MARZO DE 2024

HOME EQUITY LINE OF CREDIT ANALYSIS

PROYECTO FINAL MACHINE LEARNING

ULISES DIEZ SANTAOLALLA
2º IMAT A

Contenidos

1. Exploratory Data Analysis

Para comenzar este proyecto se realizó un análisis inicial del set de datos con el que se va a estar realizando, el cual contiene información sobre aplicaciones “Home Equity Line of Credit (HELOC)” hechas por propietarios reales.

Para comenzar, se analizaron el número de datos faltantes “NaN” por cada variable del dataset, donde se obtuvo:

```
    Número de NaN por variable:
RiskPerformance                2197
ExternalRiskEstimate              0
NetFractionRevolvingBurden        0
AverageMInFile                    0
MSinceOldestTradeOpen            27
PercentTradesWBalance            56
PercentInstallTrades              0
NumSatisfactoryTrades            17
NumTotalTrades                   23
PercentTradesNeverDelq            0
MSinceMostRecentInqexcl7days      0
```

También se tuvo en cuenta los caracteres especiales del dataset, entre los que se encuentran:

- -9: No Bureau Record or No Investigation
- -8: No Usable/Valid Trades or Inquiries
- -7: Condition not Met (e.g. No Inquiries, No Delinquencies)

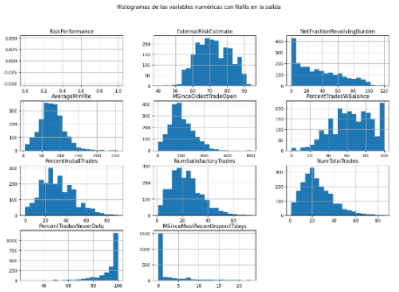
Al ver que los datos -9 se encontraban en 50 filas completas, por lo que correspondían a filas sin datos, se procedieron a eliminarse.

```
7440  1;-9;-9;-9;-9;-9;-9;-9;-9;-9;-9
7441  0;-9;-9;-9;-9;-9;-9;-9;-9;-9;-9
7442  1;-9;-9;-9;-9;-9;-9;-9;-9;-9;-9
7443  1;-9;-9;-9;-9;-9;-9;-9;-9;-9;-9
```

En el caso del -8 vemos que hay 52 celdas del Dataset que contienen este tipo de dato, al igual que 50 en el caso del -7, por lo que no se ha considerado necesario eliminar las filas completas que contienen estos tipos de datos ya que se perdería mucha información de una manera innecesaria, por lo que al ser casos puntuales se decidió realizar imputación sobre estos tipos de datos.

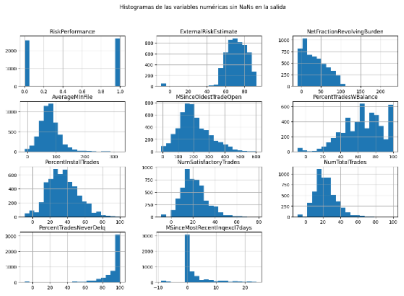
```
7294  0;88;0;103;211;71;57;13;14;100;-7
7295  1;75;43;59;169;83;36;21;22;100;-7
7296  1;66;42;69;127;58;33;17;18;94;-7
7297  0;81;36;91;220;67;39;20;23;100;-7
7298  1;71;87;95;177;100;75;24;24;96;-7
7299  1;69;37;43;112;83;43;5;7;100;-7
```

Dado a que la imputación por media es muy sensible a posibles valores atípicos, y que la imputación por moda suele ser más útil para variables categóricas, se decidió usar imputación por mediana, la cual es más robusta frente a valores atípicos y cuando los datos no tienen una distribución normal.



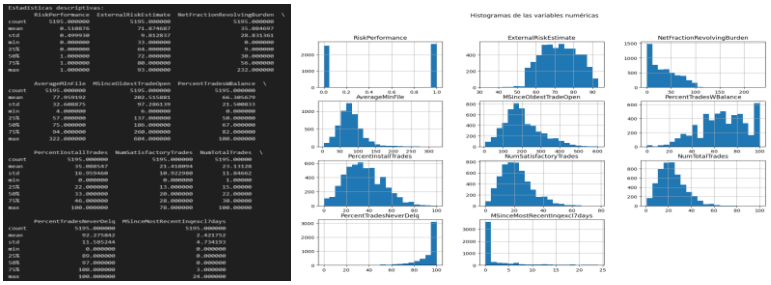
Al igual que con estos valores, se decidió hacer imputación sobre los datos faltantes en las distintas variables de entrada. Se decidieron eliminar las filas que contenían datos faltantes en la variable de salida, ya que estos datos no se pueden imputar, y sería trabajar con información de la que no sabemos su output.

