HW1 - Classification models in sklearn

You'll be building a few classifier models and using some of the tech tools we learned about in Modules 1 and 2.

The Raw Data

The data is the the KC housing data. **I've made all the necessary data files available to you in the assignment folder.**

Kaggle source: <https://www.kaggle.com/harlfoxem/housesalesprediction>

Basic data dictionary

[https://geodacenter.github.io/data-and-lab//KingCounty-HouseSales2015/](https://geodacenter.github.io/data-and-lab/KingCounty-HouseSales2015/)

Link to discussion item meaning of CONDITION and GRADE fields:

<https://www.kaggle.com/harlfoxem/housesalesprediction/discussion/141767>

Preliminary Data Prep

In order to use this data for a classification problem, I did some data prep work. Our target variable is a new variable that I created called price\_gt\_1M which is a binary variable:

* 1 - house price is greater than or equal to 1 million dollars
* 0 - house price is less than a million dollars

The data for this classification problem can be found in ./data/kc\_house\_data\_classification.csv.

If you want to see my data prep code, see the hw1\_sklearn\_dataprep.ipynb notebook.

The Problem

Our overall goal is to build classifier models to predict price\_gt\_1M using the the other variables. You must use sklearn Pipelines that contain your preprocessing steps and your model estimation step. We did this in the class notes.

You should do your work in a Jupyter Notebook.

Task 1 - Folder structure

Start by creating a new project folder structure with the cookiecutter-datascience-simple template that I covered in Module 1. Put the data files into an appropriate folder and put this notebook in the main project folder. Any additional notebooks and/or Python files you end up creating should go in the main project folder.

Task 2 - Version control

Put your new project folder under version control using git. You should **NOT** track the data file. You must track all notebooks, Python scripts or additional text files you end up creating. Put appropriate information into your readme file.

Task 3 - EDA

I suggest you start by reading the csv file into a pandas dataframe. I called my dataframe, housing\_df. Then start with some basic EDA. You can certainly use automated tools such as pandas-profiling, skimpy or SweetViz as I showed in the class notes. Remember, when you run some of those tools, you **must** have your notebook open in the classic Jupyter Notebook interface (and **NOT** in Jupyter Lab) Check their docs to see if Jupyter Lab is supported yet. I pip installed SweetViz and it seems to be working fine now with Jupyter Lab. As we've seen, the reports get created as HTML documents. These should go in your output folder within your project.

Task 4 - Categorize feature types

As we did in the Pump it Up class notes, we are going to need to create a list of categorical variables and a list of numeric variables so that we can apply the appropriate pre-processing to each. In the notes we used the data type of the columns to create lists of numeric and categorical variables. That's not necessarily going to work here as all the variables will come in as numeric. So, you'll have to come up with another way to create lists of the categorical variables and the numeric variables.

Since we are using regularization, all of the numeric variables will need to rescaled using the StandardScaler. You'll do this later as part of the Pipeline. For any variables that you decide should be treated as categorical in your models, use the OneHotEncoder on them in the preprocessing stage.

Be careful, just because a variable has a numeric datatype in the pandas dataframe, it does **not** mean that it's necessarily a numeric variable in the context of the classification models. Think about each column and look at your EDA reports and decide whether or not it's truly numeric or needs to be treated as categorical data in the models.

Even though our target variable, price\_gt\_1M, is categorical, you do **NOT** need to do any preprocessing on it. As I mentioned in our class notes, scikit-learn will automatically detect that and will do any encoding needed on its own.

Finally, you'll partition the dataset into training and test datasets for modeling:

* I broke up the housing\_df into two separate dataframes that I called X and y, to use in the models. Here's my code for that:

X = housing\_df.iloc[:, 0:18]

y = housing\_df.iloc[:, 18]

* Please use the following code for your data partitioning so that we all end up with the same training and test split:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=73)

Task 4 - Logistic regression models

You are going to build a few different logistic regression models using all of the variables in our housing dataset. For each of these models you will:

* Create a pipeline to do the preprocessing (the scaling and encoding) and the modeling (we did this in the Pump it Up project)
* I'll be giving you different specifications and hyperparameter parameter settings to try
* You'll be scoring the models on overall accuracy for both the training and test data. Discuss any evidence of overfitting or underfitting as well as how the model does in comparison to the null model.
* There will be some additional tasks/questions for each model - details below

**IMPORTANT** You always should put summary comments in one or more markdown cells. Do **NOT** write them as comments in a code cell. The whole point of Jupyter notebooks is to be able to mix markdown cells with code cells. Yes, you should also include code comments but those are different than analysis comments.

