75.06/95.58
Organización de Datos
Primer cuatrimestre de 2018

# Trabajo Práctico Nº 2:
# Competencia de Machine Learning


| Integrantes | Padrón |
|---|---|
| Cozza, Fabrizio Luis | 97.402 |
| López Lecube, Lucio | 96.583 |
| Giannotti, Luciano | 97.215 |

# Manipulación de los datos

***CSVs:***

Al momento de juntar los datos se reveló una enorme cantidad de registros para analizar (como también un enorme tamaño de los archivos resultantes), por lo que en un principio se realizó una transformación de los tipos de datos numéricos para reducir el uso de memoria. Sobre este problema también se trabajó con aquellas columnas que no poseían valores numéricos, transformándolas en variables categóricas según correspondiese.

Luego, se transformaron aquellas columnas que tenían un formato determinado (como por ejemplo las fechas) pero no estaban determinados con su formato correspondiente en el dataframe.

A continuación, se verificó que no se produzca la aparición de valores nulos o con NaNs para no molestar a ciertos algoritmos de machine learning y se procedió a eliminar duplicados con cierto criterio (por ejemplo idpostulante e idaviso al mismo tiempo) para no perder información.

A partir del análisis exploratorio de datos se descubrió que en los archivos de avisos y postulaciones del set de datos podría llegar a haber información que se podría recuperar y llegar a ser útil para la parte del aprendizaje, por lo cual se procedió a intentar recuperar la mayor parte posible.

***Training set y testing set:***

Cómo los algoritmos de Machine Learning trabajan solo con variables numéricas se adoptaron dos tipos de posturas para el manejo de variables categóricas (strings):

- One Hot Encoding: que implica agregar columnas por cada variable categórica indicando con 1 o 0 si el registro cuenta con esa variable o no.
- LabelEncoder: que implica asignarle un valor numérico a cada variable categórica sin agregar nuevas columnas.

Si bien se intentó con diferentes posturas, posiblemente las más usadas fueron aquellas que fueron codificadas.

Aquellos a los cuáles no se les produjo codificación se les realizó una dummificación para ciertas columnas. En algunos casos sobre estos se obtuvieron samples en vez del set completo debido a que el procesamiento era muy largo e incluso inviable.

Finalmente, se realizó Train/Test Split y Cross validation para la reducción de overfitting.

# Algoritmos probados

*A continuación se muestra una lista de los algoritmos testeados a lo largo del trabajo práctico:*

*Más relevantes:*

- Random Forest (Regressor y Classifier)
- Extra Trees (Classifier)
- Gradient Boosting (Regressor y Classifier)
- Light GBM

- XGBoost (Classifier)
- MLP Classifier
- Adaboost

*Secundarios:*

- Linear Regression
- Ridge
- Ridge CV
- Lasso
- Elastic Net
- Lasso Lars
- Orthogonal Matching Pursuit
- Bayesian Ridge
- Huber Regressor
- Extra Trees (Regressor)
- Stochastic Gradient Descent

# Métodos de testeo utilizados

1. **Confusion matrix:** es una matriz de mxm, donde m es la cantidad de clases a predecir. Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real.
   Pueden darse 4 casos:
   a) True positives (TP), que son aquellos valores positivos y fueron clasificados como positivos.
   b) False positives (FP), que son aquellos valores negativos y fueron clasificados como positivos.
   c) False negatives (FN), que son aquellos valores positivos y fueron clasificados como negativos.
   d) True negatives (TN), que son aquellos valores negativos y fueron clasificados como positivos.
2. **Classification report:** permite visualizar la precision, recall y F1-Score de cada clase.
   a) Precision: se calcula cómo TP(TP + FP) y permite ver la cantidad de valores que son negativos y fueron clasificados como positivos, es decir los TN.
   b) Recall: se calcula cómo TP(TP + FN) y da una idea de la habilidad del clasificador para encontrar positivos.
   c) F1-Score: relaciona la precisión y el recall entre sí. El mejor valor es 1 y el peor 0.
3. **Accuracy Score:** para el problema binario indica la similitud de Jaccard entre las dos clases.
4. **ROC AUC:** Para combinar la Precisión y el Recall en una sola métrica, primero calculamos las dos métricas anteriores con muchos umbrales diferentes (por ejemplo, 0.00; 0.01, 0.02, ..., 1.00) para Logistic Regression por ejemplo, luego las trazamos en un solo gráfico, con los valores de Precisión en las abscisas y los valores de Recall en la ordenada. La curva resultante se llama curva ROC, y la métrica que consideramos es el AUC de esta curva, que llamamos AUROC. Se toma la curva de un predictor aleatorio que tiene un AUROC de 0.5. El predictor aleatorio se usa comúnmente como referencia para ver si el modelo es útil.