Sin embargo, antes de realizar estas imputaciones y eliminaciones en los datos, se decidió graficar el comportamiento de los datos, dividiendo en dos dataframes, uno con los datos que la variable de salida es NAN, y otro en el que la variable de salida es correcta. Una vez que se vio que el comportamiento de los datos no varía en el caso de que la variable de salida es faltante, se procedió a eliminar e imputar los datos mencionados anteriormente.



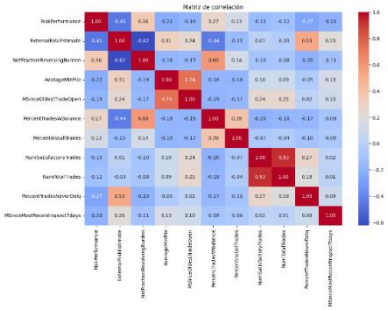
Una vez realizados estos cambios iniciales, para tener una idea general de los datos de cada variable se ofreció una serie de estadísticas descriptivas generales para cada una de las clases, permitiendo analizar el número de datos por cada uno, su media, desviación típica, mínimo, máximo, y sus correspondientes cuartiles.

Así se pudo ver cómo las clases se distribuyen lo cual se acompañó de una serie de histogramas para poder ver la distribución de estas mismas.

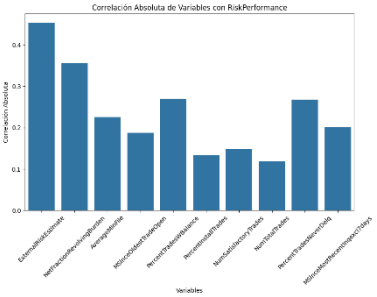


De esta manera se pudo ver cierto comportamiento interno dentro de las variables, como en el caso de cierto comportamiento Lognormal por parte de variables como “NumTotalTrades” o “NumSatisfactoryTrades”, al igual que el comportamiento bimodal de la variable de salida.

Además, por la diferencia entre varianzas y medias, las variables no se encuentran a la misma escala, por lo que se procedió a estandarizar los datos. Aunque la estandarización se debería de aplicar después de la división de train/test, se aplicará ahora y se tendrá en cuenta, al igual que se ha llevado a cabo en las prácticas realizadas a lo largo del curso.

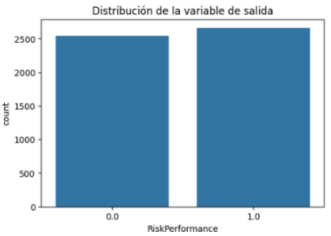


Para ver las relaciones internas de los datos se calculó la matriz de correlación del Dataset, viendo fuertes correlaciones directas como en el caso de “NumTotalTrades”-“NumSatisfactoryTrades o “MSinceOldestTradeOpen”-“AverageMinFile”, al igual que variables que están fuertemente correlacionadas de manera inversa, como “ExternalRiskEstimate”-”NetFractionRevolvingBurden“. Sin embargo, al contar con pocos datos, no se consideró necesario eliminar una de las variables fuertemente correlacionadas, ya que podría resultar en pérdida de información para nuestros modelos, que se entrenarán con pocos datos.

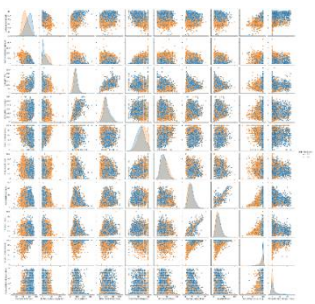


Para tener una idea genérica de la relación que tiene cada variable con la variable de salida, y por lo tanto, la importancia que tiene cada variable en base a correlaciones con el output del dataset, se graficó de manera independiente el valor absoluto de las correlaciones, para más adelante poder comparar esta hipótesis de importancia de variables con la importancia de estas variables en cada modelo.

Además de esto, se realizó una gráfica individual de la variable de salida para ver su comportamiento, viendo de esta manera que los datos de salida se encuentran distribuidos de una manera equitativa entre un Riesgo Alto y Bajo.



Finalmente se graficó la matriz de dispersión para visualizar las relaciones entre las diferentes variables del Dataset, coloreando los puntos en base a la variable de salida para así poder ver la distribución de estos, al igual que en la diagonal se ajustó un gráfico de densidad de kernel para así ver la distribución de cada variables.



Como se puede ver en esta representación, al ver la distribución de los datos de salida respecto a las relaciones entre las variables, en varios casos podemos ver cierta estructura interna, por lo que se debería de considerar este punto.

Una vez realizado este análisis exploratorio preliminar para entender la distribución y comportamiento de los datos dentro de las variables, se procedió a realizar la división del Dataset entre set de datos de train y el de test, haciendo una división del 80-20%.

Los respectivos sets de datos quedaron de la siguiente manera:

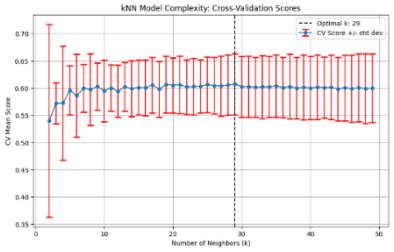
- Dimensiones del conjunto de entrenamiento (X_train, y_train): (4156, 10) (4156,)
- Dimensiones del conjunto de prueba (X_test, y_test): (1039, 10) (1039,)

En este ejercicio se ha decidido no eliminar variables altamente correlacionadas como se ha mencionado anteriormente, ya que, aunque esto podría mejorar la interpretación del modelo y reducir el riesgo de multicolinealidad y sobreajuste, al no reducir significativamente la eficiencia computacional ya que el Dataset no es muy grande, se prefirió no arriesgar una pérdida de información relevante, al igual que la introducción de sesgo en el modelo si se eliminan variables importantes para predecir la variable objetivo.

Hasta aquí, tanto la división de los Datasets en test-train, la estandarización, y técnicas de exploración de los datos se decidió hacerlo de manera manual, sin el uso de la librería SKLearn.

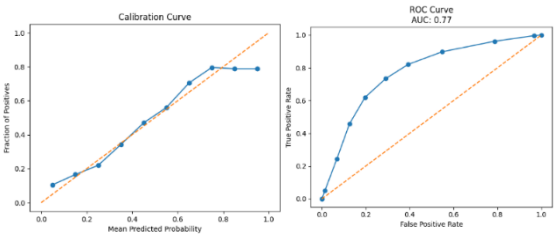
2. Clasificación

En esta parte se hizo uso de diferentes modelos de regresión y clasificación para entrenarlos con los datos obtenidos en los sets de train y comprobar su eficacia en los sets de test.



1. KNN

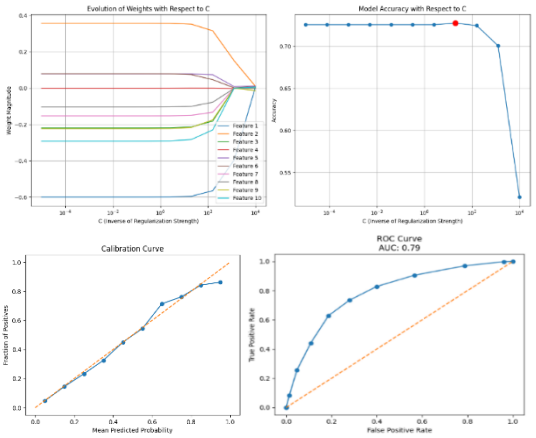
En primer lugar se uso el método de K-Nearest Neighbors, por lo que en primer lugar se buscó la k óptima para el modelo a través de cross-validation con nFolds=5. Este código no se ha incluido en el archivo Python por el tiempo que conlleva en ejecutarse, pero se puede encontrar en el Notebook que se ha adjuntado con la práctica, que es donde se ha ido desarrollando todo el código sin su formato modular.



Una vez obtenida la k óptima (k=29), se entrenó el modelo usando la distancia de Minkowski con p=2 (distancia Euclídea), y se elaboraron diferentes gráficas y métricas para poner a prueba su funcionamiento.

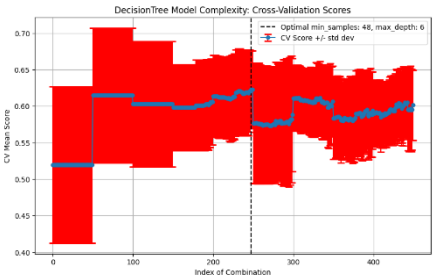
2. Logistic Regression

Para aplicar este modelo, se usó la regularización ElasticNet, ya que esta junta las regularizaciones Lasso y Ridge y puede conducir a un modelo que es robusto a varios tipos de datos y previene el sobreajuste. Para esto, se calculó la C óptima a través de entrenar el modelo con diferentes valores de C y ver cuál es óptimo.



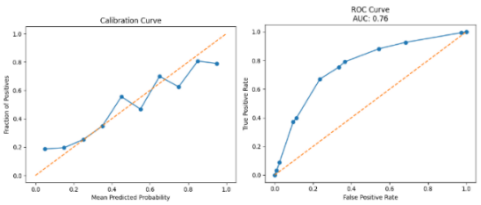
Una vez calculada este valor de regularización, se entrenó el modelo, obteniendo los siguientes resultados:

3. Decision Trees



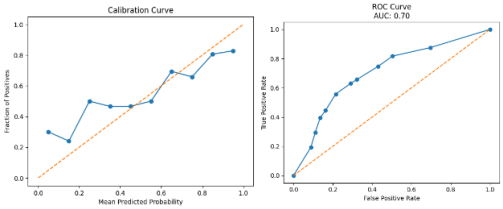
Además de los modelos anteriores, los cuales además de haberlos visto en clase, se habían trabajado en las prácticas, por lo que se partía de cierto conocimiento de implementación en el código, con la ayuda de diversas fuentes en Internet se procedió a diseñar el modelo de árboles de decisiones.

Una vez creada la clase del modelo, se crearon diferentes árboles variando la altura máxima de cada árbol, y para cada altura, variando el mínimo número de observaciones requerido en cada nodo y se comprobó la accuracy en cada uno de ellos a través de cross-validation con nFolds=5. De esta manera, se decidió poner un número mínimo de observaciones de 48 y una altura máxima del árbol de 6.



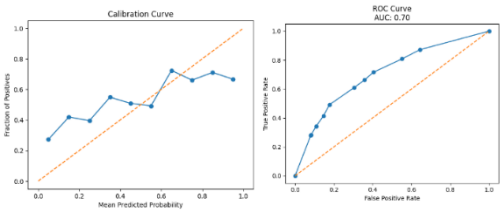
4. Boostrap Aggregating (Bagging)

Una vez creada la clase DecisionTree, se decidió crear los modelos de Bagging y RandomForest, ya que consisten en la implementación de los árboles anteriores, pero con la aplicación de boostrap en el caso de Bagging. En el caso de este modelo, al igual que en Random Forest, se decidió que los árboles que se crearan dentro de ellos partiesen de los parámetros obtenidos en cross-validation en el apartado anterior, aunque esto se podrá modificar en caso necesario



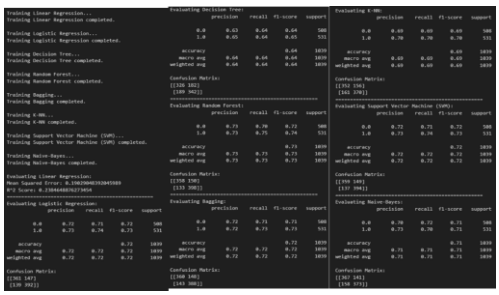
5. Random Forest

En el caso de Random Forest, además de aplicar la selección de muestras boostrap, se implementó la selección de índices del set de datos de manera aleatoria, obteniéndose las siguientes métricas:



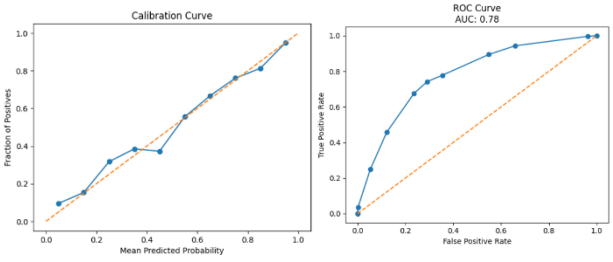
6. SK-Learn

Finalmente, para comparar los modelos realizados a mano con modelos ya creados, se realizó este apartado extra, en el cual se pusieron a prueba los sets de datos con los diferentes modelos, pero esta vez los obtenidos de la librería SK-Learn de una manera genérica, sin entrenar los parámetros, ya que se hizo esto de una manera puramente comparativa, para después obtener las métricas que también nos ofrece esta librería.

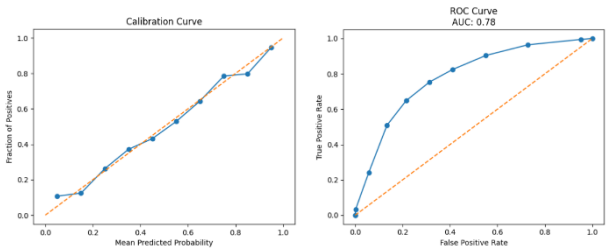


Además, al poder utilizar los diferentes modelos que ofrece esta librería, se implementó también el modelo de SVM (Máquinas de Vector Soporte) tanto con un kernel lineal como el kernel radial RBF, y Naive-Bayes, ya que se ha visto su funcionamiento en clase y podría resultar de interés para ver su rendimiento ante este set de datos y compararlo con el resto de los modelos.

A continuación, se puede ver los resultados del modelo SVM entrenado con un kernel lineal, en el cual se entrenó el parámetro de regularización C, obteniéndose el óptimo en 0.01. Además, se aplicó un escalador para estandarizar las características, eliminando la media y escalando a la varianza unitaria. Finalmente, se aplicó un calibrador de probabilidad para mejorar las estimaciones de probabilidad del clasificador.

