Grupo 27 (Contribuições)

- Cosmin Trandafir – 57101 - (25h)
- João Serafim – 56376 - (22h)
- Martim Baptista – 56323 - (22h)
- Martim Paraíba – 56273 - (20h)

Introdução e Objetivos

Neste projeto, tivemos como objetivo treinar e otimizar um modelo de classificação que pudesse classificar as instâncias do dataset fornecido pelos docentes da cadeira. Este dataset é uma versão aumentada(*augmented*) e alterada do Data set QSAR biodegradation Data Set.

Identificamos a presença de dados ausentes no Data set, ou seja, algumas entradas não continham informações completas. Para lidar com essa situação, aplicamos técnicas de imputação, para estimar os valores vazios (NaN), assim como potencialmente alguns com valores nulos (como 0.0), com base nos dados disponíveis. Utilizámos abordagens como a substituição pela média ou pela mediana, pelos vizinhos mais próximos ou por iterações.

O dataset original apresentava, também, dados não normalizados, ou seja, as variáveis possuíam escalas diferentes, o que poderia afetar a performance do classificador. Portanto, aplicamos técnicas de normalização para garantir que todas as variáveis tivessem a mesma escala, possibilitando uma comparação justa e adequada entre elas.

Por fim, durante o processo de procura do melhor modelo para os nossos dados, decidimos testar diferentes hyper parâmetros, à medida que procurámos os melhores componentes (Imputer, Scaler, FeatureSelector e Classificador), a fim de os comparar justamente, em vez de comparar os componentes com parâmetros *Default* e fazer o hypertunning do modelo final.

Um parâmetro importante foi o “random_state” em funções como “train_test_split”, IterativeImputer, LogisticRegression, RandomForest, DecisionTree, SVC. Este é essencial para manter os resultados consistentes a cada execução.

Adicionalmente, o classificador inicial que escolhemos para testar os diferentes Imputers, Scalers e FeatureSelectors foi o LogisticRegression, um modelo usado para variáveis dependentes binárias. Nota: Usamos outros classificadores em vez do LogisticRegression, como o KNeighborsClassifier, no entanto obtivemos resultados piores no modelo final.

Quanto às métricas usadas para comparar os componentes, inicialmente usamos o F1_score, no entanto foi-nos relembrado que cada classe tem um valor diferente de F1 e que o nosso dataset é altamente desequilibrado, portanto chegamos à conclusão de que a melhor métrica a ser usada seria o Matthews’s correlation coefficient, uma vez que este é ideal para este tipo de situações.

O dataset é dividido duas vezes, a primeira divisão é um 80%/20% split, os 20% são reservados para mais tarde como Independent Validation Set (IVS) para avaliar o modelo final, os restantes 80% serão utilizados para treinar e testar o modelo, sendo subsequentemente divididos num 67%/33% split, 66% para treinar o modelo e 33% para testar o mesmo.

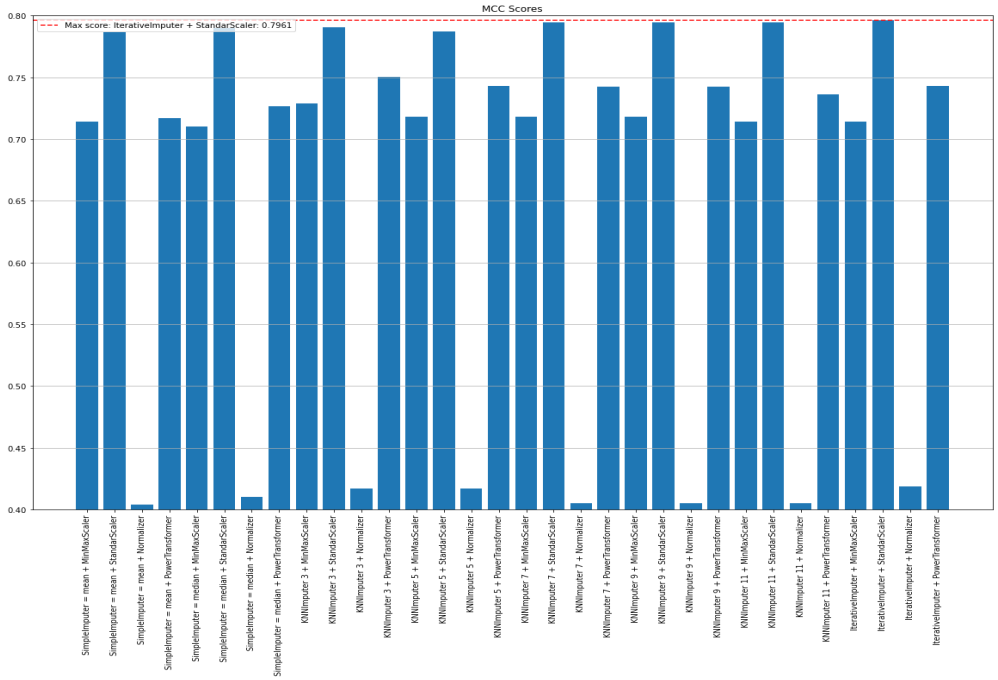
Processamento de Dados (Imputation e Scaling)