## Random forest

**Regressor:**
*Resultados:*
Mean squared error: 0.19485293319871383
Precission: 88.25 %

**Regressor + PCA:**
*Resultados*:
Mean squared error: 0.21324309259701693
Precission: 11.58 %

**Classifier:**
*Resultados*:
*Confusion matrix:*

| | |
|---|---|
| 722292 | 328803 |
| 351141 | 1429754 |

*Classification report:*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.67 | 0.69 | 0.68 | 1051095 |
| 1 | 0.81 | 0.80 | 0.81 | 1780895 |
| Avg / total: | 0.76 | 0.76 | 0.76 | 2831990 |

*Accuracy is:*
0.7599059318712283

*Area under the curve:*
0.745005

**Classifier + Random search:**
*Resultados*:
Precission: 0.7185498800631019

**Classifier + PCA*:***
*Resultados*:
Precission: 76.78 %

## Extra trees

**Classifier + Random search*:***
*Resultados*:
Precission: 0.999571502603188

## Gradient boosting

**Regressor:**

*Resultados*:
Mean squared error: 0.2556160523924702
Precission: 37.677365878911914 %

**Light GBM**
*Resultados*:
*Confusion matrix:*

| | |
|---|---|
| 612 | 1051796 |
| 214 | 1779368 |

*Classification report:*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.74 | 0.00 | 0.00 | 1052408 |
| 1 | 0.63 | 1.00 | 0.77 | 1779582 |
| Avg / total: | 0.67 | 0.63 | 0.49 | 2831990 |

*Accuracy is:*
0.1779980

*Area under the curve:*
0.500231

**XGBoost**
*Resultados*:
Accuracy : 0.6327
AUC Score (Train): 0.563374

**MLP Classifier**
*Resultados*:
Precission: 49.99 %

**MLP + down-sample majority class**
*Resultados*:
Precission: 49.97 %

**MLP sin PCA y todas las columnas:**
Mean squared error: 0.19009649511969026
Precission: 81.81 %

**Adaboost**
**Classifier:**
*Resultados*:
Mean squared error: 0.24437379665662057
Precission: 0.7674009138170442

**Secundarios**

Para los algoritmos secundarios se vió un rendimiento muy bajo por lo cual entran en esta categoría.

*Resultados:*

Precision = 0.35 % , name = LinearRegression
Precision = 0.35 % , name = Ridge
Precision = -78121918887.95 % , name = RidgeCV
Precision = 0.02 % , name = Lasso
Precision = 0.02 % , name = Elastic Net
Precision = 0.00 % , name = LassoLars
Precision = 0.19 % , name = OrthogonalMatchingPursuit
Precision = 0.35 % , name = BayesianRidge
Precision = -0.04 % , name = HuberRegressor
Precission: 1.474309678783925e-05 %, MSE: 0.23348..., name = ExtraTreesRegressor

## **Algoritmo final utilizado:** Gradient Boosting Classifier + SMOTE

*Confusion matrix:*

| | |
|---|---|
| 29373 | 26289 |
| 43416 | 50922 |

*Classification report:*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.40 | 0.53 | 0.46 | 55662 |
| 1 | 0.66 | 0.54 | 0.59 | 94338 |
| Avg / total: | 0.56 | 0.54 | 0.54 | 150000 |

*Accuracy is:*
0.5353

*Area under the curve:*
0.533743

### ***Score en Kaggle:*** *0.59059*

## “Problemas” y “soluciones”:

- Alta cantidad de registros para entrenamiento:
    - “Solución 1”: PCA (Principal component analysis) es una técnica utilizada para describir un set de datos en términos de nuevas variables ("componentes") no correlacionadas. Los componentes se ordenan por la cantidad de varianza original que describen, por lo que la técnica es útil para reducir la dimensionalidad de un conjunto de datos.
    - “Solución 2”: Feature Importance tomando aquellos n mejores valores aprendidos por el algoritmo de RandomForest al ser entrenado con todas las columnas.
    - “Solución 3”: utilizar el método de pandas .corr() para analizar la correlación de todas las columnas con respecto a sepostulo.
- Clases no balanceadas: al tener más vistas (sepostulo = 0) que postulaciones (sepostulo = 1) contamos con un desbalance de clases que puede implicar un cierto sesgo para el clasificador.
    - “Solución 1”: Over-Sampling técnica que implica aumentar el número registros de la clase minoritaria hasta igualar al número de la clase mayoritaria utilizando SMOTE y Resampling.
    - “Solución 2” Under-Sampling técnica que implica disminuir el número de registros de la clase mayoritaria hasta igualar al número de registros de la clase minoritaria utilizando Resampling.

## Análisis general y conclusiones

Teniendo en cuenta el *No Free Lunch Theorem* se intentó de tratar el problema propuesto con la mayor cantidad de algoritmos posibles y las diferentes variables que se podrían producir sobre ellos dentro de un tiempo de procesamiento razonable.

