|  |  |  |
| --- | --- | --- |
| Understand data | Make first observation | 1 Passengerid is useless  2 Extract name info  3 Sex convert to numeric value  4 Missing age  5 Ticket is useful?  6 Carbin too many null, may drop this info  7 Embark has null  8 Embark convert to numeric value  9 Fare needs to be grouped  10 Age needs to be grouped |
|  | Use python to read basic data stats |  |
|  | