




FIUBA – PRIMER CUATRIMESTRE 2018

7506 – ORGANIZACIÓN DE DATOS

TRABAJO PRÁCTICO – PARTE 2

GRUPO 29

MARTÍN, MARIANO – APARICIO, AXEL



Introducción

Objetivo

Dados los datos acerca de postulaciones a avisos laborales provistos por la cátedra, entrenar un modelo de machine learning capaz de predecir la probabilidad de que los postulantes detallados en el archivo `test_final_100k_` (también provisto por la cátedra) se postulen a los avisos detallados en el mismo.

Enfoque y resolución del problema

El flujo de trabajo que desarrollamos a lo largo de la duración del Proyecto puede dividirse en tres etapas principales:

- 1) Preprocesamiento de los datos.
- 2) Entrenamiento de varios modelos de machine learning.
- 3) Selección de los modelos con los que mejores resultados tuvimos.

A continuación, haremos un detalle de cada una de estas etapas.

Primera etapa: Preprocesamiento de los datos

La primera tarea que realizamos es la que más tiempo llevó: limpiar los datos de anomalías (por lo general campos vacíos o sin sentido) y seleccionar cuántos y con cuáles quedarnos para armar nuestro set de entrenamiento.

En un principio seleccionamos los datos de todas las postulaciones registradas en el set de datos. Pronto nos encontramos con la primera cuestión a resolver: así como teníamos casos de candidatos que decidieron postularse, íbamos a necesitar también casos de candidatos que decidieron no postularse a algún aviso.

Lo primero que se nos ocurrió fue simplemente tomar al azar pares postulante-aviso, teniendo cuidado de que ninguno de estos pares se encuentre entre los datos de postulaciones realizadas. Sin embargo, pronto nos dimos cuenta de lo poco conveniente que era este enfoque: tomar pares al azar no era para nada representativo, tal vez el candidato tomado al azar no se postuló al aviso dado simplemente porque no tuvo oportunidad de verlo. Teniendo esto en mente, decidimos tomar como casos de no postulación a los candidatos que, habiendo visto un determinado aviso, no registraron una postulación en él.

Una vez obtenidos los registros que necesitábamos, lo que faltaba era definir con cuáles features de los mismos quedarnos.

Luego de hacer algunas pruebas con predictores básicos, decidimos quedarnos con el siguiente conjunto de features:

- **Edad:** Edad del postulante
- **Longitud_descripcion:** Cantidad de caracteres que posee la descripción del aviso
- **Cantidad_vistas:** Por cuántos candidatos fue visto el aviso
- **Denominacion_empresa:**Cuál es la empresa detrás del aviso
- **Tipo_de_trabajo:** Carga horaria
- **Longitud_titulo:** Cantidad de caracteres que posee el título del aviso
- **Nombre_area:** Área laboral
- **Cod_edu:** Grado de educación del postulante
- **Cod_est:** Estado en el que está la educación del postulante
- **Cod_sexo:** Sexo del postulante
- **Nivel_laboral:** Nivel de experiencia del perfil buscado por el anuncio

Segunda etapa: Entrenamiento de modelos

Trabajamos con los siguientes modelos:

- **Árbol de decisión**: Elegimos este algoritmo para dar los primeros pasos en la resolución del problema ya que, si bien no es demasiado poderoso por sí solo, se puede entrenar rápidamente y nos permitió probar varias cosas para hacernos una idea de cómo se comportaban las predicciones.
- **KNN**: Siempre hay que revisar KNN antes de aventurarse a probar otros métodos más sofisticados. Obtuvimos un resultado decente en nuestro notebook con un tiempo de entrenamiento un poco más elevado que el del árbol de decisión. Probablemente lo tengamos en cuenta para un futuro ensamble, pero no nos resulta de gran ayuda por separado.
- **Bagging**: Usando como predictor base al árbol de decisión que entrenamos, ya empezamos a obtener predicciones mucho mejores. Luego de realizar un random search sobre el árbol base, obtuvimos hiperparámetros que mejoraron el rendimiento y nos permitieron llevar la predicción de bagging a un nada despreciable 71% en Kaggle.
- **Random forest**: Random forest es una mejora del bagging con árboles que además admite paralelización, disminuyendo drásticamente el tiempo de entrenamiento. Luego de los buenos resultados que obtuvimos con bagging, claramente quisimos probar este algoritmo y no decepcionó: mejoramos la predicción a 73% de acierto.
- **XGBoost**: Probamos XGBoost en gran medida debido a la buena fama que tiene: es uno de los algoritmos preferidos en las competencias de Kaggle. Sin hacer prácticamente un refinamiento de los hiperparámetros obtuvimos buenos resultados.
- **Adaboost**: Armamos adaboost usando como base el árbol de decisión que teníamos. Si bien este método mejoró la precisión que el árbol obtenía, no era una mejora respecto a los observados en random forest.

Eso, sumado a que su elevado tiempo de entrenamiento comparado con RF, hizo que no profundicemos mucho más en él.

- **Otras alternativas exploradas:** Además de los algoritmos detallados, fueron tenidas en cuenta también los siguientes algoritmos: naive bayes, lineal regression, svm y stacking.

Conclusiones: Mejores modelos

Los mejores resultados en la competencia de kaggle los hemos conseguido con los ensambles de random forest y xgboost.

En el caso de random forest, conseguimos los mejores resultados con estos hiper parámetros:

- `n_estimators=22,`
- `min_samples_split = 100,`
- `criterion = 'entropy',`
- `max_depth = 70,`
- `min_samples_leaf = 20,`
- `warm_start = True,`
- `oob_score = True,`
- `bootstrap = True,`
- `max_features = 'sqrt'`

En cuanto a xgboost, los mejores resultados se obtuvieron con:

- `n_estimators=22,`
- `max_depth = 70,`
- `learning_rate = 0.3`

