvalho consoante a humidade relativa e a temperatura do ar
def filldewPointNullValues(t, rh, dp, road_condition):
    if dp == 0.0:
        if (rh > 0.0) & (t > 0.0):
            tCelsius = fahrenheitToCelsius(t)
            dpCelsius = calculateDewPoint(rh, tCelsius)
            dpFahrenheit = celsiusToFahrenheit(dpCelsius)
            return dpFahrenheit
        elif (rh == 0.0) | (t == 0.0):
            value = checkDewPointByCondition(road_condition)
            return value
        else:
            return dp
    else:
        return dp
# Calculo da humidade relativa consoante a temperatura do ar e ponto de orvalho
def fillHumidityNullValues(t, rh, dp, road_condition):
    if rh == 0.0:
        if (dp > 0.0) & (t > 0.0):
            tCelsius = fahrenheitToCelsius(t)
            dpCelsius = fahrenheitToCelsius(dp)
            calculatedRh = calculateHumidity(tCelsius, dpCelsius)
            return calculatedRh
        elif (t == 0.0) | (dp == 0.0):
            value = checkHumidityByCondition(road_condition)
            return value
        else:
            return rh
    else:
```

```
45          return rh
```

Listagem 3.18: Funções para inserir os dados em falta nas amostras.

Uma vez inseridos os dados em falta, através das funções anteriormente apresentadas, já é possível preparar os dados para que estes sejam utilizados no treino dos algoritmos de aprendizagem automática. Como o conjuntos de dados contém um atributo categórico, é necessário codificar a sua gama de valores para se poder treinar os algoritmos. Para além da codificação do atributo categórico, realiza-se a normalização dos atributos numéricos, não só para normalizar os intervalos de valores, mas também para diminuir a capacidade de cálculo necessária durante o treino (listagem 3.19).

```
# Pipeline numerico
num_pipeline = Pipeline([
        ('std_scaler', StandardScaler()),
    ])
# Pipeline categorico
cat_pipeline = Pipeline([
        ('one_hot', OneHotEncoder()),
    ])
# Pipeline final constituido pelo pipeline numerico e pipeline categorico
full_pipeline = ColumnTransformer([
        ("num", num_pipeline, num_attributes),
        ("cat", cat_pipeline, cat_attributes),
    ])
# Aplicacao do pipeline aos dados de treino
road_surface_prepared = full_pipeline.fit_transform(X_train)
```

Listagem 3.19: *Pipeline* de preparação dos dados.

3.5 CONFIGURAÇÃO DO DISPOSITIVO EDGE

Antes de iniciar a implementação do algoritmo de aprendizagem automática selecionado, é necessário configurar o dispositivo *edge*. Numa primeira fase, procede-se à preparação do cartão microSD que irá conter a imagem do *NVIDIA Jetson Nano Developer Kit*. Para tal, é necessário utilizar um outro computador, no qual a imagem será descarregada e posteriormente escrita no cartão microSD. Este processo de escrita da imagem no cartão microSD pode ser realizado a partir da linha de comandos ou com recurso a uma aplicação como o Etcher (2020). Todo os passos de configuração da imagem estão descritos para cada sistema operativo na página oficial da NVIDIA (2020).

O próximo passo consiste em instalar o `pip3` para futuras operações necessárias. Para além da instalação do `pip3`, também é preciso instalar o `Cython`, um compilador otimizado para a linguagem `Python`. De seguida, procede-se à instalação do `pandas` e da versão `1.1.0` do `scipy`, compatível com a implementação do modelo de aprendizagem automática e com

o `scikit-learn`. Com todas as dependências resolvidas, instala-se o `scikit-learn`. Este último passo necessita de vários minutos para concluir a sua instalação, pois o processo de compilação deste componente é bastante complexo.

Para além dos passos anteriormente descritos e, uma vez que serão testadas redes neuronais neste dispositivo, instalou-se o `TensorFlow`. Antes de instalar o `TensorFlow` têm que se instalar alguns pacotes dos quais ele depende. Todos estes passos estão resumidos na listagem 3.20.

```
# Cython instalation
$ sudo -H pip3 install cython
# Pandas instalation
$ sudo -H pip3 install pandas
# Instalation of the version 1.1.0 of scipy
$ sudo -H python3 -m pip install scipy==1.1.0
# Scikit-learn instalation
$ sudo -H pip3 install scikit-learn
# Package Python dependencies instalation
$ sudo pip3 install -U numpy==1.16.1 future==0.17.1 mock==3.0.5 h5py
    ==2.9.0 keras_preprocessing==1.0.5 keras_applications==1.0.8 gast
    ==0.2.2 futures protobuf pybind11
# Instalation of the last version of TensorFlow
$ sudo pip3 install --pre --extra-index-url https://developer.download.
    nvidia.com/compute/redist/jp/v44 tensorflow
```

Listagem 3.20: Configuração do dispositivo edge.

3.6 APLICAÇÃO DE ALGORITMOS DE CLASSIFICAÇÃO

Um dos objetivos da presente dissertação é a seleção de um algoritmo de aprendizagem automática que simultaneamente classifique corretamente a condição do piso e que exija recursos computacionais/armazenamento moderados. Deve ainda ser capaz de classificar as amostras a uma taxa que o torne viável ao apoio à condução automóvel. Uma vez que as fases de recolha, análise e tratamento de dados se encontram concluídas, inicia-se a aplicação de diversos algoritmos de aprendizagem automática, de modo a apurar quais destes satisfazem os requisitos definidos. Nesta seção, apresentam-se os algoritmos e os parâmetros utilizados na sua otimização.

Uma vez concluído o processo de tratamento dos dados, pode avançar-se para o treino dos algoritmos de classificação e para a procura dos hiperparâmetros que otimizem a classificação do estado do piso. Tal como já foi referido, no início da presente dissertação não se sabia qual o algoritmo mais adequado para a classificação do estado do piso e que respeitasse as condições impostas (boa capacidade de classificação e baixa necessidade de recursos computacionais). Todos os algoritmos já apresentados na secção 2.3 serão alvo de

estudo, de modo a apurar o mais adequado. Para cada algoritmo de classificação utiliza-se validação cruzada, recorrendo ao método `GridSearchCV` da biblioteca `scikit learn`. Ao mesmo tempo que é possível obter a classificação média, este método devolve também os hiperparâmetros que melhor regularizam o algoritmo de classificação em questão.

O primeiro algoritmo a ser testado é a **Regressão Logística**. Neste caso, a otimização procura os melhores valores para sete hiperparâmetros: `penalty`, `tol`, `C`, `fit_intercept`, `solver`, `multi_class` e `max_iter`. O hiperparâmetro `penalty` define a regra utilizada como penalização durante a fase de aprendizagem. `tol` é a tolerância no critério de paragem, que notifica o algoritmo se este já se encontrar suficientemente próximo de um máximo ou mínimo. O hiperparâmetro `C` corresponde à regularização do algoritmo. Quanto maior for o valor de `C`, menor é a intensidade da regularização. O hiperparâmetro `fit_intercept` indica se uma constante, como o *bias* ou *intercept*, devem ser adicionada à função que toma a decisão. O `solver` define qual o algoritmo utilizado na otimização do problema em questão. Para este hiperparâmetro, a otimização tem que ser reajustada, uma vez que certos algoritmos de otimização só são compatíveis com algumas regras de penalização, como é o caso dos algoritmos `newton-cg`, `lbfgs` e `sag`, que apenas suportam a penalização `l2` ou nenhuma. O hiperparâmetro `multi_class` especifica a maneira como o problema será abordado. Por exemplo, se o valor for `ovr`, para cada uma das classes existentes, o algoritmo irá substituir a classe positiva (classe que se pretende identificar) pelo valor `1` e `0` as as restantes classes. Esta estratégia denomina-se de um contra todos ou *one versus rest*. No entanto, se o parâmetro tomar o valor `multinomial`, o problema será abordado sem alteração à variável dependente ou seja, as etiquetas das classes originais são usadas. Por último, `max_iter` define o número máximo de iterações a executar pelo algoritmo. Na tabela 8 encontram-se todos os valores dos hiperparâmetros utilizados.

Tabela 8: Parâmetros utilizados no algoritmo de Regressão Logística.

Parâmetro	Valores
penalty	*'l1', 'l2', 'elasticnet'*
tol	0.0001, 0.001, 0.01, 0.1, 1
C	0.01, 0.1, 1, 10
fit_intercept	*'true', 'false'*
solver	*'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'*
multi_class	*'auto', 'ovr', 'multinomial'*
max_iter	100, 1000, 10000, 100000

No **algoritmo K-NN** são quatro os hiperparâmetros a otimizar: `n_neighbors`, `weights`, `algorithm` e `p`. O hiperparâmetro `n_neighbors` corresponde, tal como o nome indica, ao número de vizinhos presentes. `weights` corresponde à função utilizada para calcular o peso

dos pontos relativos às amostras presente no conjunto de dados utilizados. Caso a função utilizada seja `uniform`, todos os pontos duma vizinhança terão um peso idêntico. Caso seja usada a função `distance`, os pontos que possuírem uma menor distância entre eles, terão um peso maior na previsão do que os pontos mais distantes. O hiperparâmetro `algorithm` especifica qual o algoritmo responsável por calcular os vizinhos mais próximos. O último hiperparâmetro testado, denominado de `p`, representa a potência utilizada na métrica de *Minkowski*, em que `1` corresponde à distância de *Manhattan* e `2` à distância euclidiana. Na tabela 9 listam-se os valores utilizados.

Tabela 9: Parâmetros testados com o algoritmo K-NN.

Parâmetro	Valores
n_neighbors	3, 4, 5, 6, 7
weights	*'uniform', 'distance'*
algorithm	*'ball_tree', 'kd_tree', 'brute'*
p	1, 2

Relativamente ao **algoritmo SVM**, como é possível observar na tabela 10, e tal como acontece no algoritmo *Random Forest*, são considerados seis hiperparâmetros: `C`, `kernel`, `degree`, `gamma` e `max_iter`. O hiperparâmetro `C` tem a mesma finalidade do que na regressão logística. Na otimização deste algoritmo testam-se vários *kernels*, pois dada a natureza do problema é possível verificar que não se trata de um caso de resolução linear, podendo ser necessário elevar os dados a uma maior dimensão de maneira a encontrar uma fronteira que os separe. Para o caso de otimização do algoritmo com um *kernel* do tipo `poly` ou polinomial, define-se também um conjunto de valores para teste no hiperparâmetro `degree`, que possui o grau do polinómio calculado. Posteriormente, definem-se o valor de `gamma`, que define o grau de influência que uma amostra possui na definição do hiperplano, ou seja, se o valor de `gamma` for grande o algoritmo apenas considerará os pontos mais próximos da fronteira. Ao utilizar um valor de `gamma` menor, o algoritmo terá em conta pontos mais distantes. Por último, o hiperparâmetro `max_iter` define o limite de iterações que algoritmo executará.

Tabela 10: Parâmetros testados com o algoritmo *kernel* SVM.

Parâmetro	Valores
C	0.1, 1, 10
kernel	'poly', 'rbf', 'sigmoid'
degree	3, 4, 5
gamma	0.1, 0.5, 1
max_iter	1000, 10000, 100000

No algoritmo **Árvore de Decisão** foram considerados cinco parâmetros: `criterion`, `max_depth`, `min_samples_split`, `min_samples_leaf` e `max_features`. O parâmetro `criterion` especifica a função que avalia a qualidade da separação dos nós, tendo em conta o ganho de informação (`gini`) ou o grau de impureza (`entropy`). `max_depth` define a profundidade máxima que as árvores de decisão podem ter, `min_samples_split` representa o número mínimo de amostras para separar um nó interno e `min_samples_leaf` corresponde ao número mínimo de amostras para que o nó seja considerado um nó folha. Por fim, `max_features` seleciona o número máximo de atributos a considerar na separação dos nós. A tabela 11 apresenta os valores atribuídos a cada um destes parâmetros.

Tabela 11: Parâmetros testados com o algoritmo Árvore de Decisão.

Parâmetro	Valores
criterion	*'gini', 'entropy'*
max_depth	4, 5, 6, 7
min_samples_split	2, 3, 4, 5
min_samples_leaf	1, 2, 3, 4
max_features	*'auto', 'sqrt', 'log2'*

No algoritmo ***Random Forest*** foram avaliados seis hiperparâmetros. Cinco destes hiperparâmetros possuem a mesma finalidade e designação do que nas árvores de decisão. O único parâmetro diferente é `n_estimators`, o qual corresponde ao número de árvores de decisão presentes na floresta. Na tabela 12 enumeram-se os valores considerados durante a otimização deste algoritmo.

Tabela 12: Parâmetros testados com o algoritmo *Random Forest*.

Parâmetro	Valores
n_estimators	500, 1000, 1500
criterion	`'gini', 'entropy'`
max_depth	4, 5, 6
min_samples_split	2, 3, 4
min_samples_leaf	1, 2, 3, 4
max_features	*'auto', 'sqrt', 'log2'*

No **algoritmo *XGBoost*** foram avaliados sete hiperparâmetros, em que os parâmetros `n_estimators` e `max_depth` possuem a mesma funcionalidade do que nos algoritmos das árvores de decisão e *Random Forest*. O parâmetro `min_child_split` corresponde ao mínimo da soma do peso dos nós filho, em que caso a soma seja inferior ao valor deste parâmetro, o algoritmo termina a sua execução e não procede a mais nenhuma separação de nós. A função `gamma` no algoritmo *XGBoost* define a redução mínima de perda necessária para que

exista uma nova separação do nó atual. Quanto maior for o valor de `gamma` mais conservador é o algoritmo. O parâmetro `sub_sample` representa o rácio dos dados de treino retirados antes da construção das árvores, de modo a prevenir o *overfitting*. Enquanto que o parâmetro `sub_sample` define a fração de dados de treino a utilizar na construção de uma árvore. Por seu lado, o parâmetro `colsample_bytree` representa a fração de colunas, ou atributos do conjunto de dados, a utilizar no treino da árvore. Por fim, o parâmetro `learning_rate`, ou `eta`, define o decaimento dos pesos de cada atributo do conjunto de dados a cada iteração, de maneira a tornar o algoritmo cada vez mais conservador. Na tabela 13 encontram-se todos os valores atribuídos a cada um dos parâmetros durante a otimização do algoritmo *XGBoost*.

Tabela 13: Parâmetros testados com o algoritmo *XGBoost*.

Parâmetro	Valores
n_estimators	*400, 500, 600*
max_depth	4, 5, 6
min_child_weight	0.1, 1, 5
gamma	0.1, 1, 1.5
sub_sample	0.6, 0.8, 1.0
colsample_bytree	0.6, 0.8, 1.0
learning_rate	0.01, 0.1, 1

Outro algoritmo de *boosting* analisado foi o ***Adaboost***, para o qual foram considerados três hiperparâmetros, uma vez que os restantes pertencem ao `base_estimator`, em que neste caso o classificador base é a árvore de decisão. Os parâmetros `learning_rate` e `n_estimators` coincidem com os parâmetros do mesmo nome no algoritmo *XGBoost*. O parâmetro `algorithm` representa o algoritmo utilizado para efetuar o *boosting*. A tabela 14 apresentada todos os valores testados.

Tabela 14: Parâmetros testados com o algoritmo *Adaboost*.

Parâmetro	Valores
learning_rate	0.01, 0.1, 0.5, 1, 10
n_estimators	500, 1000, 1500, 2000
algorithm	'SAMME', 'SAMME.R'

No método ***Stacking Classifier***, ou ***Voting Classifier***, a abordagem seguida no tratamento de dados é exatamente a mesma que foi utilizada em todos os outros algoritmos. No entanto, enquanto que nos algoritmos anteriormente apresentados se aplica a função `GridSearchCV` para encontrar os parâmetros que melhor se adequam ao problema, este classificador é uma fusão dos classificadores que obtiveram os melhores resultados. No entanto, não se deve incluir neste classificador algoritmos que sejam idênticos ou que produzam o mesmo tipo

de erros, pois apesar de se aumentar a necessidade de recursos computacionais, dai não resulta qualquer vantagem adicional em usar este tipo de classificador. Para o classificador em questão, utilizaram-se os algoritmos *Random Forest*, SVM com *kernel* Gaussiano e K-NN.

Tal como descrito na seção 2.3 relativamente ao algoritmo ***Pairwise Classifier***, para cada par de classes existentes no conjunto de dados utilizado é necessário treinar um classificador. Os pares de classes existentes são *Dry-Ice*, *Dry-Wet*, *Dry-Snow*, *Wet-Snow*, *Wet-Ice* e *Ice-Snow*. Para que isto seja possível, para cada um dos pares identificados, extraiu-se do conjunto de dados original uma cópia apenas com as amostras representativas desse par, ou seja, para o caso do classificador *Dry-Wet* recolhem-se as amostras em que o piso se encontra seco ou molhado. Na listagem 3.21 encontra-se o código utilizado nesta tarefa.

