




Organización de Datos 75.06/95.58
Trabajo Práctico N° 2

GRUPO: 1

Apellido y Nombre	Padrón	Correo electrónico
Funes Federico	98372	fede.funes96@gmail.com
Caceres Julieta	96454	julieta.agustina.caceres@gmail.com
Lasanta Tito	99162	titonoolvida@gmail.com
Almejún Santiago	99441	santi.alme@gmail.com

Github: 

<https://github.com/fedefunes96/TpDatos>

Índice

1. Introduction	2
2. Análisis de datos	2
3. Limpieza	2
4. Feature engineering	3
5. Set de entrenamiento	3
6. Estimación de datos incompletos	4
7. Búsqueda de hiperparámetros	4
8. Algoritmos utilizados	4
9. Conclusiones	7

1. Introduction

El objetivo de este informe es mostrar los resultados de aplicar machine learning al set de datos provisto por la empresa Navent para estimar la probabilidad de que una persona que está registrada en su dominio "https://www.zonajobs.com.ar/" se postule a un cierto aviso que figure en la página.

Todo el proyecto fue realizado en Python utilizando las siguientes librerías:

- Pandas: Para trabajar sobre los datos y organizarlos de manera que se puedan usar los mismos en algoritmos machine learning.
- Numpy: Para obtener ciertas funciones que necesitábamos, por ejemplo, funciones para calcular los NaNs que había en el dataset incompleto.
- Scikit-learn: Librería que contiene los algoritmos de machine learning que se utilizaron.

2. Análisis de datos

Utilizando los resultados que obtuvimos del trabajo práctico anterior, se nos facilitó descubrir los features de mayor importancia en los datos provisto por el otro lado, encontrar nuevos features que permitieron mejorar drásticamente los resultados de nuestros algoritmos de machine learning fue uno de los mayores desafíos. Un detalle que ocurrió, fue que los nuevos datos provistos por Navent no tenían, ni postulaciones, ni la información de si estuvieron en línea, lo que limitó la cantidad de features que se pudo extraer de información para poder realizar una predicción efectiva en la competencia de kaggle.

3. Limpieza

Lo primero que se hizo fue crear un set de datos de entrenamiento para nuestros algoritmos de machine learning, por lo cual fue necesario eliminar aquellos features que no aportaban nada para el entrenamiento (Por ejemplo: País, ya que todos los avisos eran de Argentina), así como también los respectivos outliers de los features que finalmente utilizamos. También, eliminamos aquellos features que provocaban que hubiera overfitting, siendo los mismos fechas específicas de postulaciones y vistas.

4. Feature engineering

Una vez creado el set de datos de entrenamiento, lo primero que hicimos con el mismo fue probarlo para ver su resultado con un algoritmo de machine learning (En este caso, un clasificador basado en árboles de decisión usando cross validation con 5-Folds), una vez realizado esto, analizamos la importancia de cada feature (Información provista por el clasificador) para ver sobre cuales features debíamos enfocarnos. Luego, propusimos agregar features para ver como variaban los resultados por el clasificador, obteniendo los mismos, por ejemplo, a partir de las descripciones y títulos de los avisos publicados.

Los features que se agregaron fueron los siguientes: Fechas iniciales/finales de actividad en avisos Fechas iniciales/finales de actividad de postulantes cantidad de vistas de cada aviso y de cada postulante cantidad de veces, que un postulante vio un aviso varios derivados de la descripción de anuncios

Vimos que el agregado de features mejoró drásticamente el resultado de nuestro algoritmo final, permitiéndonos pasar de un 77 % de score basado en cross-validation de 5-Folds a un 94 %.

5. Set de entrenamiento

Para entrenar a nuestro algoritmo, utilizamos los datos provisto hasta antes del 15 de abril, ya que estos son los que efectivamente tenían la información de las postulaciones. Una vez agregados los features nuevos, notamos que una gran cantidad de los mismos tenían valores NaN por parte de las vistas (Esto se debe a que faltaban datos sobre las vistas), así que utilizamos los datos de las postulaciones en esas fechas faltantes para estimar la cantidad de vistas que hubo para cada aviso y las que realizó cada postulante.