Para tornar os dados mais adequados ao treino do modelo procedemos à imputação de dados em falta e logo de seguida à normalização de todos os dados.

Percebemos que várias instâncias tinham valores vazios (NaN) ou nulos (0.0). Num passo inicial, e após alguma pesquisa, queríamos normalizar os dados e depois imputá-los, no entanto existem alguns Scalers que não conseguem, especificamente o MinMaxScaler e PowerTransformer, e por isso, escolhemos imputar primeiro os dados e depois normalizá-los.

Para a imputação exploramos três métodos diferentes: o SimpleImputer, o KNNImputer e o IterativeImputer. No SimpleImputer abordamos a imputação consoante a média e a mediana dos dados existentes. O KNNImputer, por sua vez, utiliza uma estratégia baseada nos k vizinhos mais próximos para estimar os valores ausentes. Exploramos os resultados com K-nearest neighbours = [3,5,7,9 e 11]. Para o IterativeImputer, não verificamos nenhum hyper parâmetro específico.

Para garantir que as variáveis possuíssem a mesma escala e evitar que alguma dominasse as outras devido às diferenças nas unidades ou magnitudes, testámos quatro métodos de normalização: MinMaxScaler, StandardScaler, Normalizer e PowerTransformer. O MinMaxScaler ajusta os valores para um intervalo específico, por defeito entre 0 e 1. O StandardScaler padroniza os valores para ter média zero e desvio padrão igual a 1. O Normalizer ajusta cada exemplo individualmente para que tenha uma norma específica. O PowerTransformer realiza transformações não lineares nos dados para torná-los mais gaussianos, ou seja, mais próximos de uma distribuição normal.



Como podemos observar pelo gráfico acima, a combinação do IterativeImputer e o StandardScaler resultaram no melhor MCC score (0.7961). Portanto, escolhemos essa combinação como a abordagem final para realizar a imputação dos dados faltosos e a normalização dos dados. É também de notar como o StandardScaler tem sempre um resultado muito elevado, independentemente do Imputer utilizado.

Seleção de features

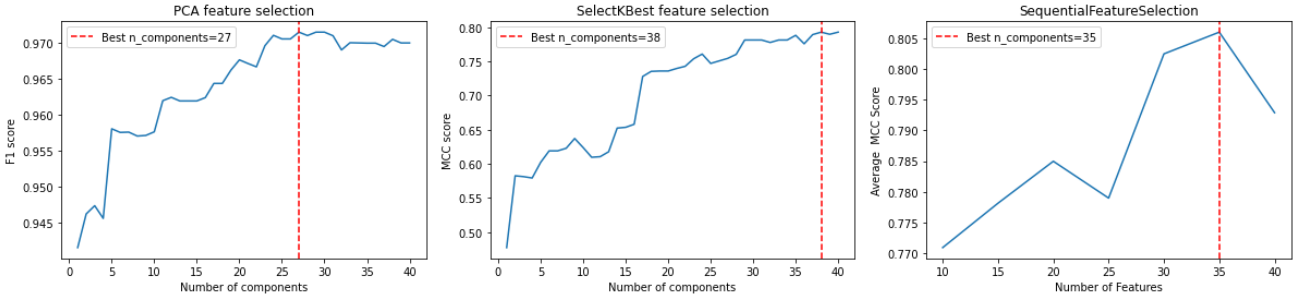
Nesta fase, decidimos transformar o dataset usando os melhores Imputer e Scaler acima identificados. Com os dados transformados, decidimos determinar se haveria features que não contribuíam para a classificação final e apenas inseriam *noise.* Para tal, testamos diferentes métodos, incluindo a correlação de Pearson, o PCA (Análise de Componentes Principais), o SelectKBest e o SequentialFeatureSelection. Também verificamos o MCC score dos dados sem seleção de features, para garantir que o modelo ganhava valor em usar esta componente.

Pearson: Sendo que o Pearson só tem em conta a relação entre cada feature e classificação, e não a dependência entre features. Observando os resultados de MCC com as restrições impostas pelo Pearson(MCC=0.2142), podemos concluir que as features deste problema possuem alguma dependência entre si. Desta forma, selecionar features com o Pearson é um mau método de Feature Selection para este problema.

PCA: Para este caso o PCA não é o mais adequado, uma vez que o número de features em relação à quantidade de entradas não é o principal desafio. Outro problema que torna o PCA indesejável é a perda do significado das features, dado pela natureza deste método. Ainda assim, determinamos que o número ideal de features para o PCA seria de 27.

SelectKBest: O SelectKBest utiliza medidas estatísticas para avaliar a relevância de cada feature em relação à variável alvo e seleciona as k melhores features com base nessa medida. Nos nossos testes descobrimos que 38 é o número ideal de features para o método.

SequentialFeatureSelection: Para os testes do número ótimo de features do SequentialFeatureSelection tivemos de reduzir a resolução dos testes, testando valores de k de 5 em 5 (em vez de 1 em 1), esta escolha deve-se ao elevado tempo de execução que seria testar de 1 em 1.



Após compararmos os resultados dos quatro métodos, obtivemos as seguintes pontuações MCC:

- Without feature selectors: MCC score = 0.7961
- Pearson: MCC score = 0.2142
- PCA: MCC score = 0.8057
- SelectKBest: MCC score = 0.7929
- SequentialFeatureSelection: MCC score = 0.8060

Assim, escolhemos o SequentialFeatureSelection (k=35), visto que é o que apresenta as melhores métricas, em especial o MCC. O PCA também apresenta métricas semelhantes, no entanto como mencionado anteriormente este não é adequado ao nosso problema, para além de este criar novas features não interpretáveis algo que não acontece com o SequentialFeatureSelection que apenas seleciona as features que levam a melhores scores.

Modelo de Classificação

Utilizando os melhores métodos de imputação, escalamento e seleção de variáveis. O IterativeImputer, o StandardScaler e o SequentialFeatureSelection (k=35), testamos os seguintes modelos de classificação:

- DecisionTreeClassifier
- RandomForestClassifier
- SVC
- LogisticRegression
- KNeighborsClassifier

Antes de comparamos os modelos de classificação, iremos descobrir quais os melhores hyper parâmetros para os mesmos.

Hyperparameter tunning

Para cada classificador usamos o GridSearchCV sobre uma grid com um range de híper parâmetros adequados ao respetivo classificador. Seguem-se as grids definidas e os hyper parâmetros otimos, encontrados pelo GridSearchCV utilizando o MCC como score.

Para o DecisionTreeClassifier:

```
'criterion': ['gini', 'entropy'],
'max_depth': [2, 4, 6, 8, 10],
'min_samples_split': [2, 4, 6, 8, 10],
'min_samples_leaf': [1, 2, 3, 4, 5]
```

Melhores Hiper parâmetros: `{'criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 4, 'min_samples_split': 2}`

Para o RandomForestClassifier:

```
'n_estimators': [50, 100, 200],
'max_depth': [5, 10, 15],
'min_samples_split': [2, 5, 10],
'min_samples_leaf': [1, 2, 4]
```

Melhores Hiper parâmetros: `{'max_depth': 15, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 50}`

Para o SVC:

```
'gamma': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100],
'C': [0.01, 0.1, 1, 10, 100, 1000, 10000]
```

Melhores Hiper parâmetros: `{'C': 100, 'gamma': 0.1}`

Para o Logistic Regression:

```
'C': [0.1, 1.0, 10.0],
'penalty': ['l1', 'l2'],
'solver': ['liblinear', 'saga'],
'max_iter': [100, 200, 500, 1000, 10000]
```

Melhores Hiper parâmetros: `{'C': 10.0, 'max_iter': 500, 'penalty': 'l1', 'solver': 'saga'}`

Para o KNeighborsClassifier:

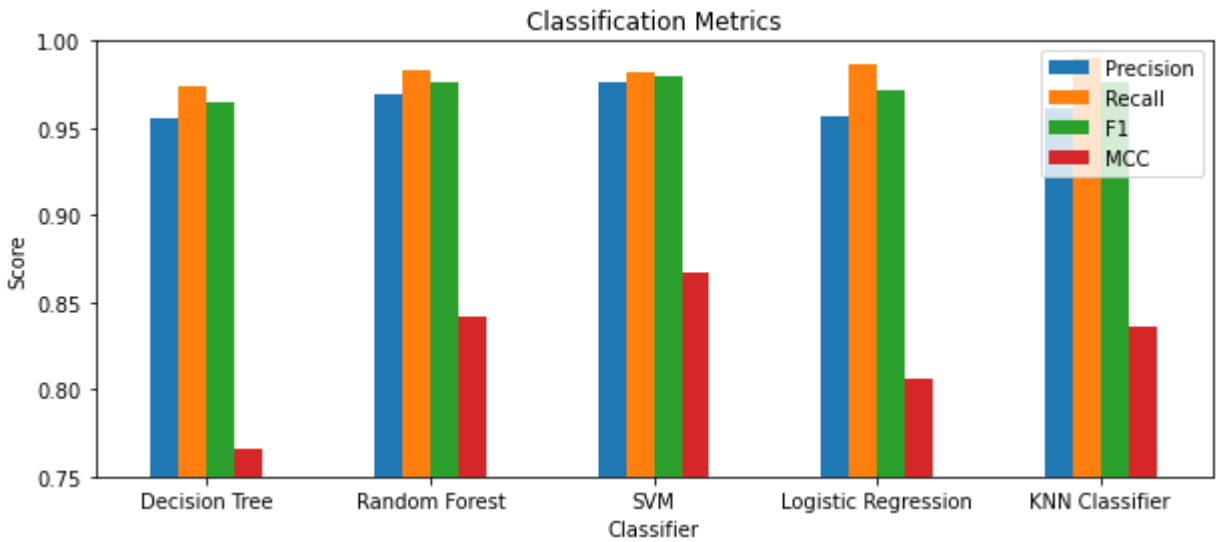
```
'n_neighbors': [3, 5, 7, 9],
'weights': ['uniform', 'distance'],
'p': [1, 2]
```

Melhores Hiper parâmetros: `{'n_neighbors': 3, 'p': 1, 'weights': 'distance'}`

Escolher o modelo de classificação

Após realizarmos o processo de imputação, escalonamento e seleção de variáveis, obtivemos os melhores resultados utilizando os seguintes métodos: IterativeImputer para a imputação de dados em falta, StandardScaler para o escalonamento e SequentialFeatureSelection com k=35 para a seleção de variáveis.

De seguida, procedemos à avaliação dos diferentes classificadores utilizando o MCC.



Valores MCC para:

- Decision Tree: 0.765571;
- Random Forest: 0.842121;
- SVM: 0.866652;
- Logistic Regression: 0.806028;
- KNN Classifier: 0.836530.

Visto que o resultado do SVM é o melhor, podemos concluir que é o melhor classificador para o nosso conjunto de dados.

Discussion and Conclusions

Concluímos que o melhor modelo para classificar os dados do QSAR biodegradation Data Set é constituído pelos seguintes componentes: **IterativeImputer, StandardScaler, SequentialFeatureSelection(k=35) e o SVM(C: 100, gamma: 0.1)**. Testamos o modelo final com o Independent Validation Set (IVS), obtemos os seguintes resultados:

	Precision	*Recall*	*F1-score*	*Support*
NRB	0.90	0.92	0.91	153
RB	0.98	0.98	0.98	760

MCC: 0.8913

Estamos satisfeitos com o nosso modelo, pois obtivemos bons resultados. O F1 score é elevado tanto para a classe mais representada (RB) como para a menos representada (NRB), o MCC também é elevado, o que é positivo. Os resultados elevados poderiam ser atribuídos a overfitting, mas estes são provenientes do teste no IVS, sendo este um set independente é impossível o modelo estar overfitted ao mesmo.