Esto lleva a realizar un análisis sobre la manipulación de datos. Nuestro principal objetivo inicial fue el de tener la mayor cantidad de información posible, tratando tanto de conservar la mayor parte posible de los datasets como también recuperar de ellos información que se considerase relevante desde cierto punto de vista a partir de los títulos y descripciones, esto debido a que no se notó una gran cantidad de datos que podrían ser anómalos desde el análisis exploratorio como para poder realizar un filtrado.

Este planteo fue contraproducente al momento de entrenar los datos, primero debido a que el tiempo de procesamiento se incrementó, haciendo ineficiente la búsqueda de hiper parámetros y produciendo que esta sea muy acotada, ya que en memoria no se soportaba la prueba de muchos parámetros al mismo tiempo. Luego esto produjo una reducción en el porcentaje de entrenamiento a realizarse, debido a tener que esperar un tiempo considerable para ver una mejora sobre cada algoritmo y en algunos casos obligando a tener que sacrificar datos agarrando muestras para poder correr ciertos algoritmos. Por último se vió en algunos algoritmos la clara aparición de overfitting ya que ciertos algoritmos predecían de manera casi óptima el set de entrenamiento pero tendían a producir pobres resultados al momento de predecir el set de test, también podría considerarse que se produjo overfitting por la complejidad del modelo de datos. Intentar arreglar este problema ya sea con el cambio de parámetros como también con cross validation no ultimó en nada útil.

Considerando todo esto y viendo nuestro resultado final de la competencia podría concluirse que lo que se podría haber cambiado principalmente es la manipulación presurosa de los datos, ya que es improbable no haber logrado ningún resultado fructuoso a partir de ningún algoritmo, teniendo también en consideración las variantes mencionadas que se produjeron sobre el training set.

Se puede decir en líneas generales que los algoritmos que mejor rindieron para el problema fueron los de Random Forest y Gradient Boosting.

Quizás fue precipitado depositar la confianza en la cantidad de algoritmos considerando el análisis de datos ya realizado y tenido en cuenta, pero es una lección a tener en cuenta.

**LINK a Github:** ***https://github.com/lucioll/NaventDatosTP***

## Métodos de testeo utilizados

## Bibliografía


- **Regression**
- (http://scikit-learn.org/stable/supervised_learning.html#supervised-learning)
- **Classification**
- (http://scikit-learn.org/stable/supervised_learning.html#supervised-learning)
- **Preprocessing**
- (http://scikit-learn.org/stable/modules/preprocessing.html#preprocessing)
- **Model selection**
- (http://scikit-learn.org/stable/model_selection.html#model-selection)
- **Dimensionality Reduction**
- (http://scikit-learn.org/stable/modules/decomposition.html#decompositions)
- **XGBoost: variables categoricas dummification vs enconding**
- (https://stackoverflow.com/questions/34265102/xgboost-categorical-variables-dummification-vs-encoding)
- **XGBoost: data preparation**
- (https://machinelearningmastery.com/data-preparation-gradient-boosting-xgboost-python/)
- **XGBoost: api docs**
- (https://xgboost.readthedocs.io/en/latest/build.html)
- **XGBoost: parameter tuning**
- (https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/)
- **XGBoost: teoria + parameter tuning**
- (https://www.slideshare.net/ShangxuanZhang/kaggle-winning-solution-xgboost-algorithm-let-us-learn-from-its-author)
- **XGBoost: Random search**
- (http://danielhnyk.cz/how-to-use-xgboost-in-python/)
- **XGBoost + pandas**
- (https://jessesw.com/XG-Boost/)
- **XGboost: early stopping**
- (https://machinelearningmastery.com/avoid-overfitting-by-early-stopping-with-xgboost-in-python/)
- **XGBoost: uso de gamma para reducir overfitting**
- (https://medium.com/data-design/xgboost-hi-im-gamma-what-can-i-do-for-you-and-the-tuning-of-regularization-a42ea17e6ab6)
- **LightGBM: documentacion**
- (http://lightgbm.readthedocs.io/en/latest/
- https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/
- https://www.analyticsvidhya.com/blog/2017/03/imbalanced-classification-problem/
- https://www.neuraldesigner.com/blog/methods-binary-classification
- https://machinelearningmastery.com/feature-selection-machine-learning-python/
- https://elitedatascience.com/dimensionality-reduction-algorithms
- http://dataaspirant.com/2017/02/13/save-scikit-learn-models-with-python-pickle/
- https://github.com/jbn/nbmerge

- https://lightgbm.readthedocs.io/en/latest/Parameters.html
- https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc)
- **Random Forest**
- (https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74
- https://medium.com/@taplapinger/tuning-a-random-forest-classifier-1b252d1dde92
- https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/)
- **Gradient boosting**
- (https://medium.com/@mohtedibf/in-depth-parameter-tuning-for-gradient-boosting-3363992e9bae
- https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/)
- **Guardar modelos**
- (http://dataaspirant.com/2017/02/13/save-scikit-learn-models-with-python-pickle/)