```
road_surface_meteo_dry_ice = wavelength_data.loc[(wavelength_data["Road_Surface_Conditions"] ==
    "Dry") | (wavelength_data["Road_Surface_Conditions"] == "Ice")].copy()
road_surface_meteo_dry_wet = wavelength_data.loc[(wavelength_data["Road_Surface_Conditions"] ==
    "Dry") | (wavelength_data["Road_Surface_Conditions"] == "Wet")].copy()
road_surface_meteo_dry_snow = wavelength_data.loc[(wavelength_data["Road_Surface_Conditions"] ==
     "Dry") | (wavelength_data["Road_Surface_Conditions"] == "Snow")].copy()
road_surface_meteo_wet_snow = wavelength_data.loc[(wavelength_data["Road_Surface_Conditions"] ==
     "Wet") | (wavelength_data["Road_Surface_Conditions"] == "Snow")].copy()
road_surface_meteo_wet_ice = wavelength_data.loc[(wavelength_data["Road_Surface_Conditions"] ==
    "Wet") | (wavelength_data["Road_Surface_Conditions"] == "Ice")].copy()
road_surface_meteo_ice_snow = wavelength_data.loc[(wavelength_data["Road_Surface_Conditions"] ==
     "Ice") | (wavelength_data["Road_Surface_Conditions"] == "Snow")].copy()
```

Listagem 3.21: Extração das amostras para os pares de classes no algoritmo *Pairwise Classifier*.

Uma vez extraídas as cópias das amostras do conjunto de dados original, a estes novos sub-conjuntos de dados aplica-se o *pipeline* utilizado nos restantes algoritmos, ou seja, a normalização das variáveis numéricas e a codificação das variáveis categóricas. Após este passo, treinaram-se seis SVM lineares, pois a complexidade de procura de um limite que separe as classes diminui significativamente com a presença de apenas duas classes em vez das quatro globais. Para além disto, a aplicação de *kernels* neste caso em concreto provocou *overfitting*. No final, agregaram-se os resultados de cada um dos classificadores, para que fosse possível obter a classe mais votada pelos mesmos. Os resultados são agrupados num novo *dataframe*, onde para cada amostra testada se recolheu a classe obtida pelo classificador, em conjunto com a probabilidade de certeza da classificação. No excerto de código representado na listagem 3.22, numa primeira fase retira-se o total de classificações obtidas e as classes presentes em cada um dos classificadores presentes.

```
# Get the total number of predictions for each classifier
# Same process for Dry_Wet, Dry_Snow, Wet_Snow, Wet_Ice and Ice_Snow
indexes_argmax_dry_ice = np.argmax(y_pred_dry_ice, axis=1)
# Get the classes of each classifier
# Same process for Dry_Wet, Dry_Snow, Wet_Snow, Wet_Ice and Ice_Snow
```

```
dry_ice_classes = svm_dry_ice.classes_
dry_ice, dry_wet, dry_snow, wet_snow, wet_ice, ice_snow = [], [], [], [], [], []

# Tuple (class, probability) creation
for i in range(len(y_val)):

  dry_ice_class = dry_ice_classes[indexes_argmax_dry_ice[i]]; dry_ice_prob = y_pred_dry_ice[i,
    indexes_argmax_dry_ice[i]]; dry_ice_tuple = (dry_ice_class, dry_ice_prob)
  dry_wet_class = dry_wet_classes[indexes_argmax_dry_wet[i]]; dry_wet_prob = y_pred_dry_wet[i,
    indexes_argmax_dry_wet[i]]; dry_wet_tuple = (dry_wet_class, dry_wet_prob)
  dry_snow_class = dry_snow_classes[indexes_argmax_dry_snow[i]]; dry_snow_prob = y_pred_dry_snow
    [i, indexes_argmax_dry_snow[i]]; dry_snow_tuple = (dry_snow_class, dry_snow_prob)
  wet_snow_class = wet_snow_classes[indexes_argmax_wet_snow[i]]; wet_snow_prob = y_pred_wet_snow
    [i, indexes_argmax_wet_snow[i]]; wet_snow_tuple = (wet_snow_class, wet_snow_prob)
  wet_ice_class = wet_ice_classes[indexes_argmax_wet_ice[i]]; wet_ice_prob = y_pred_wet_ice[i,
    indexes_argmax_wet_ice[i]]; wet_ice_tuple = (wet_ice_class, wet_ice_prob)
  ice_snow_class = ice_snow_classes[indexes_argmax_ice_snow[i]]; ice_snow_prob = y_pred_ice_snow
    [i, indexes_argmax_ice_snow[i]]; ice_snow_tuple = (ice_snow_class, ice_snow_prob)

  dry_ice.append(dry_ice_tuple); dry_wet.append(dry_wet_tuple); dry_snow.append(dry_snow_tuple)
  wet_snow.append(wet_snow_tuple); wet_ice.append(wet_ice_tuple); ice_snow.append(ice_snow_tuple
    )

data = np.array([dry_ice, dry_wet, dry_snow, wet_snow, wet_ice, ice_snow, y_val])
# Creation of the final dataset that includes all the tuple from all classifiers
meta_dataset = pd.DataFrame({'Dry_Ice': data[0, :], 'Dry_Wet': data[1, :], 'Dry_Snow': data[2,
    :], 'Wet_Snow': data[3, :], 'Wet_Ice': data[4, :], 'Ice_Snow': data[5, :], 'y_val': data[6,
    :]})
```

Listagem 3.22: Agregação dos resultados obtidos pelos classificadores no *Pairwise Classifier*.

Depois desta fase itera-se sobre as previsões de cada classificador de modo a retirar a classe obtida e a probabilidade de certeza da classificação. Isto permite agrupar estes dois valores num tuplo. Quando obtidos todos os tuplos, estes são agregados em *arrays*, que representam as colunas utilizadas para a criação do *dataframe* que armazena todos os resultados. Na tabela 15 está representado o fragmento do *dataframe* final, o qual contém todos os resultados agrupados por classificador.

Tabela 15: Agregação dos resultados no *Pairwise Classifier*.

Dry_Ice	Dry_Wet	Dry_Snow	Wet_Snow	Wet_Ice	Ice_Snow
(Dry, 0.987)	(Dry, 0.999)	(Dry, 0.973)	(Wet, 0.536)	(Ice, 0.625)	(Ice, 0.824)
(Ice, 0.999)	(Wet, 0.999)	(Snow, 0.980)	(Wet, 0.999)	(Wet, 0.900)	(Snow, 0.922)
(Dry, 0.992)	(Dry, 0.999)	(Dry, 0.973)	(Snow, 0.999)	(Ice, 0.996)	(Ice, 0.824)
(Dry, 0.997)	(Dry, 0.999)	(Dry, 0.973)	(Snow, 0.978)	(Ice, 0.894)	(Ice, 0.824)
(Dry, 0.999)	(Dry, 0.999)	(Dry, 0.973)	(Wet, 0.553)	(Wet, 0.574)	(Ice, 0.824)

Segundo Jelonek and Stefanowski (1998), uma maneira de obter a classe mais votada pelos classificadores passa por uma simples soma de qual foi a classe mais vezes obtida. Outra forma consiste em aplicar um *threeshold* às probabilidades de classificação presentes nos tuplos, ou seja, se a probabilidade não for igual ou superior a um determinado *threeshold* este não conta para a soma. Neste caso são implementados estes dois últimos métodos. O código representado na listagem 3.23 tem por objetivo obter a classe mais votada por cada classificador, para cada uma das amostras. Aqui, realiza-se o somatório das ocorrências de cada uma das classes e é devolvida a mais votada. Em caso de empate, é selecionada a classe que está em maioria no conjunto de dados original.

```
def get_more_voted_class(dry_ice, dry_wet, dry_snow, wet_snow, wet_ice, ice_snow):
  dic_votes = {'Dry': 0, 'Wet': 0, 'Ice': 0, 'Snow': 0}

  dry_ice_vote = dry_ice[0]; dry_wet_vote = dry_wet[0]; dry_snow_vote = dry_snow[0]
  wet_snow_vote = wet_snow[0]; wet_ice_vote = wet_ice[0]; ice_snow_vote = ice_snow[0];

  dic_votes[dry_ice_vote] += 1; dic_votes[dry_wet_vote] += 1; dic_votes[dry_snow_vote] += 1;
  dic_votes[wet_snow_vote] += 1; dic_votes[wet_ice_vote] += 1; dic_votes[ice_snow_vote] += 1;

  return max(dic_votes, key=dic_votes.get)
```

Listagem 3.23: Método para obter a classe mais votada no *Pairwise Classifier*.

Na listagem 3.24 o processo de seleção da classe mais votada pelos classificadores é o mesmo que se apresentou na listagem 3.23, apenas com a presença da validação perante um *threeshold*, que neste caso foi definido como `0.80`. Ou seja, um classificador terá de possuir uma probabilidade na classificação igual ou superior a `80%` para que seja válido o somatório desta ocorrência.

```
def get_more_voted_class_sum_th(dry_ice, dry_wet, dry_snow, wet_snow, wet_ice, ice_snow):
  # initialization of the dictionaries
  dic_votes = {'Dry': 0, 'Wet': 0, 'Ice': 0, 'Snow': 0}
  # initialization of the threeshold
  threeshold = 0.80
  # get predicted class from each base classifier
  dry_ice_vote = dry_ice[0]; dry_wet_vote = dry_wet[0]; dry_snow_vote = dry_snow[0]
  wet_snow_vote = wet_snow[0]; wet_ice_vote = wet_ice[0]; ice_snow_vote = ice_snow[0];
  # validation with the probabilitie of the base classifier
  if dry_ice[1] >= threeshold:
    dic_votes[dry_ice_vote] += 1
  if dry_wet[1] >= threeshold:
    dic_votes[dry_wet_vote] += 1
  if dry_snow[1] >= threeshold:
    dic_votes[dry_snow_vote] += 1
  if wet_snow[1] >= threeshold:
    dic_votes[wet_snow_vote] += 1
  if wet_ice[1] >= threeshold:
```

```
19      dic_votes[wet_ice_vote] += 1
20    if ice_snow[1] >= threeshold:
21      dic_votes[ice_snow_vote] += 1
22    return max(dic_votes, key=dic_votes.get)
```

Listagem 3.24: Método para obter a classe mais votada com *threeshold* no *Pairwise Classifier*.

Por último, foi desenvolvida uma **ANN** com o recurso à API `Keras` do `TensorFlow`. O tratamento aplicado aos dados é o mesmo que foi utilizado com os algoritmos anteriormente descritos, em que a única diferença consiste na codificação da variável dependente. Na tabela 16 encontram-se as redes neuronais treinadas com o conjunto de dados descrito. Todas as redes possuem doze nós de entrada, que correspondem ao número exato de variáveis independentes utilizados no treino. Como o nosso problema tem quatro classes alvo, o número de nós de saída para cada uma das redes também é quatro. A diferença entre as três ANN treinadas, diz respeito ao número de camadas escondidas incluídas em cada uma delas. Para cada uma das redes, a primeira camada possui dez nós e, a cada camada que é adicionada à rede, decrementa-se dois ao número de nós. Contudo, a inclusão de mais camadas provocou um aumento do *overfitting*.

Tabela 16: Redes neuronais utilizadas e respetivos parâmetros.

	ANN 1	**ANN 2**	**ANN 3**
Input Layer	activation: 'relu' input_dim=12 kernel_initializer='uniform' units=12	activation: 'relu' input_dim=12 kernel_initializer='uniform' units=12	activation: 'relu' input_dim=12 kernel_initializer='uniform' units=12
Hidden Layer 1	activation: 'relu' kernel_initializer='uniform' units=10	activation: 'relu' kernel_initializer='uniform' units=10	activation: 'relu' kernel_initializer='uniform' units=10
Hidden Layer 2		activation: 'relu' kernel_initializer='uniform' units=8	activation: 'relu' kernel_initializer='uniform' units=8
Hidden Layer 3			activation: 'relu' kernel_initializer='uniform' units=6
Output Layer	activation: 'softmax' kernel_initializer='uniform' units=4	activation: 'softmax' kernel_initializer='uniform' units=4	activation: 'softmax' kernel_initializer='uniform' units=4

4

DISCUSSÃO DE RESULTADOS

O presente capítulo divide-se em duas partes: (i) análise da otimização dos algoritmos de classificação e (ii) execução dos algoritmos no dispositivo *edge*. Na primeira seção realiza-se uma análise e comparação dos resultados obtidos em cada classificador, utilizando fusão sensorial e rácios dos sinais infravermelhos, já com os hiperparâmetros dos algoritmos otimizados. O objetivo central é perceber se a introdução das variáveis meteorológicas melhora o desempenho de classificação dos vários tipos de piso. Na segunda seção executam-se os classificadores no dispositivo *edge*, para inferir o comportamento dos mesmos num cenário próximo do real. Para além disto, procura-se averiguar se o modelo é capaz de continuar o seu normal funcionamento de classificação de amostras, perante a falha parcial ou total do sensor de condição do piso.

4.1 ANÁLISE DA APLICAÇÃO DOS ALGORITMOS DE CLASSIFICAÇÃO

Na seção 3.6 apresentaram-se os algoritmos que foram aplicados ao conjunto de dados obtido com a fusão das variáveis meteorológicas e os rácios dos comprimentos de onda. Com o objetivo de verificar se a introdução dos dados meteorológicos na classificação da condição do piso apresenta vantagens, os mesmos algoritmos foram aplicados ao conjunto de dados contendo apenas os rácios dos sinais infravermelhos. Contudo, para cada um dos dois casos foi realizada a otimização de hiperparâmetros, de maneira a ser possível realizar uma comparação justa. Em todas as próximas figuras, onde se que apresentam as matrizes de confusão obtidas, à esquerda encontra-se a matriz relativa ao conjunto de dados proveniente de fusão sensorial e à direita a matriz relativa aos rácios dos sinais infravermelhos. Tal como já foi referido anteriormente, o número total de amostras dos rácios dos comprimentos de onda não está na mesma proporção que os dados de fusão e as primeiras não incluem exemplos de piso com neve. Estes aspetos influenciam os resultados obtidos.

Dada a sensibilidade do problema em questão, classificar a condição do piso onde um veículo circula, o classificador a implementar necessita de ter um elevado *recall* em condições adversas. Neste caso, entendem-se condições adversas como piso com gelo, neve ou molhado, que é quando o veículo ajusta as definições do carro às características presentes. Contudo,

não necessita de um *recall* tão grande nas condições de piso seco, uma vez que um veículo não irá demonstrar uma condução errática caso este circule com definições de piso com chuva nestas condições.

4.1.1 *Regressão Logística*

Na figura 19 verifica-se que o classificador relativo à fusão de dados apresenta uma elevada taxa de amostras de piso com gelo classificadas como piso molhado, concretamente `105` em `528`. Neste caso, em aproximadamente `20%` dos casos o piso com gelo é classificado como molhado, o que em determinadas situações se torna numa situação adversa, pois o veículo terá de redobrar as precauções e não irá fazê-lo.

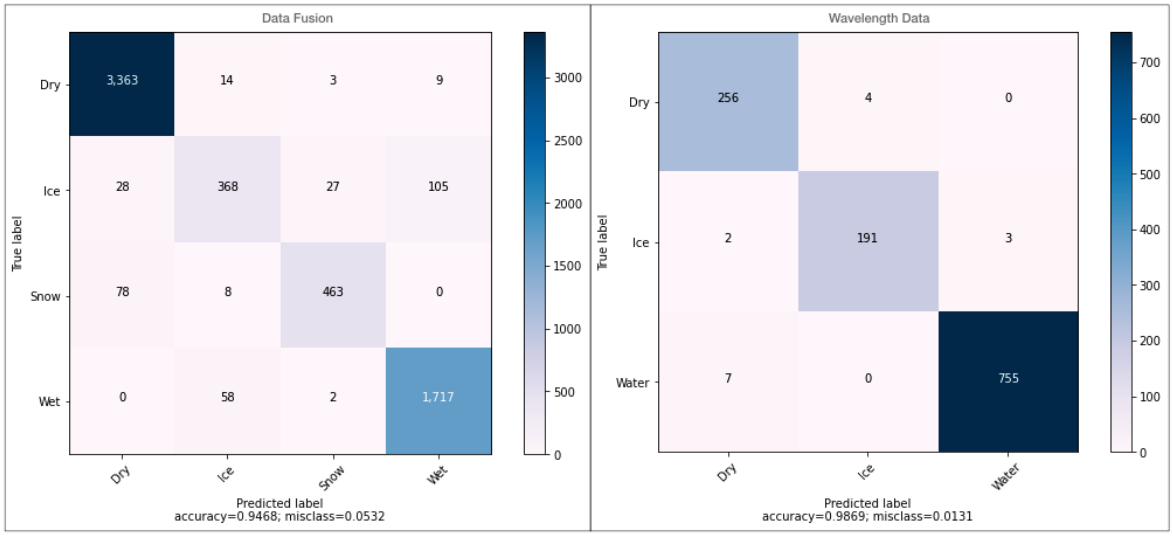


Figura 19: Matrizes de confusão da Regressão Logística.

A tabela 17 apresenta os melhores parâmetros obtidos com a regressão logística, quando aplicada na classificação (i) dos dados obtidos com a fusão entre as variáveis meteorológicas e os rácios dos sinais infravermelhos e (ii) apenas os rácios dos sinais. A principal diferença entre os dois classificadores reside não só na norma de penalização, mas também na taxa de regularização utilizada. O baixo valor de `C` pertencente ao classificador do conjunto de dados dos rácios de sinais infravermelhos, pode ser explicado pelo fato de este possuir poucas amostras quando comparado com o obtido a partir de fusão sensorial. Este último necessita de um maior valor de `C`, ou seja, uma menor regularização. Pode também verificar-se que o classificador dos rácios possui uma maior precisão e um maior *recall* na classificação do piso com gelo, em comparação com o outro modelo. Nos restantes casos, os dois classificadores demonstram-se idênticos. De notar que o classificador da fusão de dados demonstra uma boa precisão e bom *recall* na deteção de piso com neve.

Tabela 17: Resultados de Regressão Logística.

	Rácios dos sinais infravermelhos	**Fusão Sensorial**
Melhores Parâmetros	C: 10, fit_intercept: 'true', max_iter: 100, multi_class: 'auto', penalty: 'l1', solver: 'liblinear', tol: 0.0001	C: 1, fit_intercept: 'true', max_iter: 100, multi_class: 'auto', penalty: 'l2', solver: 'liblinear', tol: 0.0001
Precisão	Dry: 0.95 Ice: 0.99 Water: 0.98	Dry: 0.97 Ice: 0.82 Wet: 0.94 Snow: 0.94
Recall	Dry: 1.00 Ice: 0.91 Water: 0.98	Dry: 0.99 Ice: 0.70 Wet: 0.97 Snow: 0.84
Acurácia	0.98	0.95

4.1.2 *K-Nearest Neighbours*

As matrizes de confusão que resultam de aplicar o algoritmo de classificação K-NN encontram-se representadas na figura 20. Pode verificar-se que, nos casos de piso molhado, o algoritmo treinado a partir dos dados de fusão sensorial tem melhor comportamento que o algoritmo treinado apenas a partir dos rácios. Apesar de serem poucos os casos, este último classificou erradamente amostras de piso molhado como sendo de piso seco, ao passo que o outro modelo apenas confundiu amostras de piso molhado com piso com gelo. Classificar erradamente piso molhado como piso seco leva a tomadas de decisão erradas, pois o veículo não tomará as precauções necessárias. No entanto, o modelo baseado nos rácios é o melhor a classificar as amostras de piso seco.

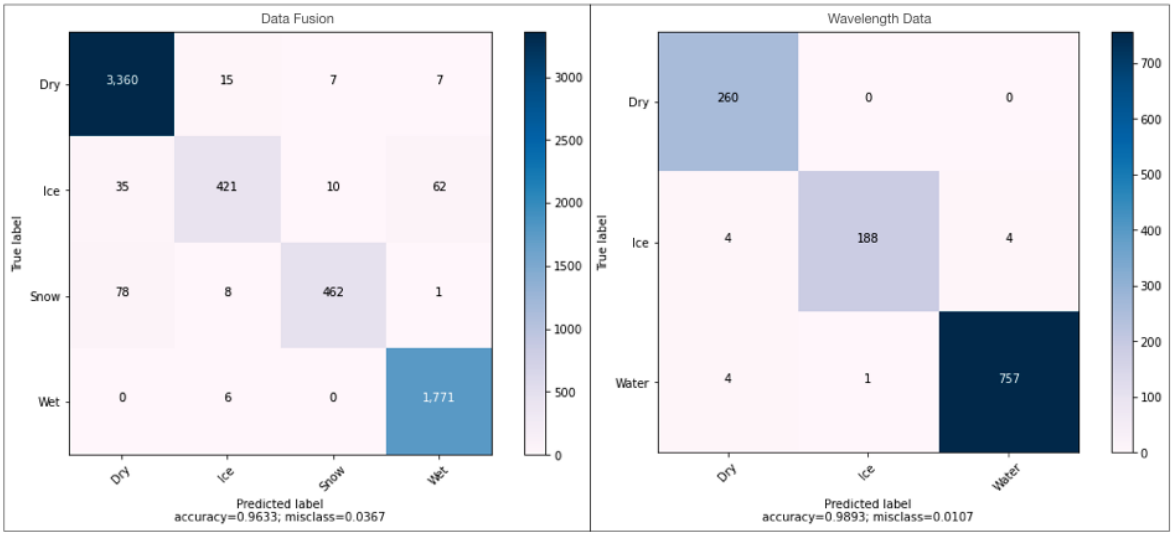


Figura 20: Matrizes de confusão do classificador *K-NN*.

As conclusões retiradas da análise das matrizes de confusão encontram-se mais detalhadas na tabela 18, onde se nota uma grande diferença nos melhores parâmetros dos dois modelos. Ambos os modelos têm como melhor algoritmo de computação dos vizinhos mais próximos o `ball_tree`. O número de vizinhos utilizado difere bastante entre os dois. Enquanto que no caso dos dados dos rácios se utiliza o valor `3`, com os dados de fusão sensorial utiliza-se `6`. As razões que explicam isto são (i) a diferença no número de classes que se pretende classificar nos dois conjuntos de dados, quatro nos dados de fusão sensorial e três nos dados dos rácios, e (ii) o reduzido número de amostras nos dados dos rácios, que consequentemente pode ter influência no grau de separação das amostras de cada uma das classes. A segunda razão apresentada pode também ser utilizada para justificar a função que define os pesos das amostras na classificação, em que nos dados de fusão sensorial é a `uniform` e nos dados dos rácios é atribuído o valor `distance`. Embora a diferença não seja elevada, o modelo treinado a partir da fusão de dados possui um *recall* superior nos casos de piso molhado, ou seja, o modelo classificou corretamente praticamente todas as amostras que verdadeiramente corresponde a piso molhado.

Tabela 18: Resultados do classificador *K-NN*.

	Rácios dos sinais infravermelhos	**Fusão Sensorial**
Melhores Parâmetros	algorithm: 'ball_tree', n_neighbors: 3, p: 2, weights: 'distance'	algorithm: 'ball_tree', n_neighbors: 6, p: 1, weights: 'uniform'
Precisão	Dry: 0.97 Ice: 0.99 Water: 0.99	Dry: 0.97 Ice: 0.94 Wet: 0.96 Snow: 0.96
Recall	Dry: 1.00 Ice: 0.96 Water: 0.99	Dry: 0.99 Ice: 0.80 Wet: 1.00 Snow: 0.84
Acurácia	0.99	0.96

4.1.3 *Máquinas de Vetores de Suporte*

Relativamente ao algoritmo SVM, a partir das matrizes de confusão representadas na figura 21 é possível verificar que o modelo aplicado aos dados de fusão sensorial tem tendência a classificar erradamente amostras de piso com neve como sendo de piso seco, em cerca de `13%` dos casos. O mesmo acontece com uma parte das amostras de piso com gelo, que são erradamente classificadas como piso molhado. Isto pode ser justificado pelo fato de estas duas superfícies possuírem índices de reflexão idênticos. No caso de amostras de piso com neve, não é possível comparar com o modelo baseado nos rácios dos sinais infravermelhos, uma vez que não possui referências para esta condição de piso. Já em relação

a amostras de piso molhado, o modelo referente a fusão sensorial possui um comportamento bastante aceitável na classificação das mesmas.

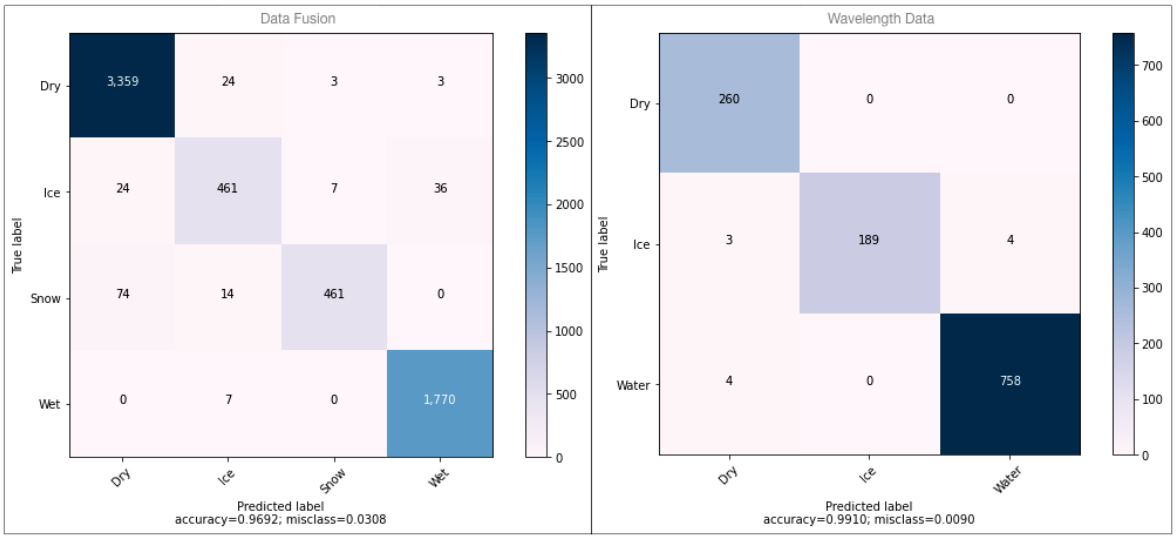