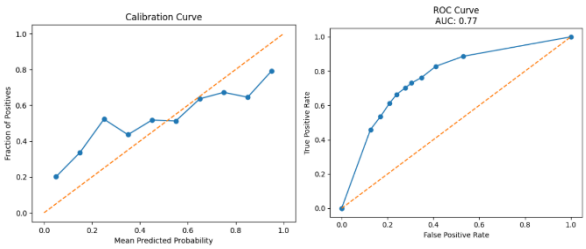


Creando un modelo de SVM, pero usando en este caso el kernel radial RB, se entrenó el parámetro de regularización C, obteniendo en este caso el óptimo en 1000, y el parámetro gamma, el cual controla el ancho del kernel RBF, obteniéndose el óptimo de este en 0,01. Al igual que antes, se usó un escalador para la estandarización de las características.



En ambos modelos de SVM se usó GridSearch con Cross Validation para encontrar los parámetros óptimos de los modelos.

Al igual que en el modelo anterior, se entrenó un modelo de Naive-Bayes Gaussiano, donde se usó cross-validation con nFolds=10 para probar diferentes priors, pero ninguno de la lista probada resultó eficiente, por lo que se decidió ejecutar el modelo de SKLearn sin ningún prior preestablecido.



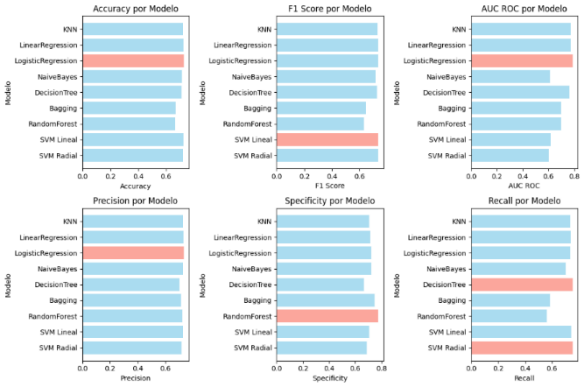
7. Análisis Comparativo

Para la elaboración de este apartado, cada vez que se elaboraban las métricas de cada modelo, a través de la función “escribir_resultados_en_csv” se han ido guardando a lo largo de su ejecución en el csv nombrado “MetricasModelos.csv” las métricas de cada modelo. A continuación, se puede ver una lista con las características de cada modelo que se han ido guardando en este archivo:

- TP,FP,FN,TN
- Accuracy (Nos permite ver la proporción de predicciones correctas sobre el numero de predicciones totales)
- Precision (Mide la capacidad del modelo para no clasificar erróneamente como positivos a elementos que son negativos)
- Recall (Mide la capacidad del modelo para identificar todos los elementos positivos de manera correcta)
- Specificity (Mide la capacidad del modelo para identificar todos los elementos negativos de manera correcta)
- F1 Score (Es la media armónica de precision y recall)
- AUC ROC (Representa la capacidad del modelo para distinguir entre las clases)

Con estos datos, después de haber implementado los diferentes modelos y haber visto sus diferentes resultados al hacer la comparativa de rendimiento frente al set de test una vez han sido entrenados con el set de train, voy a proceder al análisis comparativo de los modelos gracias al uso de este csv.

Para ello, se elaboró el archivo “models_comparison.py”, el cual nos permite graficar los diferentes modelos para cada métrica y colorear el modelo con mejor rendimiento, como se puede ver a continuación:



Como se puede ver reflejado, el modelo que mejor ha rendido en términos de Accuracy, AUC ROC Y Precision es el modelo de Regresión Logística, aunque todos los modelos han obtenido

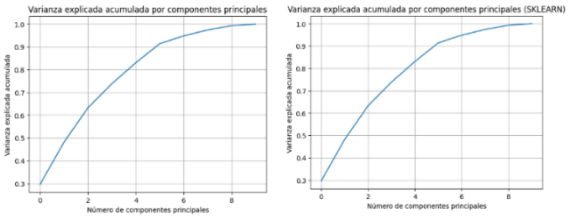
puntuaciones parecidas. Teniendo en cuenta el Recall, vemos que el modelo que lo lidera es SVM con kernel RBF a la par que Decision Tree, mientras que en campos como Specificity, Random Forest es el modelo que mejor ha funcionado frente al set de test, mientras que SVM con el kernel lineal ha obtenido mejor resultado respecto a la métrica de F1 Score.

Sin embargo, teniendo en cuenta que respecto a Accuracy, la cual considero una de las métricas más relevantes a tener en cuenta junto a AUC ROC, y que a nivel general ha trabajado de una manera mejor con el set de test, el modelo que pienso que es más óptimo para este set de datos es el de Regresión Logística. Además, aunque en ciertas métricas han obtenido puntuaciones más altas los modelos de Random Forest y SVM, hay que tener en cuenta la complejidad del modelo, lo cual es algo a considerar ya que si se trabajase con sets de datos más grandes, en SVM se escogiesen kernels más complejos, o en Random Forest se aumentas el número de estimadores, la necesidad de cómputo que necesitarían los modelos sería mucho más elevada que la de Regresión Logística, sin resultar esto en una mejora significante en los resultados de los modelos.

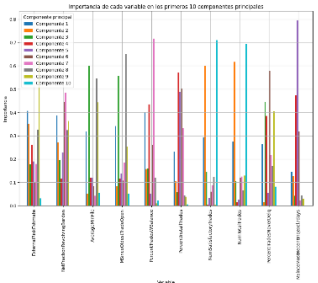
3. Aprendizaje No Supervisado

Una vez finalizado con los modelos de clasificación, el estudio de las diferentes técnicas utilizadas, tanto con la comparativa entre los diferentes modelos, se dará paso a la parte de Aprendizaje No Supervisado.

1. PCA (Principal Component Analysis)

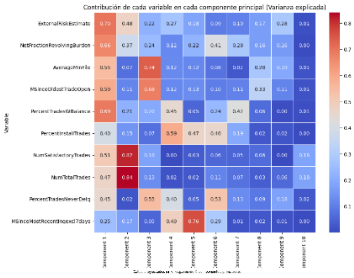


Para el análisis por componentes principales se realizó a mano la clase PCA, la cuál se incluyó en el módulo de modelos. A partir del dataset modificado, es decir, estandarizado y con los datos correctamente imputados y eliminados correspondientemente, se ejecutó el modelo PCA, al igual que el modelo PCA ofrecido por la librería SKLearn, viendo que el resultado es idéntico:

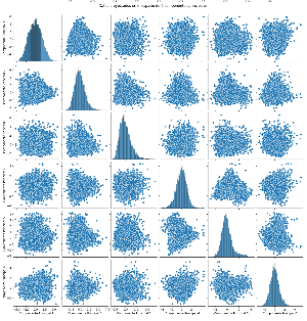


De esta manera se puede observar que la varianza de los datos se puede expresar al 95% usando sólo 6 de sus 10 componentes principales, y en torno al 75% con sólo 3 de ellas.

Además de esto, se decidió graficar la importancia de cada variable respecto a cada componente, al igual que la contribución de cada variable en cada componente principal a través de un mapa de calor. A través de estas visualizaciones podemos obtener información útil sobre la relación entre las variables originales y los componentes principales. Gracias a esta gráfica se puede observar visualmente cómo la primera componente es la que más información guarda de los datos, mientras que la información que proporciona la última componente es mínima



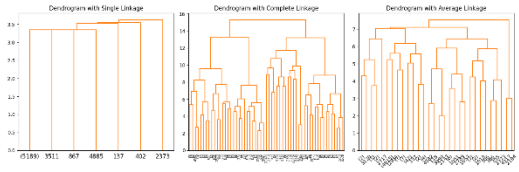
Finalmente, como se ha podido ver anteriormente que con 6 componentes principales se puede explicar en torno al 95% de la varianza del modelo, se decidió proyectar los datos en el espacio de 6 componentes principales y ver su comportamiento.



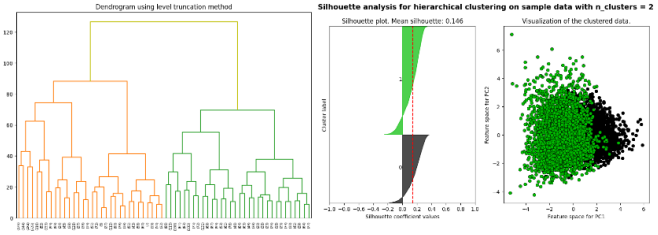
2. Clustering Algorithms

Para los algoritmos de Clustering se aplicaron tres técnicas de las vistas en clase para datos multivariantes.

1. *Hierarchical*



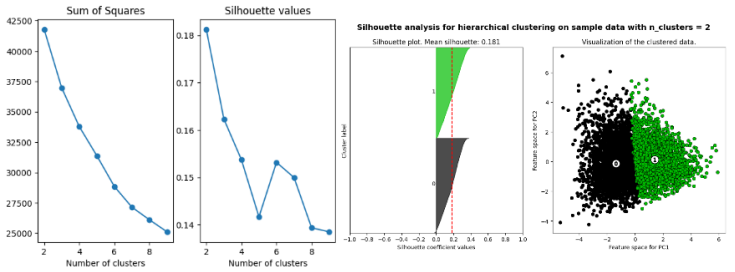
Usando el método de Hierarchical clustering, se realizó el dendograma de los datos, usando diferentes métodos de linkage para así ver las diferencias entre sí.



A continuación, se usó el linkage de Ward, ya que busca la mínima varianza, y a continuación se aplicó el método de truncamiento. Como se puede ver, se distinguen dos clústers, por lo que con este número se realizó el Hierarchical Clustering para ese número de clústers, obteniendo el resultado y siluetas de a continuación.

2. *K-Means*

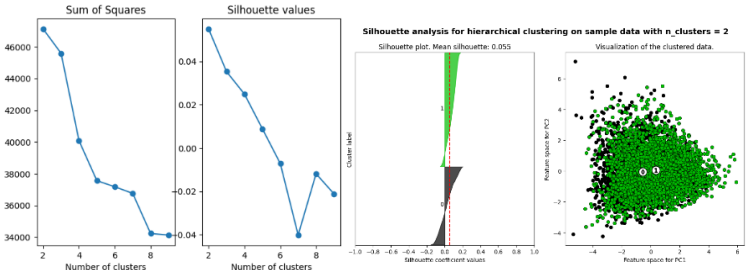
En este caso se realizó el modelo desde 2 clústers hasta 10, comparando la silueta de los modelos y la distancia, obteniéndose la gráfica resultaste de a continuación.



Por lo que, dejando a un lado la técnica del codo que se podría usar en la gráfica de la izquierda, en la gráfica de la derecha podemos ver reflejado que el número de clúster que tiene mayor silueta también es el de dos, como era de esperar.

Con este dato, se procedió a graficar el modelo de manera particular para dos clústers obteniendo para el clúster 0 un total de 2659 datos, y 2536 para el clúster 1.

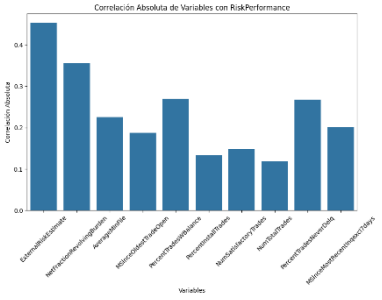
3. *Gaussian Mixture Models*



Al igual que en el modelo anterior, se realizó el modelo desde 2 clústers hasta 10, reflejando los datos en la gráfica de a continuación. Volviendo a tener en cuenta la gráfica de la derecha, se escogió el valor de 2 clústers, ya que es el valor asociado a la silueta mayor.

Al igual que en los casos anteriores, se realizó el modelo de manera específica para 2 clústers, obteniendo el resultado de a continuación.

4. Conclusiones

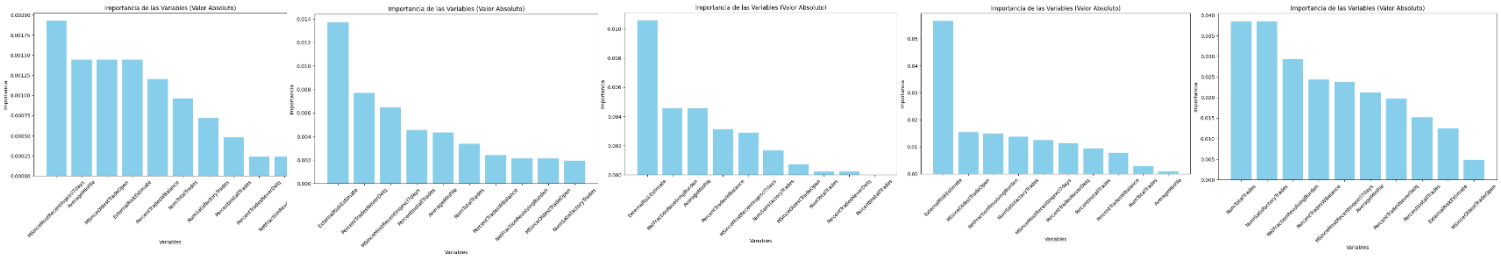


Para concluir este proyecto, cabe destacar que las hipótesis que se estableció en el apartado 1, donde viendo la correlación de cada variable con el output, se estableció que la variable con más importancia era ExternalRiskEstimate, mientras que la que menos importancia tenía era NumTotalTrades. Para verificar esto, a lo largo del apartado 2, para cada modelo realizado manualmente, se elaboraron unas series de funciones que permitieron el análisis de cada variable en cada modelo.

En el caso de KNN y Logistic Regression se creó una función que calcula la accuracy del modelo de manera global, y luego la accuracy del modelo si se elimina cada una de las variables, para ver de esta manera cuánto contribuye cada variable a la variación de accuracy del modelo. En el caso de árboles, se calcula igualmente la accuracy inicial, y luego se calcula la accuracy de permutar de

manera aleatoria cierta variable, para así ver como varían las accuracies y ver cómo contribuye cada variable.

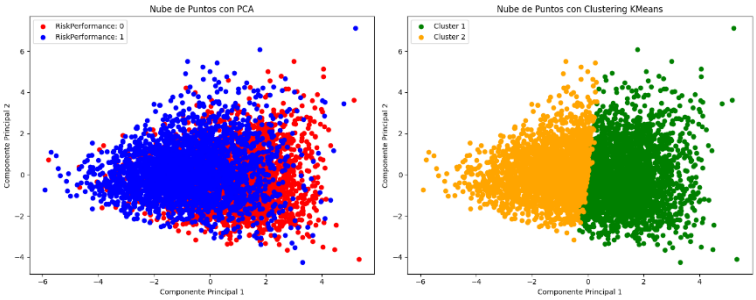
Como se puede observar, en la mayoría de los modelos la variable que cobra más importancia es la de ExternalRiskEstimate, y en varios modelos NumTotalTrades se encuentra entre las variables con menos peso.



Como se mencionó en la comparativa de los modelos de clasificación en el apartado 2, teniendo en cuenta las diferentes métricas obtenidas con los diferentes modelos, y teniendo también en consideración la complejidad de los modelos, los parámetros necesarios a entrenar para cada modelo, y los resultados de estos, el modelo que mejor se ajusta a los datos es el de Regresión Logística, en el que se empleó regularización Elastic-Net, y Cross-Validation para obtener el valor óptimo de regularización.

En relación a los métodos de clustering, cabe destacar que si al igual que en los modelos de clasificación, los datos se separan en train y en test, y se entrenan los modelos con el set de train para después ponerlos a prueba con el set de test, los resultados que podrían ser parecidos si los dos clústers se asocian con la salida 0 y 1 de la variable RiskPerformance. Sin embargo, hay que tener en cuenta que los algoritmos de clasificación tienen en consideración la variable de salida para entrenar a los modelos en base a su output, mientras que en clustering lo que se busca es, por ejemplo, minimizar la distancia intra-cluster y maximizar la distancia inter-cluster, sin usar datos etiquetados para el entrenamiento de los modelos. También hay que considerar que las métricas de evaluación son diferentes, ya que, al no contar con la variable de salida, en clustering se usan métricas como el índice de silueta, mientras que en clasificación encontramos métricas tales como accuracy, precisión, F1 Score... Es por esto por lo que usar modelos de clutering en sustitución a los de clasificación carecería de la robustez y rendimiento de los modelos de clasificación.

Para comprobar esto, reduje la dimensión de mis datos a sus dos componentes principales usando PCA, y comparé los datos con salida 0 y 1 con los resultados que se obtienen al separar los datos en dos clusters usando KMeans. De esta manera podemos ver que usar clustering como un método de clasificación no resultaría óptimo.



De esta manera se da por concluido este análisis de los datos, donde tras un análisis inicial del set de datos proporcionados, se ha podido ver el funcionamiento de diferentes modelos de clasificación programados a mano, al igual que otros trabajados a través de la librería SKLearn, en nuestro set de datos, para finalmente estudiar el Análisis por Componentes Principales del mismo y una serie de modelos de Clustering.

Bibliografía

Analytics Vidhya. (2024). Retrieved from Understand Random Forest Algorithms With Examples: https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

Analytics Vidhya. (2024). Retrieved from PCA | What Is Principal Component Analysis & How It Works?: https://www.analyticsvidhya.com/blog/2016/03/pca-practical-guide-principal-component-analysis-python/

DataCamp. (2024). Retrieved from Principal Component Analysis (PCA) in Python: https://www.datacamp.com/tutorial/principal-component-analysis-in-python

Geeks For Geeks. (2024). Retrieved from Decision tree implementation: https://www.geeksforgeeks.org/decision-tree-implementation-python/

IBM. (2024). Retrieved from Funcionamiento de SVM: https://www.ibm.com/docs/es/spss-modeler/saas?topic=models-how-svm-works

IBM. (2024). Retrieved from What is random forest?: https://www.ibm.com/topics/random-forest#:~:text=Random%20forest%20is%20a%20commonly,Decision%20trees

Machine Learning Maestry. (2024). Retrieved from How To Implement The Decision Tree Algorithm From Scratch In Python: https://machinelearningmastery.com/implement-decision-tree-algorithm-scratch-python/

Matplotlib. (2024). Retrieved from Visualization with Python: https://matplotlib.org/

Numpy. (2024). Retrieved from The fundamental package for scientific computing with Python: https://numpy.org/

OpenIA. (n.d.). Retrieved from ChatGPT: https://openai.com/

Pandas. (2024). Retrieved from Pandas: https://pandas.pydata.org/

Pythonic Perabulations. (2024). Retrieved from In Depth: Principal Component Analysis: https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html

Scikit-Learn. (2024). Retrieved from Machine Learning in Python: https://scikit-learn.org/stable/

Seaborn. (2024). Retrieved from statistical data visualization: https://seaborn.pydata.org/