Finalmente, como el set de entrenamiento poseía aproximadamente 6 millones de datos, optamos por utilizar no mas de un 25 % de la data aproximadamente, ya que de otra manera el proceso hubiese tardado mucho para la búsqueda de hiperparámetros y el consumo de memoria hubiese sido demasiado grande. Como necesitabamos datos para predecir las no postulaciones, decidimos rellenar el dataset con una cierta cantidad de no postulaciones generadas por nosotros, haciendo variar esta en relación a la cantidad de postulaciones efectivamente realizadas.

6. Estimación de datos incompletos

Como el set de datos provisto estaba incompleto, hubo que realizar estimaciones para varios features (Aproximadamente el 15 % de los datos a predecir tenían datos faltantes como por ejemplo un anuncio que no fue visto, o un postulante sin genero). Para solucionar este problema tomamos dos rutas:

- Estimar los features: Para determinar que features requerían una mejor estimación, nos basamos en la importancia de los features que nuestro algoritmo calculó. Los features menos importantes los rellenamos utilizando el promedio con respecto a ese feature para los demás avisos/postulantes, en cambio, los features más importantes intentamos estimarlos utilizando un algoritmo de machine learning, pero vimos finalmente que no había gran diferencia con estimarlo utilizando el promedio también.
- Estimar la predicción directamente: debido a que los datos faltaban en si es informacion, intentamos rellenar las predicciones que no lograbamos debido a no tener los datos.

7. Búsqueda de hiperparámetros

Para la búsqueda de hiperparámetros de cada algoritmo que usamos, utilizamos de la libreria scikit-learn la clase GridSearchCV, la cual nos devuelve un conjunto de hiperparámetros de ciertas combinaciones de hiperparámetros que nosotros le dimos al mismo para que calcule cuales son los mejores para nuestro problema de machine learning utilizando cross-validation.

8. Algoritmos utilizados

- Decision Tree: Nuestro primer algoritmo utilizado y sobre el cual nos basamos para ir viendo las mejoras de nuestro set de entrenamiento. No logramos la mejor precisión para la competencia pero, al ser rápido, nos permitió probar varias combinaciones de hiperparámetros y hacer pruebas más seguido.
- Naive Bayes: Este algoritmo fue muy sencillo de aplicar y funcionaba muy rápidamente (ya que se adapta particularmente bien a grandes

cantidades de datos) pero lamentablemente no dio resultados favorables. A pesar de probar con distintas cantidades de información y relaciones unos a ceros, no logró superar los resultados de otros algoritmos, como Random Forests.

- GradientBoosting: Probamos utilizar este algoritmo y fue uno de los que peor precisión tuvo para nuestro problema además de su elevado tiempo de ejecución. Lo descartamos de inmediato.
- Adaptive Boosting: Al igual que Gradient Boosting, AdaBoost no solo tenía elevados tiempos de ejecución aún con poca data, sino que no proporcionaba resultados contundentes. También fue descartado inmediatamente.
- Bagged Decision Trees: Este fue uno de los primeros ensambles que se probó. Al utilizar el concepto de bagging con árboles de decisión, se obtuvieron buenos resultados: se logró un 70 % de precisión como mínimo, pero no resultó ser lo suficientemente bueno como para lograr resultados superiores al 90 %.
- RandomForest: Fue el que mejor se adaptó a nuestro modelo dándonos la mayor precisión de todos y el que finalmente utilizamos en la competencia, obteniendo un resultado de 94 % para nuestro set de entrenamiento.
- Extra Trees: Este algoritmo es muy similar a RandomForests, pero consiste en árboles extremadamente aleatorizados donde tanto el atributo como la condición de corte en un nodo no son seleccionados óptimamente, sino al azar. De este modo, aumenta la precisión y la eficiencia del algoritmo. Se lograron resultados similares a los de RandomForest, aunque no tan buenos.
- SVD más algoritmos: Lo primero que se hizo visualizar la cantidad de información que se perdía al reducir dimensiones:

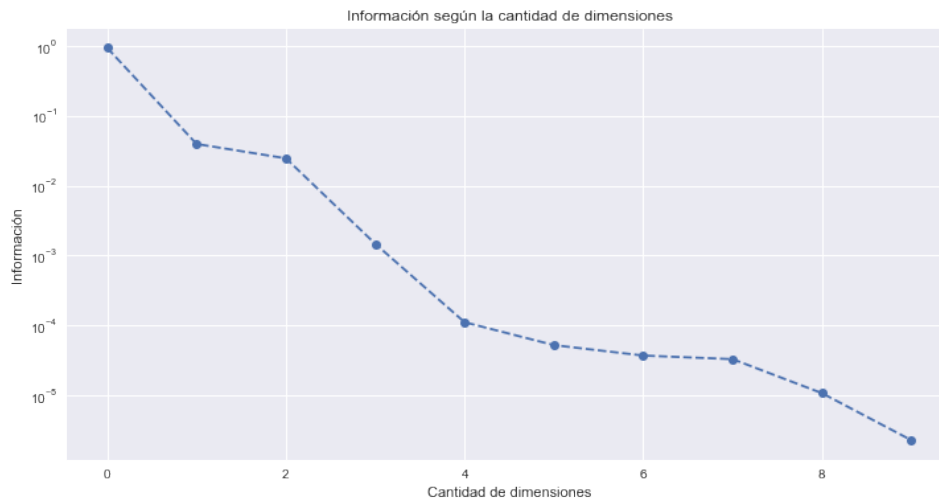


Figura 1: Información al reducir dimensiones

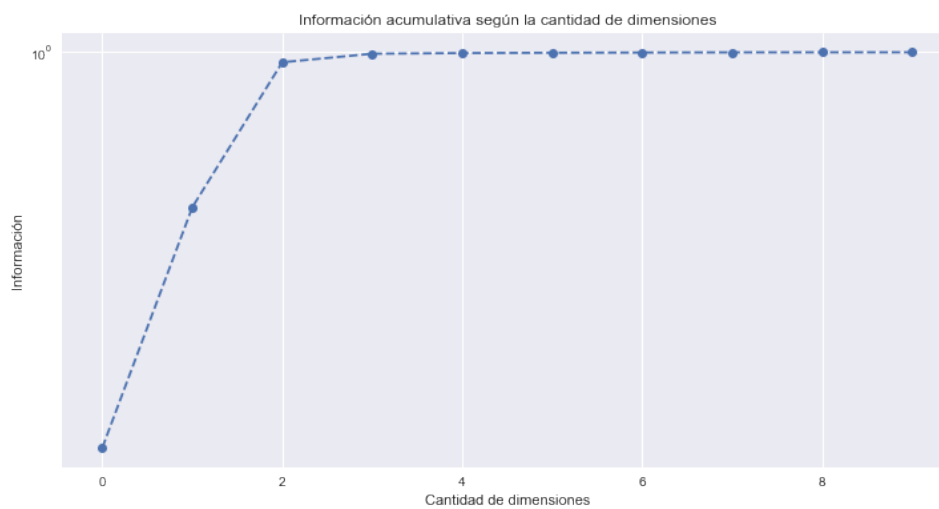


Figura 2: Información acumulativa al reducir dimensiones

Con lo que vimos que con 4/5 dimensiones acumulamos aproximadamente el 0.99 % de la información.

- LogisticRegression: Ejecución rápida y se adaptaba muy bien al set de entrenamiento (88 % precisión) pero dejó mucho que desear para la precisión en el set de pruebas (Decayó a un 78 %).
- LinearSVC: Igual que con LogisticRegression, se adaptó muy bien al set de entrenamiento (Cercano a un 88 %) pero predecía bastante mal.

Concluimos que reducir dimensiones no nos ayudó a mejorar nuestra

precisión pero nos dio una noción de la cantidad de dimensiones y la importancia de los features.

9. Conclusiones

Finalmente utilizamos RandomForest para la competencia de kaggle, dándonos para nuestro modelo un 95 % de precisión usando cross-validation de 5-Folds, la cual consideramos que es una precisión bastante alta, mientras que en la competencia este bajó a un 90 %, siendo la causa de esto la estimación de los datos faltantes en el set de datos. Podríamos concluir que nuestro algoritmo hubiese mejorado aún más mejorando la calidad de los features a utilizar, usando una mejor estimación para los valores que no fueron provistos y modelando mejor los no postulados que utilizamos para el set de entrenamiento.