Figura 21: Matrizes de confusão do classificador *SVM*.

As conclusões descritas anteriormente, obtidas com a análise das matrizes de confusão da figura 21, encontram-se pormenorizadas na tabela 19. Verifica-se mais uma vez que, ao classificar as amostras de piso molhado, o modelo treinado com base nos dados de fusão sensorial apresenta um *recall* superior do que o modelo dos rácios dos sinais infravermelhos. A aplicação deste algoritmo melhorou ainda o *recall* obtido na classificação das amostras de piso com gelo, um caso em que até agora se tem verificado uma diferença notória entre os modelos treinados com os dois conjuntos de dados. Contudo, a acurácia dos dois modelos é quase idêntica, diferindo apenas em 1%. Relativamente aos melhores parâmetros de cada modelo, as diferenças incidem no valor de `gamma` e no número máximo de iterações. O valor deste último justifica-se dado o maior volume de dados utilizado no treino do modelo dos dados de fusão sensorial. No caso da fusão sensorial, o valor de `gamma` é menor do que no caso dos dados dos rácios dos comprimentos de onda. O elevado valor de `gamma` nos dados dos rácios pode ser justificado com a possibilidade de estes se encontrarem muito bem separados e, com isto apenas amostras que se encontrem próximas são consideradas idênticas.

Tabela 19: Resultados do classificador SVM.

	Rácios dos sinais infravermelhos	Fusão Sensorial
Melhores Parâmetros	C: 10, gamma: 1, kernel: 'rbf', max_iter: 1000	C: 10 , gamma: 0.1, kernel: 'rbf', max_iter: 10000
Precisão	Dry: 0.95 Ice: 0.98 Water: 0.99	Dry: 0.97 Ice: 0.91 Wet: 0.98 Snow: 0.98
Recall	Dry: 1.00 Ice: 0.93 Water: 0.98	Dry: 0.99 Ice: 0.87 Wet: 1.00 Snow: 0.84
Acurácia	0.98	0.97

4.1.4 *Árvore de Decisão*

Na figura 22 estão ilustradas as matrizes de confusão correspondentes ao algoritmo Árvore de Decisão. Uma característica, que se evidencia na matriz do modelo baseado nos dados de fusão sensorial, é a fraca capacidade de classificação dos casos em que o piso se encontra com gelo. Neste caso, 23% das classificações realizadas, correspondem a amostras de piso com gelo classificadas erradamente como piso seco. Ora, este aspeto é considerado grave do ponto de vista de tomada de decisão do veículo. Já relativamente ao modelo treinado com base nos rácios dos sinais infravermelhos, demonstra uma reduzida diminuição da capacidade de classificação de amostras de piso molhado, em comparação com os modelos já apresentados.

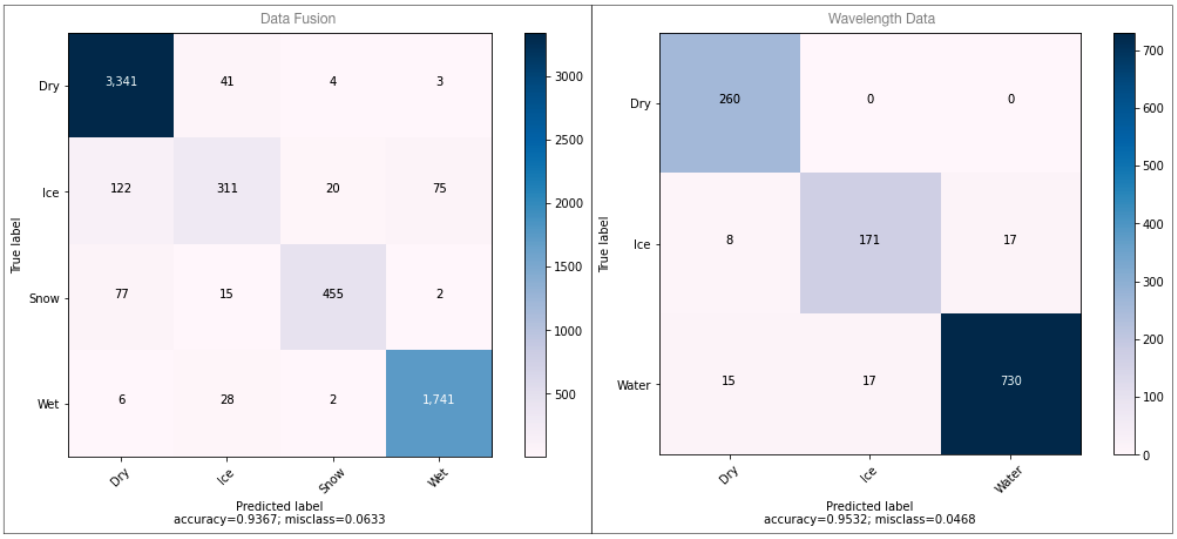


Figura 22: Matrizes de confusão do classificador Árvore de Decisão.

Na tabela 20 é possível verificar o fraco desempenho do classificador da fusão de dados perante amostras de piso com gelo, o qual possui 59% de *recall* nestes casos. No entanto, nas

amostras de piso molhado este classificador apresenta um *recall* melhor do que o classificador baseado nos rácios dos sinais infravermelhos. Apesar de este ser um aspeto positivo, o fraco comportamento face ao piso com gelo retira "credibilidade"a este classificador.

Tabela 20: Resultados do classificador Árvore de Decisão.

	Rácios dos sinais infravermelhos	**Fusão Sensorial**
Melhores Parâmetros	criterion: 'gini', max_depth: 6, max_features: 'auto', min_samples_leaf: 1, min_samples_split: 2	criterion: 'entropy', max_depth: 6, max_features: 'sqrt', min_samples_leaf: 1, min_samples_split: 5
Precisão	Dry: 0.95 Ice: 0.84 Water: 0.96	Dry: 0.95 Ice: 0.79 Wet: 0.96 Snow: 0.95
Recall	Dry: 0.94 Ice: 0.90 Water: 0.94	Dry: 0.99 Ice: 0.59 Wet: 0.98 Snow: 0.83
Acurácia	0.94	0.94

4.1.5 *Random Forest*

Comparando com o algoritmo Árvore de Decisão, a análise da figura 23 mostra que, o problema associado à classificação errada do piso com gelo como sendo piso seco, é atenuado com um *ensemble* de várias Árvores de Decisão, ou seja, com uma *Random Forest*. Contudo, continua a haver uma elevada taxa de classificações erradas deste tipo de piso. Já relativamente às amostras de piso molhado, o modelo baseado nos dados provenientes de fusão sensorial possui um melhor desempenho do que o modelo treinado com base nos dados dos rácios dos sinais infravermelhos.

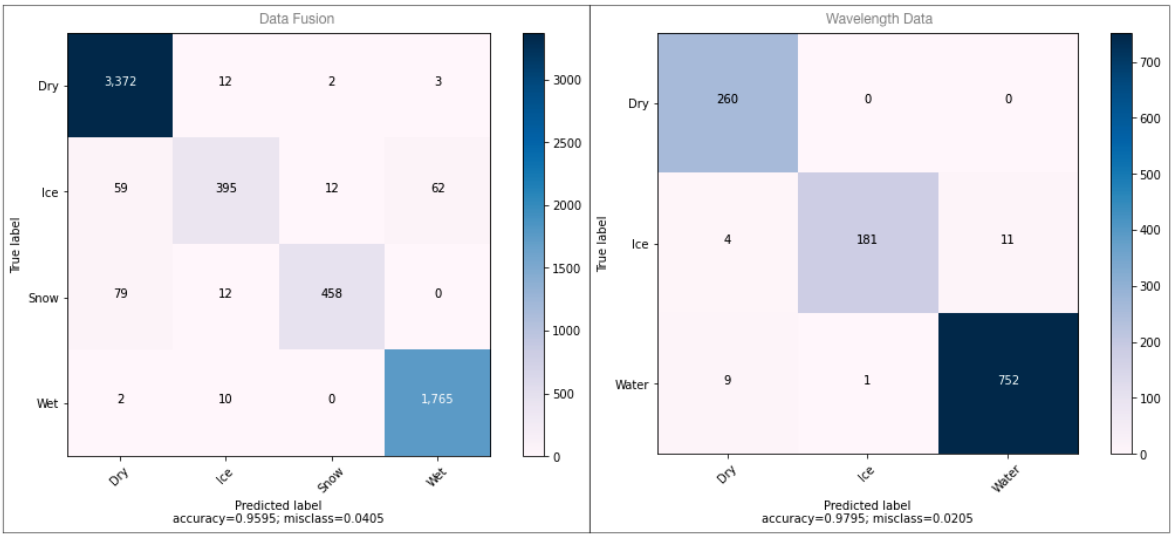


Figura 23: Matrizes de confusão do classificador *Random Forest*.

Ao analisar a tabela 21, verifica-se o que já foi concluído da visualização das matrizes de confusão da figura 23: apesar de o *ensemble* de Árvores de Decisão melhorar a proporção de classificações corretas das amostras de gelo, ainda assim apresenta uma razoável quantidade de classificações erradas. Relativamente aos melhores parâmetros obtidos para o modelo de *Random Forest*, os critérios de avaliação da qualidade de separação são diferentes para os dois conjuntos de dados. O modelo baseado nos dados de fusão sensorial dá prioridade à impuridade, enquanto que o modelo dos rácios dos comprimentos de onda dá prioridade ao ganho de informação. Outra diferença entre os dois modelos assenta no número de atributos a considerar durante a separação de um nó. No caso do modelo dos rácios, este considera o logaritmo de base 2 do número total de atributos do conjunto de dados, enquanto que o modelo baseado na fusão de dados considera a raiz quadrada do número total atributos. No entanto, os dois possuem semelhanças, tais como o número total de árvores de decisão a considerar e a profundidade máxima de cada árvore.

Tabela 21: Resultados do classificador *Random Forest*.

	Rácios dos sinais infravermelhos	**Fusão Sensorial**
Melhores Parâmetros	criterion: 'entropy', max_depth: 6, max_features: 'log2', min_samples_leaf: 1, min_samples_split: 2, n_estimators: 1000	criterion: 'gini', max_depth: 6, max_features: 'sqrt', min_samples_leaf: 1, min_samples_split: 3, n_estimators: 1000
Precisão	Dry: 0.96 Ice: 0.97 Water: 0.98	Dry: 0.96 Ice: 0.92 Wet: 0.96 Snow: 0.97
Recall	Dry: 1.00 Ice: 0.89 Water: 0.98	Dry: 0.99 Ice: 0.75 Wet: 0.99 Snow: 0.83
Acurácia	0.97	0.96

4.1.6 *XGBoost*

A análise da figura 24 revela que o desempenho do classificador *XGBoost*, na classificação de pisos com gelo, é dos que apresenta melhores resultados. Este comportamento positivo também se verifica no caso de amostras de piso molhado. No entanto, nota-se uma maior ocorrência de classificações erradas em casos de piso seco. Em relação ao modelo baseado nos dados dos rácios dos comprimentos de onda, este continua a apresentar melhores resultados na classificação de amostras de piso seco e de pisos com gelo, o qual continua ainda a ser o ponto mais sensível do classificador baseado em fusão sensorial.

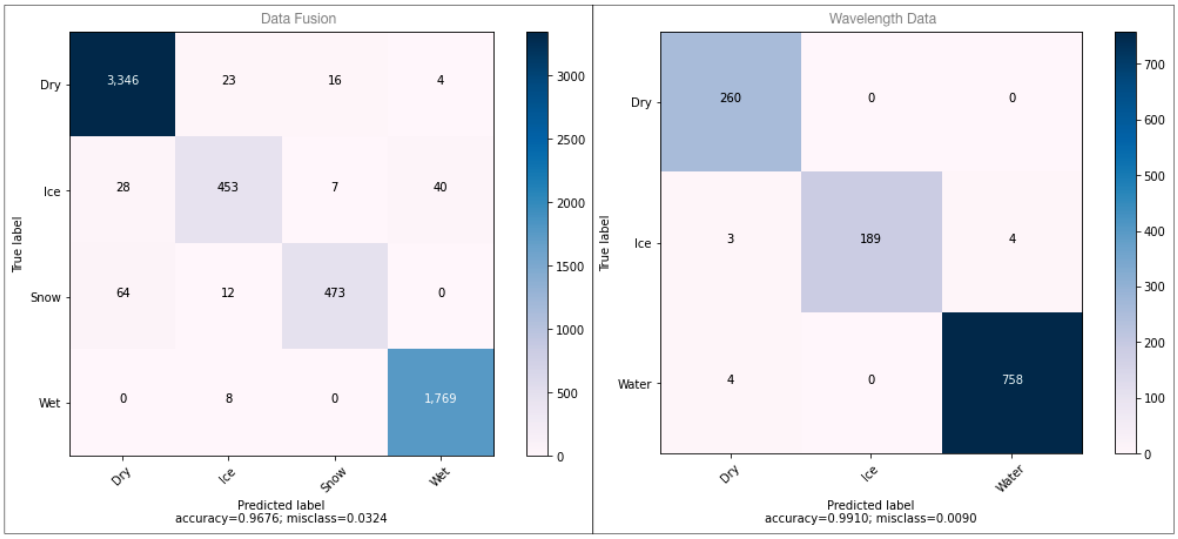


Figura 24: Matrizes de confusão do classificador *XGBoost*.

Ao melhorar a classificação correta de amostras de piso com gelo, comprovado pelo *recall* apresentado na tabela 22, o modelo *XGBoost* baseado em fusão sensorial é o melhor de todos os algoritmos descritos. Nota-se também um decréscimo, embora mínimo, do *recall* referente à classificação de amostras de piso seco. Este classificador possui também um bom comportamento na classificação de amostras de piso com neve. Relativamente aos melhores hiperparâmetros obtidos para os dois modelos, ambos possuem o mesmo número de árvores e árvores com a mesma profundidade máxima. Uma das diferenças reside no número de atributos que o algoritmo seleciona na construção de cada uma das árvores (`colsample_bytree`), em que no caso do modelo baseado nos rácios utiliza todos os atributos presentes, enquanto que no modelo de fusão sensorial apenas se utiliza `60%` dos atributos disponíveis. As restantes principais diferenças assentam na taxa de aprendizagem do algoritmo, em que no caso do modelo de fusão sensorial este parâmetro é inferior ao dos rácios, e no valor de `gamma`, que na fusão sensorial é superior aos rácios.

Tabela 22: Resultados do classificador XGBoost.

	Rácios dos sinais infravermelhos	**Fusão Sensorial**
Melhores Parâmetros	colsample_bytree: 1.0, gamma: 0.1, learning_rate: 1, max_depth: 5, min_child_weight: 5, n_estimators: 600, subsample: 0.6	colsample_bytree: 0.6, gamma: 1, learning_rate: 0.1, max_depth: 5, min_child_weight: 3, n_estimators: 600, subsample: 1.0
Precisão	Dry: 0.95 Ice: 0.99 Water: 0.99	Dry: 0.97 Ice: 0.91 Wet: 0.98 Snow: 0.95
Recall	Dry: 0.99 Ice: 0.95 Water: 0.99	Dry: 0.98 Ice: 0.88 Wet: 0.99 Snow: 0.91
Acurácia	0.98	0.97

4.1.7 *Adaboost*

Relativamente à aplicação do algoritmo *Adaboost* aos dois conjuntos de dados, as matrizes de confusão representadas na figura 25 mostram um aumento do número de classificações erradas ao nível do piso molhado, com gelo e com neve, tanto para o modelo baseado nos rácios dos sinais infravermelhos, como para o modelo baseado em fusão sensorial. Ao contrário dos modelos anteriormente descritos, o modelo *Adaboost* apresenta um elevado número de amostras de piso com neve classificadas como piso seco. Para além disto, o número de amostras de piso molhado classificadas como piso com gelo é o maior de todos os algoritmos analisados até aqui. Já o modelo baseado nos rácios dos comprimentos de onda, apresenta um ligeiro aumento no número de amostras mal classificadas de piso com gelo, assim como nas amostras de piso molhado. No geral, a par do algoritmo Árvore de Decisão, este demonstra uma menor capacidade de classificação comparativamente aos outros que foram testados, o que por consequência o coloca numa posição de ser descartado para seleção.

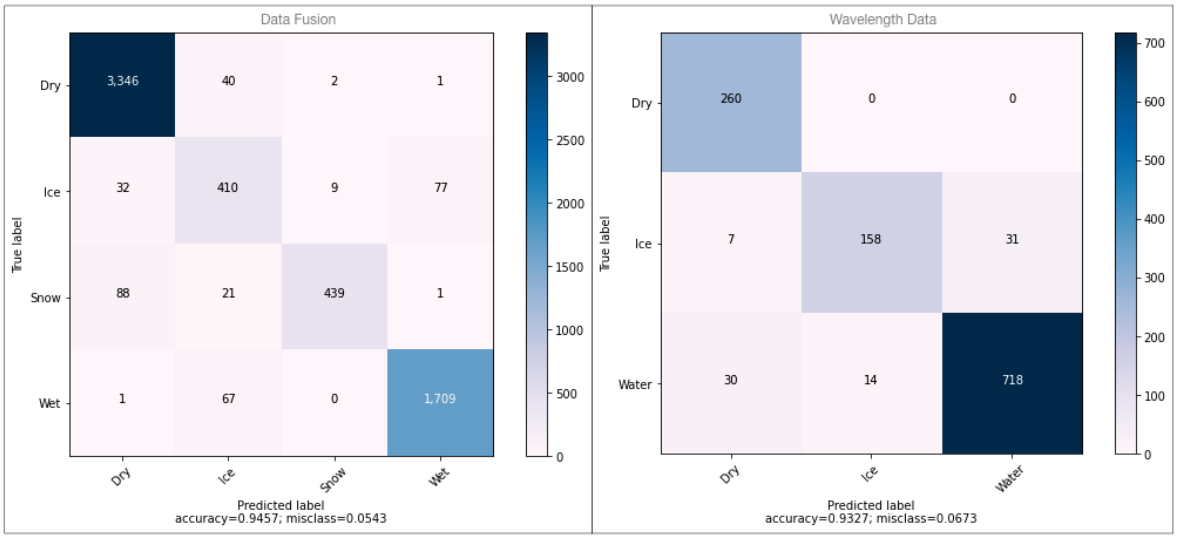


Figura 25: Matrizes de confusão do classificador *Adaboost*.

O baixo desempenho dos modelos *Adaboost* é comprovado pelos resultados incluídos na tabela 23. Verifica-se que este algoritmo não se ajusta ao conjunto de dados baseado nos rácios dos sinais infravermelhos. Enquanto que os restantes modelos, baseados neste conjunto de dados, classificam corretamente quase todas as amostras de piso com gelo. Com a aplicação de *Adaboost* o cenário inverte-se. No caso de piso com neve, acontece o mesmo no classificador baseado em fusão sensorial, em que se nota uma descida na capacidade de classificação correta, como se pode observar pelo valor de 80% do *recall*. Relativamente aos melhores parâmetros obtidos para os dois classificadores, o algoritmo e o número de árvores presentes são iguais. A taxa de aprendizagem do modelo baseado em fusão sensorial

é menor que o modelo baseado nos rácios. Isto pode derivar do fato de existirem mais amostras e mais atributos nesse conjunto de dados, o que requer uma aprendizagem mais demorada, mas o modelo de fusão sensorial obteve um desempenho superior.

Tabela 23: Resultados do classificador *Adaboost*.

	Rácios dos sinais infravermelhos	**Fusão Sensorial**
Melhores Parâmetros	algorithm: 'SAMME', learning_rate: 0.5, n_estimators: 500	algorithm: 'SAMME', learning_rate: 0.1, n_estimators: 500
Precisão	Dry: 0.89 Ice: 0.90 Water: 0.91	Dry: 0.97 Ice: 0.76 Wet: 0.96 Snow: 0.98
Recall	Dry: 0.93 Ice: 0.68 Water: 0.96	Dry: 0.99 Ice: 0.78 Wet: 0.96 Snow: 0.80
Acurácia	0.91	0.95

4.1.8 *Voting Classifier*

Nas matrizes de confusão incluídas na figura 26 verifica-se que ambos os *voting classifiers* treinados apresentam melhorias na classificação de todas as classes. No classificador baseado em fusão sensorial, a melhoria mais significativa ocorre ao nível da classificação de amostras de piso com gelo e com neve. Apesar de os outros algoritmos também apresentarem bons resultados na classificação de piso molhado, a aplicação do *Voting Classifier* melhorou ainda a capacidade de identificação desta condição do piso.

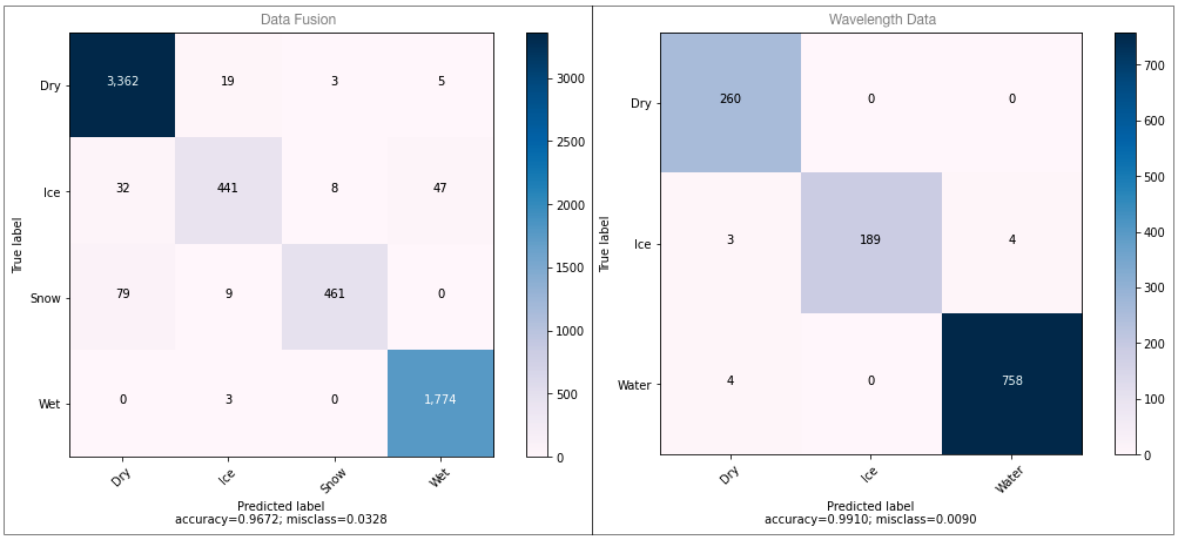


Figura 26: Matrizes de confusão do *Voting Classifier*.

Tal como descrito na seção 3.6, para aplicar o *Voting Classifier* são selecionados os algoritmos que melhores resultados obtiveram em experiências anteriores. Na tabela 24 encontram-se os algoritmos utilizados para cada um dos modelos, assim como os respetivos resultados. Pode desde já concluir-se que o agrupamento dos melhores classificadores de cada um dos modelos apresenta diferenças positivas, mas os recursos computacionais exigidos tornam a sua aplicação prática pouco atrativa, pois já foram anteriormente descritos algoritmos que obtiveram os mesmos ou melhores resultados com menos recursos. Um exemplo é o modelo *XGBoost* baseado em dados de fusão sensorial. Este modelo é o que possui os melhores resultados no que toca à classificação de pisos com neve, em que apresenta um *recall* de 91%, ou seja, deteta 91% das amostras de piso com neve, sendo 95% destas amostras classificadas como neve realmente neve. O mesmo acontece com os modelos SVM e K-NN.

Tabela 24: Resultados do *Voting Classifier*.

	Rácios dos sinais infravermelhos	**Fusão Sensorial**
Melhores Algoritmos	Random Forest, Support Vector Machine, K-Nearest Neighbors	Random Forest, Support Vector Machine, XGBoost
Precisão	Dry: 0.97 Ice: 1.00 Water: 0.99	Dry: 0.97 Ice: 0.93 Wet: 0.97 Snow: 0.98
Recall	Dry: 1.00 Ice: 0.95 Water: 0.99	Dry: 0.99 Ice: 0.84 Wet: 1.00 Snow: 0.84
Acurácia	0.99	0.97

4.1.9 *Pairwise Classifier*

Tal como foi descrito na seção 3.6, o algoritmo *Pairwise Classifier* utiliza dois métodos para votação da classe mais obtida pelos seis classificadores elaborados: somatório da classe mais vezes obtida e somatório com *threshold*. Uma vez que os modelos baseados nos rácios dos sinais infravermelhos já apresentam bons resultados, este classificador não foi aplicado a este conjunto de dados, apenas aos dados de fusão sensorial, no sentido de colmatar a sensibilidade de classificação de amostras de piso com gelo. A figura 27 mostra as matrizes de confusão resultantes da aplicação deste método. A matriz no lado esquerdo da figura diz respeito ao somatório simples e a matriz do lado direito é relativa ao somatório com *threshold*. Numa primeira análise, verifica-se que a aplicação deste classificador não introduziu melhorias na classificação, antes pelo contrário. O método de somatório simples introduziu uma ligeira melhoria na classificação de amostras com gelo. No entanto, enquanto que nos algoritmos anteriormente aplicados a classificação de piso com neve apresentava bons resultados, neste método é o que apresenta os piores resultados. Já o método de

somatório com *threshold*, para além de apresentar maus resultados para piso com neve, também apresenta maus resultados no piso com gelo.

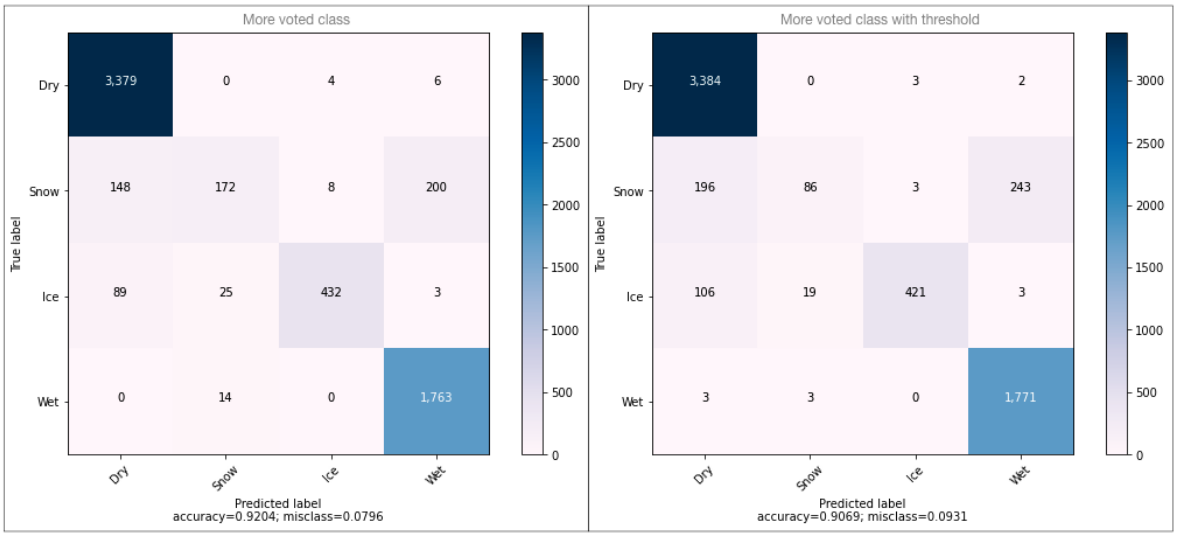


Figura 27: Matrizes de confusão do *Pairwise Classifier*.

A partir da tabela 25 é possível ter uma visão mais pormenorizada dos resultados descritos da análise das matrizes de confusão. No caso da classificação de pisos com neve, apenas 33% dessas amostras foram identificadas pelo somatório simples e 16% pelo somatório com *threshold*. Conclui-se assim que a aplicação deste modelo não é de todo viável, tendo em conta a sensibilidade do problema a resolver.

Tabela 25: Resultados do Pairwise Classifier.

	Classe mais votada	**Classe mais votada c/ threshold**
Precisão	Dry: 0.93 Ice: 0.97 Wet: 0.89 Snow: 0.82	Dry: 0.92 Ice: 0.99 Wet: 0.88 Snow: 0.80
Recall	Dry: 1.00 Ice: 0.79 Wet: 0.99 Snow: 0.33	Dry: 1.00 Ice: 0.77 Wet: 1.00 Snow: 0.16
Acurácia	0.92	0.91

4.1.10 *Rede Neuronal Artificial*

Por último, na figura 28 encontra-se a matriz de confusão resultante da classificação com uma rede neuronal artificial. Pela mesma razão apontada para o *Pairwise Classifier*, apenas se aplica a rede neuronal ao conjunto de dados proveniente de fusão sensorial, de maneira

a tentar compensar os erros cometidos pelos classificadores anteriores na classificação de amostras de piso com gelo. Verifica-se que a rede neuronal melhorou ligeiramente a capacidade de deteção do piso com gelo. No entanto, este classificador aumentou o número de amostras de piso com neve classificadas erradamente como piso seco.

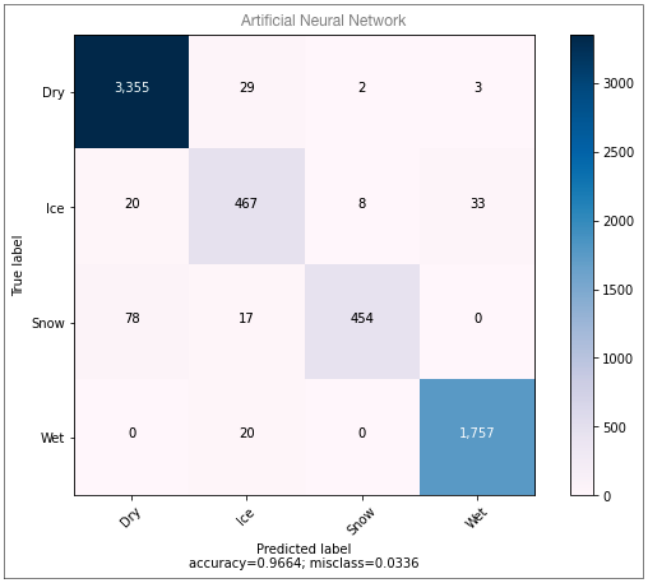


Figura 28: Matriz de confusão da classificação com *ANN*.

Ao analisar a tabela 26 pode comprovar-se o comportamento do classificador evidenciado a partir da análise da matriz de confusão. Tal como foi referido, o classificador apresenta melhorias na classificação de amostras de piso com gelo, em que deteta corretamente 88% das amostras pertencentes a este tipo de piso. Verifica-se que apesar da diminuição de desempenho na deteção de amostras com neve, este modelo situa-se entre os melhores analisados. A melhor rede neuronal obtida é a que possui apenas uma camada escondida, pois com a introdução de camadas adicionais (até três neste caso), nota-se um aumento do *overfitting* do modelo aos dados de treino.

Tabela 26: Resultados da classificação com ANN.

Melhor ANN	Input Layer: activation: 'relu', input_dim=12, kernel_initializer='uniform', units=12 Hidden Layer 1: activation: 'relu', kernel_initializer='uniform', units=10 Output Layer: activation: 'softmax', kernel_initializer='uniform', units=4
Precisão	Dry: 0.97 Ice: 0.88 Wet: 0.98 Snow: 0.98
Recall	Dry: 0.99 Ice: 0.88 Wet: 0.99 Snow: 0.83
Acurácia	0.97

4.2 IMPLEMENTAÇÃO DOS CLASSIFICADORES DE PISO NO DISPOSITIVO EDGE

Para além da análise dos resultados obtidos com os classificadores em execução num computador pessoal, bem como dos respetivos hiperparâmetros, a seleção do modelo que melhor se adequa ao problema da classificação do piso onde circula um automóvel, não se deve basear somente na precisão ou *recall*. De modo a complementar e enriquecer o processo de seleção do modelo mais adequado, decidiu-se testar os modelos em condições mais próximas das encontradas no cenário real de um automóvel.

A classificação da condição do piso é realizada em plena circulação do automóvel a diversas velocidades, pelo que o RCS estará constantemente a retirar amostras dos rácios dos sinais infravermelhos a uma taxa bastante elevada. Segundo os investigadores projeto *Sensible Car*, esta taxa ronda em média as `4` amostras por décima de segundo, o que equivale a cerca de `40` amostras a cada segundo de circulação. Com isto, em condições reais, o classificador terá de ser capaz de processar no mínimo `40` amostras por segundo, incluindo não só os rácios dos sinais infravermelhos, mas também as variáveis meteorológicas naquele local.

De modo a retirar conclusões sobre a taxa de classificações por segundo, todos os modelos de classificação foram implementados no dispositivo *edge* Nvidia Jetson Nano, em que cada modelo foi implementado de acordo com os melhores parâmetros obtidos. Os melhores parâmetros para os modelos encontram-se descritos na seção 4.1. Cada modelo terá de efetuar as classificações num conjunto de dados de validação elaborado para o efeito. Nenhuma das amostras pertencentes a este conjunto de dados foi previamente utilizada em fases de treino ou de teste. Este conjunto de dados de teste possui cerca de `6300` amostras. Na tabela 27 encontram-se os resultados obtidos os testes efetuados. O modelo de classificação que possui o pior resultado, em termos de amostras processadas por segundo, é o *Random Forest*, que efetua em média apenas `2` predições por segundo. Assim, este modelo deixa de ser um opção válida, pois não demonstra capacidade para acompanhar a taxa de amostras obtidas pelo RCS por segundo. Os modelos ANN, K-NN, *Adaboost* e *Stacking*, efetuam um número de classificações por segundo bastante idêntico entre si. No entanto, aplica-se o mesmo princípio utilizado para colocar de parte o modelo *Random Forest*, em que o número de classificações por segundo não se aproxima das `40` pretendidas como mínimo aceitável. Uma vez descartados os modelos cujos resultados não igualam nem superam o mínimo desejável, procede-se à avaliação dos que superam esse requisito: SVM Gaussiana, Regressão Logística, Árvore de Decisão, *XGBoost* e *Pairwise Classifier*. Apesar de o *Pairwise Classifier* respeitar a condição de classificações mínimas por segundo, a sua percentagem de classificações corretas é cerca de 5% inferior à dos outros modelos, o que impõe deixar de ser opção de seleção. Dos quatro restantes, destaca-se a Regressão Logística, que demonstra uma capacidade de classificação que ronda quase as `2100`. Este modelo possui também uma

percentagem de classificações corretas de aproximadamente 95%. De seguida, o modelo Árvore de Decisão tem a capacidade de realizar cerca de 1950 classificações por segundo. No entanto, a percentagem de amostras corretamente classificadas é 93%, ou seja, fica um pouco aquém dos outros modelos. Entre os modelos *XGBoost* e SVM Gaussiana, a percentagem de classificações corretas é praticamente idêntica. A principal diferença entre estes dois modelos assenta da discrepância existente entre as classificações por segundo, em que o modelo SVM possui uma capacidade quase 5 vezes superior ao modelo *XGBoost*.

Tabela 27: Classificações por segundo efetuadas pelo modelos no dispositivo *edge*.

Modelo de Classificação	**Classificações por segundo**	**Percentagem de classificações corretas**
Rede Neuronal Artificial	7	96,7%
Random Forest	2	95,9%
K-NN	8	96,3%
SVM Gaussiana	1040	96,9%
Regressão Logística	2081	94,6%
Árvore de Decisão	1950	93,3%
XGBoost	223	96,7%
Adaboost	6	94,5%
Pairwise Classifier	55	91,8%
Stacking Classifier	6	96,2%

A identificação dos modelos que são capazes de classificar a condição do piso, à taxa a que o RCS produz os rácios dos sinais infravermelhos, não é suficiente para selecionar o melhor dos quatro já identificados. Como se trata de um problema bastante sensível, porque afeta a segurança do condutor, é altamente relevante avaliar o comportamento dos modelos perante a possível falha de um ou mais feixes de infravermelhos do RCS. Deste modo, os melhores modelos até agora selecionados serão testados num cenário de falha de um, dois e três feixes de infravermelho, até à falha total dos 4 feixes. Todas as comparações serão realizadas após a fase de treino, com a introdução de valores em falta nos atributos alvo no conjunto de dados de teste. Estes casos de falha foram também replicados para o conjunto de dados que contém apenas os rácios dos sinais infravermelhos. Esta avaliação tem como objetivo perceber se os dados meteorológicos são capazes de compensar a falha parcial ou total do RCS. Nas tabelas a seguir apresentadas, realizar-se-á a comparação dos quatro melhores modelos anteriormente selecionados, para os dois conjuntos de dados utilizados (fusão sensorial e rácios dos sinais infravermelhos). Será avaliado o impacto de possíveis falhas no sistema e será avaliado se a introdução de variáveis meteorológicas no sistema é vantajosa.

4.2.1 *Comportamento do Modelo SVM perante Falhas nos Feixes de Infravermelhos*

Na tabela 28 faz-se a comparação dos modelos SVM treinados com os dois conjuntos de dados: fusão sensorial e rácios dos sinais infravermelhos. Quando falha um feixe o modelo treinado com os rácios apresenta um elevado *recall*, mas ocorre uma ligeira diminuição na precisão nos casos de piso seco, quando comparado com o modelo de fusão sensorial. A diminuição na precisão não se verifica no piso com gelo, em que apesar do modelo dos rácios detetar menos 3% dos casos totais de gelo em relação ao modelo de fusão sensorial, apresenta uma maior precisão. Quando existe a falha de dois feixes, o modelo baseado nos rácios não apresenta nenhum impacto no seu desempenho. O mesmo não acontece no caso do modelo de fusão sensorial, que apresenta uma ligeira diminuição no desempenho de classificação de amostras de piso com gelo. Para o caso de falha de três feixes, o modelo baseado nos rácios demonstra um agravamento no seu funcionamento. No caso das amostras de piso com gelo, o modelo apenas deteta no total 11%. Aqui, verifica-se também a diminuição do total de amostras de piso seco e molhado corretamente identificadas, bem como a diminuição da sua precisão. Relativamente ao comportamento do modelo baseado em fusão sensorial, com apenas um feixe disponível, o desempenho de classificação mantém-se para todos os tipos de piso, à exceção dos pisos com gelo em que demonstra uma maior sensibilidade. No caso de falha total, é possível verificar que o modelo baseado em fusão sensorial, que nesta fase apenas está a realizar as classificações com base nas variáveis meteorológicas, ainda que haja uma diminuição de desempenho nos casos de piso molhado e com gelo, consegue manter uma respeitável capacidade de classificação.

Tabela 28: Comparação dos classificadores SVM quando os feixes falham.

Feixes em falta	**Fusão Sensorial SVM**		**Comprimentos de onda SVM**	
	Precisão	**Recall**	**Precisão**	**Recall**
Um feixe	Dry: 0.97 Ice: 0.89 Snow: 0.97 Wet: 0.97	Dry: 0.99 Ice: 0.83 Snow: 0.84 Wet: 0.99	Dry: 0.93 Ice: 1.00 Water: 0.96	Dry: 1.00 Ice: 0.80 Water: 0.98
Dois feixes	Dry: 0.97 Ice: 0.83 Snow: 0.97 Wet: 0.95	Dry: 0.99 Ice: 0.77 Snow: 0.83 Wet: 0.98	Dry: 0.93 Ice: 1.00 Water: 0.96	Dry:1.00 Ice: 0.80 Water: 0.98
Três feixes	Dry: 0.94 Ice: 0.78 Snow: 0.95 Wet: 0.92	Dry: 0.97 Ice: 0.65 Snow: 0.83 Wet: 0.95	Dry: 0.85 Ice: 0.25 Water: 0.83	Dry: 0.89 Ice: 0.11 Water: 0.94
Todos	Dry: 0.87 Ice: 0.68 Snow: 0.87 Wet: 0.71	Dry: 0.98 Ice: 0.63 Snow: 0.82 Wet: 0.55	Falha	Falha

4.2.2 *Comportamento do Modelo Regressão Logística perante Falhas nos Feixes de Infravermelhos*

O próximo algoritmo testado sob falha dos feixes de infravermelhos foi a Regressão Logística, cujos resultados se encontram representados na tabela 29. Com três feixes disponíveis, o comportamento do classificador de Regressão Logística baseado em fusão sensorial apresenta o mesmo comportamento do classificador baseado nos rácios dos sinais infravermelhos. A diferença do impacto resultante da falha de feixes começa a ser notória quando só estão disponíveis dois feixes. Neste caso, o modelo de fusão sensorial apresenta um melhor *recall* e uma melhor precisão do que o modelo baseado nos rácios. No entanto, tal não se verifica perante a presença de piso com gelo, pois o modelo baseado em fusão sensorial apresenta uma diminuição no desempenho de classificação. Quando está disponível apenas um feixe, o classificador treinado com dados de fusão sensorial não apresenta um impacto significativo no desempenho de classificação. Este revela uma ligeira diminuição no número total de amostras corretamente classificadas para cada tipo de piso, bem como uma ligeira diminuição de precisão. Contudo, o classificador baseado nos rácios apresenta um diminuição abrupta no desempenho, nomeadamente na presença de amostras de piso com gelo. Neste caso, o modelo não é capaz de identificar qualquer amostra deste tipo e diminui ainda a precisão da classificação do piso molhado. Perante uma falha total do RCS, o modelo baseado em fusão sensorial demonstra uma diminuição da quantidade de amostras de piso molhado e de piso com gelo. Apesar de apenas detetar cerca de `60%` das amostras de piso molhado, este modelo apresenta uma precisão elevada. O mesmo não acontece para a deteção de piso com gelo, onde tanto diminui o total de amostras detetadas como a precisão.

Tabela 29: Comparação dos Classificadores de Regressão Logística quando os feixes falham.

Feixes em falta	**Fusão Sensorial Regressão Logística**		**Comprimentos de onda Regressão Logística**	
	Precisão	**Recall**	**Precisão**	**Recall**
Um feixe	Dry: 0.97 Ice: 0.80 Snow: 0.93 Wet: 0.94	Dry: 0.99 Ice: 0.70 Snow: 0.84 Wet: 0.96	Dry: 0.94 Ice: 1.00 Water: 0.95	Dry: 1.00 Ice: 0.78 Water: 0.98
Dois feixes	Dry: 0.97 Ice: 0.76 Snow: 0.94 Wet: 0.93	Dry: 0.99 Ice: 0.66 Snow: 0.84 Wet: 0.96	Dry: 0.92 Ice: 0.96 Water: 0.95	Dry:0.97 Ice: 0.78 Water: 0.98
Três feixes	Dry: 0.95 Ice: 0.74 Snow: 0.93 Wet: 0.89	Dry: 0.97 Ice: 0.61 Snow: 0.83 Wet: 0.92	Dry: 0.88 Ice: 0.00 Water: 0.78	Dry: 0.90 Ice: 0.00 Water: 0.97
Todos	Dry: 0.77 Ice: 0.67 Snow: 0.90 Wet: 0.96	Dry: 0.98 Ice: 0.49 Snow: 0.83 Wet: 0.56	Falha	Falha

4.2.3 *Comportamento do Modelo Árvore de Decisão perante Falhas nos Feixes de Infravermelhos*

A tabela 30 apresenta os resultados da comparação dos classificadores treinados com dados de fusão sensorial e com os rácios dos sinais infravermelhos, aplicando o algoritmo Árvore de Decisão. Perante a falha de um feixe de infravermelhos, o modelo baseado em fusão sensorial apresenta desde já sensibilidade na classificação de superfícies com gelo, detetando corretamente à volta de `50%` da totalidade de amostras deste tipo de piso. Em comparação com o modelo baseado nos rácios, este classificador apresenta um desempenho ligeiramente melhor na classificação de amostra de piso seco. Quando existe a falha de dois feixes, o desempenho da classificação de amostras de piso seco e piso com gelo mantém-se inalterado, apenas se verifica uma diminuição da precisão associada à identificação de amostras de piso molhado. Relativamente ao modelo baseado nos rácios, tanto para a falha de um feixe, como para a falha de dois feixes, o comportamento não se altera. Quando apenas está disponível um feixe, os impactos são visíveis em ambos os classificadores. No caso do classificador baseado nos rácios dos sinais infravermelhos, este demonstra uma ligeira diminuição no número total de amostras de piso seco e molhado detetadas corretamente. No que toca à classificação de piso com gelo, o modelo apenas é capaz de classificar corretamente cerca de `26%` do total. Relativamente ao modelo baseado em fusão sensorial, o desempenho na classificação geral mantém-se estável, registando-se apenas uma ligeira diminuição nos casos de superfícies molhadas e com neve. A diminuição mais grave ocorre, mais uma vez, na classificação de superfícies com gelo, onde se verifica uma queda de cerca de `20%` tanto na precisão, como na totalidade de amostras detetadas corretamente. Perante a falha total do sistema, o classificador baseado em fusão sensorial diminui significativamente o número total de amostras de piso molhado identificadas. Nesta situação, a precisão na classificação para as restantes amostras diminui cerca de `10%`, para os restantes tipos de piso.

Tabela 30: Comparação dos Classificadores Árvores de Decisão quando os feixes falham.

Feixes em falta	Fusão Sensorial Árvore de Decisão		Comprimentos de onda Árvore de Decisão	
	Precisão	Recall	Precisão	Recall
Um feixe	Dry: 0.95 Ice: 0.78 Snow: 0.98 Wet: 0.92	Dry: 0.99 Ice: 0.55 Snow: 0.78 Wet: 0.99	Dry: 0.93 Ice: 0.93 Water: 0.95	Dry: 0.93 Ice: 0.82 Water: 0.98
Dois feixes	Dry: 0.95 Ice: 0.81 Snow: 0.97 Wet: 0.89	Dry: 0.99 Ice: 0.47 Snow: 0.83 Wet: 0.98	Dry: 0.94 Ice: 0.97 Water: 0.96	Dry:0.98 Ice: 0.83 Water: 0.98
Três feixes	Dry: 0.95 Ice: 0.67 Snow: 0.95 Wet: 0.87	Dry: 0.94 Ice: 0.66 Snow: 0.81 Wet: 0.93	Dry: 0.89 Ice: 0.64 Water: 0.82	Dry: 0.91 Ice: 0.26 Water: 0.94
Todos	Dry: 0.80 Ice: 0.54 Snow: 0.82 Wet: 0.97	Dry: 0.95 Ice: 0.66 Snow: 0.84 Wet: 0.55	Falha	Falha

4.2.4 *Comportamento do Modelo XGBoost perante Falhas nos Feixes de Infravermelhos*

Por último, a tabela 31 mostra o comportamento dos modelos *XGBoost* quando ocorrem falhas nos feixes de infravermelhos. Caso apenas estejam disponíveis três feixes, o comportamento dos modelos treinados com dados de fusão sensorial e com os rácios é idêntico. A classificação dos pisos secos e molhados, utilizando as variáveis meteorológicas, permite que este modelo consiga classificar corretamente mais amostras deste tipo, assim como obter uma maior precisão. Isto não se verifica no caso de piso com gelo, pois apesar de o modelo dos rácios apresentar um menor *recall*, ou seja, deteta ligeiramente menos amostras que o modelo de fusão sensorial, possui uma maior precisão na classificação destes casos. Perante a falha de dois feixes, o desempenho dos dois classificadores mantém-se quase inalterado, face ao caso em que falha apenas um feixe. A única diferença verificada, consiste na diminuição do total de amostras de superfície com gelo, por parte do modelo de fusão sensorial. No caso de falha de três feixes, o impacto no classificador baseado nos rácios é elevado. Em relação à deteção de amostras de piso seco e piso molhado, existe uma ligeira descida no número total de amostras detetadas corretamente, acompanhada também por um decréscimo de quase 15% na precisão da classificação das amostras com piso molhado. Quando se trata da classificação de superfícies com gelo, tanto a precisão como o número de amostras detetadas, verifica-se uma diminuição do desempenho na ordem dos 50%. Ainda nestas condições, o modelo de fusão sensorial não exibe um impacto significativo na classificação das amostras. Para uma falha total do RCS, o modelo de fusão sensorial não apresenta alterações na classificação de superfícies com neve, em que o mesmo não acontece

para as restantes condições de piso. Na classificação de superfícies com gelo e com água, o desempenho desce praticamente para metade em relação à presença de um feixe. No entanto, enquanto que a precisão da classificação de piso com água se mantém, esta diminui 20% para a deteção de gelo. Relativamente à classificação de superfícies secas, o modelo só apresenta uma diminuição de precisão.

Tabela 31: Comparação dos Classificadores XGBoost quando os feixes falham.

Feixes em falta	Fusão Sensorial XGBoost		Comprimentos de onda XGBoost	
	Precisão	Recall	Precisão	Recall
Um feixe	Dry: 0.97 Ice: 0.88 Snow: 0.96 Wet: 0.97	Dry: 0.99 Ice: 0.84 Snow: 0.86 Wet: 0.99	Dry: 0.92 Ice: 0.99 Water: 0.95	Dry: 0.98 Ice: 0.77 Water: 0.98
Dois feixes	Dry: 0.97 Ice: 0.85 Snow: 0.93 Wet: 0.95	Dry: 0.98 Ice: 0.76 Snow: 0.86 Wet: 0.98	Dry: 0.89 Ice: 1.00 Water: 0.95	Dry:0.99 Ice: 0.79 Water: 0.97
Três feixes	Dry: 0.96 Ice: 0.80 Snow: 0.92 Wet: 0.90	Dry: 0.97 Ice: 0.70 Snow: 0.85 Wet: 0.94	Dry: 0.90 Ice: 0.61 Water: 0.81	Dry: 0.91 Ice: 0.25 Water: 0.94
Todos	Dry: 0.77 Ice: 0.60 Snow: 0.90 Wet: 0.94	Dry: 0.98 Ice: 0.37 Snow: 0.84 Wet: 0.56	Falha	Falha

4.3 SELEÇÃO DO MELHOR CLASSIFICADOR DA CONDIÇÃO DO PISO

A partir da análise da otimização dos modelos (seção 4.1) e do comportamento destes perante a falha dos feixes de infravermelhos (seção 4.2) pode identificar-se o modelo que melhor se adequa a resolver o problema da classificação do piso em que circula um automóvel. Dos quatro modelos selecionados até este momento, o que se destaca é o SVM Gaussiano. Apesar de este modelo não ser o que apresenta o maior número de classificações por segundo, é o que possui uma melhor capacidade para classificar todas as condições de piso, principalmente para identificar as superfícies com gelo, que é a superfície em que os modelos revelaram maior dificuldade de classificação. Para além disso, o modelo SVM Gaussiano é o mais robusto em caso de falha parcial ou total dos sensores de infravermelhos, assegurando um funcionamento razoável do sistema de classificação, apenas com a utilização das variáveis meteorológicas.

Durante a análise de resultados pôde constatar-se que a presença de pelo menos um feixe de infravermelhos é fundamental para o sucesso da classificação da condição do piso. Esta conclusão resulta da comparação dos resultados obtidos apenas com um feixe disponível e quando há falha total dos feixes. Foi visível a redução de quase 50% no número total de

amostras de piso molhado e de piso com gelo classificadas corretamente quando deixa de haver qualquer feixe de infravermelhos disponível ao classificador.

5

CONCLUSÃO

No presente capítulo descrevem-se as avaliações finais adquiridas do desenvolvimento desta dissertação. Numa primeira parte, expõem-se os objetivos concretizados de entre os propostos no início do trabalho. A seguir descrevem-se as limitações encontradas durante o processo de desenvolvimento, de que maneira afetaram o plano inicialmente delineado e qual a adaptação efetuada. Para além disto, inclui-se também sugestões para trabalho futuro, de maneira a enriquecer o conteúdo deste tema e a melhorar também os resultados já obtidos. Por último, faz-se uma apreciação final da dissertação e de toda estratégia seguida na sua realização.

5.1 OBJETIVOS CONCRETIZADOS

De acordo com o planeamento inicialmente definido, para além da recolha e análise da bibliografia proveniente do trabalho de pesquisa, a primeira parte consistiu na seleção do dispositivo *edge* que melhor se adequa ao problema em questão. O dispositivo selecionado foi o `Nvidia Jetson Nano`, pois além da facilidade de configuração e execução de modelos de aprendizagem automática, permite também a ligação com diversos sensores, como por exemplo sensores de humidade.

Dado que neste trabalho se pretendia utilizar um conjunto de dados proveniente de fusão sensorial e, uma vez que o conjunto de dados dos rácios dos sinais infravermelhos fornecido era de dimensão reduzida, e não continha nenhuma referência aos dados meteorológicos no momento da sua aquisição, criou-se um novo conjunto de amostras que agrega tanto dados meteorológicos como os rácios. A criação deste novo conjunto de dados consistiu em utilizar *web scrapping* e geolocalização para obter um número razoável de amostras a utilizar no treino dos algoritmos de classificação. Como já foi referido, visto que os dados dos rácios dos sinais infravermelhos não possuem referências às condições atmosféricas no momento da sua aquisição, a geração de dados sintéticos a partir dos dados naturais fornecidos foi necessária. Assim, geraram-se os rácios dos sinais infravermelhos com a mesma distribuição que os dados naturais apresentam e foram posteriormente associados às respetivas condições meteorológicas, tendo em conta a condição de piso correspondente.

Na etapa seguinte procedeu-se ao treino, teste e análise dos resultados obtidos com os vários modelos de classificação implementados. A seleção dos melhores modelos partiu da leitura das matrizes de confusão de cada um dos modelos, bem como da análise da precisão e do *recall* para cada condição de piso. Após selecionar os melhores modelos, estes foram implementados no dispositivo `Nvidia Jetson Nano`, de modo a testá-los num cenário próximo da situação real para a qual o classificador de piso foi pensado. Concluiu-se que o algoritmo SVM Gaussiana é o que resolve melhor o problema da classificação da condição do piso em que circula um automóvel.

5.2 Limitações e Trabalho Futuro

Uma das grandes limitações enfrentadas durante a fase de desenvolvimento da dissertação, foi o acesso aos dados relativos aos rácios dos sinais infravermelhos. Apesar de ter sido assinado um NDA com a `Bosch`, o tipo de acesso aos dados estava limitado a um único portátil localizado no Campus de Azurém da Universidade do Minho. Em alternativa a esta forma de acesso, foi disponibilizado um pequeno conjunto de amostras do conjunto de dados existente no projeto `Sensible Car`. Para além disto e, ao contrário do que inicialmente se pensava, este conjuntos de amostras não estavam associadas às respetivas variáveis meteorológicas no momento da medição. Isto levou a que fosse necessário a pesquisa de conjuntos de dados auxiliares, que permitissem associar a condição do piso às condições atmosféricas presentes.

A outra grande limitação, consistiu na declaração do estado de emergência nacional e consequente confinamento obrigatório devido à propagação da *COVID-19*. Esta agravante provocou um atraso de cerca de dois meses em relação ao plano inicial do trabalho, nomeadamente na fase de teste dos modelos de classificação no dispositivo *edge*.

Como trabalho futuro destacam-se quatro tópicos: introdução da temperatura da superfície do piso no conjunto de dados, ligação de sensores ao dispositivo *edge*, fusão de valores naturais dos rácios com as variáveis meteorológicas e o teste do modelo de classificação selecionado em contexto real. A utilização da temperatura da superfície da estrada permitirá ao modelo obter uma melhor distinção dos pisos com gelo, uma vez que a superfície é muitas vezes confundida com piso molhado devido ao seu índice de reflexão. A ligação de diversos sensores ao dispositivo `edge` e, caso seja possível uma ligação ao RCS, tornaria possível não só a construção de um conjunto de dados mais robusto para treino dos algoritmos, como também a fusão de valores naturais dos rácios dos sinais infravermelhos às condições atmosféricas. Por último, o teste do modelo selecionado em contexto real, permitirá obter uma melhor perceção sobre o comportamento do mesmo, para além dos testes em laboratório.

5.3 APRECIAÇÃO FINAL

O planeamento inicial foi desenvolvido de forma realista e adaptado à necessidade de aprendizagem de diversas tecnologias utilizadas. Porém, ao longo do desenvolvimento do projeto, devido à conjetura que se vive atualmente e às limitações técnicas encontradas, o planeamento sofreu alterações pontuais ao cumprimento dos objetivos. No final, com a rápida adaptação e definição de prioridades de desenvolvimento, estes entraves não se revelaram muito prejudiciais para o progresso do trabalho. A realização de reuniões semanais com o orientador da dissertação demonstrou-se crucial do ponto de vista de permitir não só a criação de uma eficaz linha de orientação para a aprendizagem da área em questão, como também a rápida correção de potenciais desvios no desenvolvimento da solução.

Todo o processo de desenvolvimento foi acompanhado da sua documentação regular, de modo a permitir uma ótima compreensão da solução obtida. Isto permite que, caso se pretenda avançar com as propostas de trabalho futuro anteriormente apresentadas, este processo se encontre facilitado do ponto de vista de retoma do trabalho de investigação e análise das mesmas.

Por fim, a realização de um trabalho num contexto atual e de elevado impacto na sociedade em dias futuros, não só motivaram a dedicação ao desenvolvimento da solução, como também possibilitaram a evolução das capacidades de análise e perceção da adequação da tecnologia a um ambiente de utilização exigente, respondendo às necessidades concretas de utilização em situações reais.

BIBLIOGRAFIA

Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, August 1996. doi: 10.1007/bf00058655. URL https://doi.org/10.1007/bf00058655.

Aleksandr Bystrov, Mohammad Abbas, Edward Hoare, Thuy-Yung Tran, Nigel Clarke, Marina Gashinova, and Mikhail Cherniakov. Analysis of classification algorithms applied to road surface recognition. In *2015 IEEE Radar Conference (RadarCon)*, pages 0907–0911. IEEE, may 2015. ISBN 978-1-4799-8232-5. doi: 10.1109/RADAR.2015.7131124. URL http://ieeexplore.ieee.org/document/7131124/.

Aleksandr Bystrov, Edward Hoare, Thuy-Yung Tran, Nigel Clarke, Marina Gashinova, and Mikhail Cherniakov. Road Surface Classification Using Automotive Ultrasonic Sensor. *Procedia Engineering*, 168:19–22, 2016. ISSN 18777058. doi: 10.1016/j.proeng.2016.11.119. URL https://linkinghub.elsevier.com/retrieve/pii/S1877705816334282.

Aleksandr Bystrov, Edward Hoare, Thuy-Yung Tran, Nigel Clarke, Marina Gashinova, and Mikhail Cherniakov. Automotive surface identification system based on modular neural network architecture. In *2017 18th International Radar Symposium (IRS)*, pages 1–8. IEEE, jun 2017. ISBN 978-3-7369-9343-3. doi: 10.23919/IRS.2017.8008124. URL http://ieeexplore.ieee.org/document/8008124/.

Aleksandr Bystrov, Edward Hoare, Thuy-Yung Tran, Nigel Clarke, Marina Gashinova, and Mikhail Cherniakov. Sensors for Automotive Remote Road Surface Classification. In *2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pages 1–6. IEEE, sep 2018. ISBN 978-1-5386-3543-8. doi: 10.1109/ICVES.2018.8519499. URL https://ieeexplore.ieee.org/document/8519499/.

Johan Casselgren, Mikael Sjödahl, and James LeBlanc. Angular spectral response from covered asphalt. *Applied Optics*, 46(20):4277, jul 2007. ISSN 0003-6935. doi: 10.1364/AO.46.004277. URL https://www.osapublishing.org/abstract.cfm?URI=ao-46-20-4277.

Johan Casselgren, Sara Rosendahl, Mikael Sjödahl, and Patrik Jonsson. Road condition analysis using NIR illumination and compensating for surrounding light. *Optics and Lasers in Engineering*, 77:175–182, feb 2016. ISSN 01438166. doi: 10.1016/j.optlaseng.2015.08.002. URL https://linkinghub.elsevier.com/retrieve/pii/S014381661500192X.

Chen Chen, Xiaohua Zhao, Hao Liu, Guichao Ren, and Xiaoming Liu. Influence of adverse weather on drivers' perceived risk during car following based on driving simulations. *Jour-*

nal of Modern Transportation, sep 2019. ISSN 2095-087X. doi: 10.1007/s40534-019-00197-4. URL http://link.springer.com/10.1007/s40534-019-00197-4.

M. Costa, W. Moniaci, and E. Pasero. INFO: an artificial neural system to forecast ice formation on the road. In *2003 International Conference Physics and Control. Proceedings (Cat. No.03EX708)*, pages 216–221. IEEE. ISBN 0-7803-7783-4. doi: 10.1109/CIMSA.2003.1227230. URL http://ieeexplore.ieee.org/document/1227230/.

Clemens Dannheim, Christian Icking, Markus Mader, and Philip Sallis. Weather Detection in Vehicles by Means of Camera and LIDAR Systems. In *2014 Sixth International Conference on Computational Intelligence, Communication Systems and Networks*, pages 186–191. IEEE, may 2014. ISBN 978-1-4799-5076-8. doi: 10.1109/CICSyN.2014.47. URL http://ieeexplore.ieee.org/document/7059167/.

Etcher. balenaetcher flash os images to sd cards & usb drives, 2020. URL https://www.balena.io/etcher/.

Dave Fisher-Hickey. 1.6 million uk traffic accidents, 2017. URL https://www.kaggle.com/daveianhickey/2000-16-traffic-flow-england-scotland-wales.

Ioannis Galanis, Priyaa Gurunathan, Dona Burkard, and Iraklis Anagnostopoulos. Weather-based road condition estimation in the era of Internet-of-Vehicles (IoV). In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2018. ISBN 978-1-5386-4881-0. doi: 10.1109/ISCAS.2018.8351582. URL https://ieeexplore.ieee.org/document/8351582/.

Aurelien Geron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Sebastopol, CA, 2017. ISBN 978-1491962299.

Kang Gui, Lin Ye, Junfeng Ge, Faouzi Alaya Cheikh, and Lizhen Huang. Road surface condition detection utilizing resonance frequency and optical technologies. *Sensors and Actuators A: Physical*, 297:111540, oct 2019. ISSN 09244247. doi: 10.1016/j.sna.2019.111540. URL https://linkinghub.elsevier.com/retrieve/pii/S0924424719300998.

Robin Heinzler, Philipp Schindler, Jurgen Seekircher, Werner Ritter, and Wilhelm Stork. Weather Influence and Classification with Automotive Lidar Sensors. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 1527–1534. IEEE, jun 2019. ISBN 978-1-7281-0560-4. doi: 10.1109/IVS.2019.8814205. URL https://ieeexplore.ieee.org/document/8814205/.

Jacek Jelonek and Jerzy Stefanowski. Experiments on solving multiclass learning problems by n2-classifier. In *Proceedings of the 10th European Conference on Machine Learning*, ECML '98, page 172–177, Berlin, Heidelberg, 1998. Springer-Verlag. ISBN 3540644172.

Patrik Jonsson. Road condition discrimination using weather data and camera images. In *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1616–1621. IEEE, oct 2011a. ISBN 978-1-4577-2197-7. doi: 10.1109/ITSC.2011.6082921. URL http://ieeexplore.ieee.org/document/6082921/.

Patrik Jonsson. Remote sensor for winter road surface status detection. In *2011 IEEE SENSORS Proceedings*, pages 1285–1288. IEEE, oct 2011b. ISBN 978-1-4244-9289-3. doi: 10.1109/ICSENS.2011.6127089. URL http://ieeexplore.ieee.org/document/6127089/.

Patrik Jonsson, Johan Casselgren, and Benny Thornberg. Road Surface Status Classification Using Spectral Analysis of NIR Camera Images. *IEEE Sensors Journal*, 15(3):1641–1656, mar 2015. ISSN 1530-437X. doi: 10.1109/JSEN.2014.2364854. URL http://ieeexplore.ieee.org/document/6936308/.

Jelena Kocic, Jovicic Nenad, and Vujo Drndarevic. Sensor and Sensor Fusion in Autonomous Vehicles. 2018.

Lu Junhui and Wang Jianqiang. Road surface condition detection based on road surface temperature and solar radiation. In *2010 International Conference on Computer, Mechatronics, Control and Electronic Engineering*, volume 4, pages 4–7. IEEE, aug 2010. ISBN 978-1-4244-7957-3. doi: 10.1109/CMCE.2010.5610255. URL http://ieeexplore.ieee.org/document/5610255/.

NVIDIA. Getting started with jetson nano developer kit, 2020. URL https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#write.

Juan Pablo. Machine learning edge devices: benchmark report, 2019. URL https://tryolabs.com/blog/machine-learning-on-edge-devices-benchmark-report/.

Malavika Panicker, Tanzeela Mitha, Kalyani Oak, Ashwini M. Deshpande, and Chandrajit Ganguly. Multisensor data fusion for an autonomous ground vehicle. In *2016 Conference on Advances in Signal Processing (CASP)*, pages 507–512. IEEE, jun 2016. ISBN 978-1-5090-0849-0. doi: 10.1109/CASP.2016.7746225. URL http://ieeexplore.ieee.org/document/7746225/.

Thierry Peynot, James Underwood, and Steven Scheding. Towards reliable perception for Unmanned Ground Vehicles in challenging conditions. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1170–1176. IEEE, oct 2009. ISBN 978-1-4244-3803-7. doi: 10.1109/IROS.2009.5354484. URL http://ieeexplore.ieee.org/document/5354484/.

Yiming Qian, Emilio J. Almazan, and James H. Elder. Evaluating features and classifiers for road weather condition analysis. In *2016 IEEE International Conference on Image Processing*

(ICIP), pages 4403–4407. IEEE, sep 2016. ISBN 978-1-4673-9961-6. doi: 10.1109/ICIP.2016.7533192. URL http://ieeexplore.ieee.org/document/7533192/.

Mahadev Satyanarayanan. The Emergence of Edge Computing. *Computer*, 50(1):30–39, jan 2017. ISSN 0018-9162. doi: 10.1109/MC.2017.9. URL http://ieeexplore.ieee.org/document/7807196/.

Scrapy. Scrapy | a fast and powerful scraping and web crawling framework, 2020. URL https://scrapy.org/.

Selenium. The selenium browser automation project, 2020. URL https://www.selenium.dev/documentation/en/.

Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, 3(5):637–646, oct 2016. ISSN 2327-4662. doi: 10.1109/JIOT.2016.2579198. URL http://ieeexplore.ieee.org/document/7488250/.

Stefan Silion and Cristian Fosalau. Wet road surfaces detection by measuring the air humidity in two points. In *2014 International Conference and Exposition on Electrical and Power Engineering (EPE)*, pages 744–747. IEEE, oct 2014. ISBN 978-1-4799-5849-8. doi: 10.1109/ICEPE.2014.6970008. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6970008.

Sudharson Sundararajan and Ismail Zohdy. Vehicle Automation and Weather : Challenges and Opportunities. (December), 2016.

Michal Taraba, Juraj Adamec, Matus Danko, and Peter Drgona. Utilization of modern sensors in autonomous vehicles, 2018.

TensorFlow. Tensorflow, 2020. URL https://www.tensorflow.org/.

Alaa Tharwat. Adaboost classifier: an overview, 02 2018.

Ville Viikari, Timo Varpula, and M. Kantanen. automotive radar technology for detecting road conditions. backscattering properties of dry, wet, and icy asphalt. pages 276 – 279, 12 2008.

M Yamada. A study of the road surface condition detection technique for deployment on a vehicle. *JSAE Review*, 24(2):183–188, apr 2003. ISSN 03894304. doi: 10.1016/S0389-4304(03)00006-7. URL https://linkinghub.elsevier.com/retrieve/pii/S0389430403000067.

Hun-Jun Yang, Hyeok Jang, and Dong-Seok Jeong. Detection algorithm for road surface condition using wavelet packet transform and SVM. In *The 19th Korea-Japan Joint Workshop on Frontiers of Computer Vision*, pages 323–326. IEEE, jan 2013. ISBN 978-1-4673-5621-3. doi: 10.1109/FCV.2013.6485514. URL http://ieeexplore.ieee.org/document/6485514/.

Shizhe Zang, Ming Ding, David Smith, Paul Tyler, Thierry Rakotoarivelo, and Mohamed Ali Kaafar. The Impact of Adverse Weather Conditions on Autonomous Vehicles.pdf. mar 2019.