Model 0: The null model

We always start with the simplest possible model and we call it the *null model*. For binary classification models, the null model is usually just to predict that each observation will fall into whichever class is most prevalent. In other words, what would be the performance of a model in which we just predict 0 for everyone? Scikit-learn has a way to create simple null models for classification with the sklearn.dummy.DummyClassifier class. See <https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html> for the details. You must use this class to create your null model.

Model 1: Ridge regression with C=1.0

Build a ridge regression model to predict price\_gt\_1M and use the default value of C=1.0. I used the following additional options with the LogisticRegression model - solver='saga', max\_iter=2000. Feel free to change these if you want. AFter fitting the model, compute its accuracy score for training and test and write out a little summary (f-strings are useful). Here's an example:

Training score: 0.974

Test score: 0.971

Create confusion matrices for both training and test.

Also, create a plot of the coefficients (as we did in the notes). If you want to use that coef\_plot function we used in the notes, you'll have to make a few modifications because we only have one set of coefficients (since we have a binary classification problem as opposed to a 3-class problem in Pump it Up).

Model 2: Lasso regression with C=1.0

Same as Model 1, but use lasso regression instead of ridge regression. Create the same outputs and compare the performance to the ridge regression model.

Model 3: Lasso regression with C=0.01

Fit another lasso regression but use C=0.01. Does this enforce more or less regularization? Create the same outputs and compare the performance to the first two models. Discuss why the plot looks so different than the previous plots.

Model 4: Lasso regression with optimal C value

Now use LogisticRegressionCV to fit a model and let sklearn determine the optimal C value to use. Again, compute the accuracy score and confusion matrices. Also, print out the optimal value of C. Does regularization help for this problem?

Task 5 - Simple decision tree

Now fit a decision tree to predict price\_gt\_1M. As we did above, for both train and test, compute the accuracy score, create a confusion matrix, and discuss the performance relative to your logistic regression models. Obviously you do not need to create a coefficient plot (why not?).

**HACKER EXTRA:** See if you can figure out how to display the fitted tree so that it's readable. :)

Task 6 - a little error exploration (more challenging)

This will challenge your pandas and your general data manipulation skills. Just give it your best shot. If you don't get, you don't get it. It doesn't require much code - just the right code. :)

I also include another data file called kc\_house\_data\_regression.csv in which the target variable is price. Everything else is exactly the same, including the order of the rows. So, here's your challenge. Using Model 2 (the lasso model with C=1.0), start by using the predict method to generate an array of predictions for the original test data. Obviously, some of the predictions are correct and some of them are not. It would be interesting to know more about the kinds of errors our model is making. We can see some things from the confusion matrix. However, since we don't have the actual price value, it's hard to visualize how the errors relate to it. For example, are we only making errors when the price is really close to 1 million? One way to visualize this is to create a histogram of the actual prices **only for those rows in test that we got wrong**. What makes this tricky? A few things:

* As I already mentioned, price is not in our original data but is in the kc\_house\_data\_regression.csv dataset. Remember, other than the target variable, this dataset is identical (including the index) to the one we used above for classification.
* We partitioned the classification dataset into training and test datasets.
* In order to create the histogram, we simply need a Series (or array) of price values corresponding to the predictions in test that we got wrong.

**HINTS**

* The pandas join method will come in handy.
* The pandas .loc selector can take a boolean array as an input for selecting rows or columns. Using one to select rows is quite useful for this problem.

Optional Hacker Extra Credit tasks

I always like to include some extra credit tasks for those who want to push themselves a little further. For this problem, consider doing one or more of the following:

* Try out the [Histogram based Gradient Boosting Classifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html?highlight=histogram%20based%20gradient%20boosting%20classifier) shown in the optional materials at the end of Module 2. Compare its performance to logistic regression and the random forest.
* I also include another data file called kc\_house\_data\_regression.csv in which the target variable is price. Use sklearn's LassoCV to find a good regression model for predicting price.

Deliverables

You should simply compress your entire project folder as either a zip file or a tar.gz file (do **NOT** ever use WinRAR to create rar files in this class). Note that when you do this, your "hidden" .git folder will get included. So, I'll be able to tell that you put the project under version control and I'll be able to look at your project folder structure. Before compressing the project folder to submit it:

* make sure all of your notebooks and other files are in the main project folder and have good filenames,
* make sure you've committed all of your changes (git),
* upload your compressed folder in Moodle.

In [ ]: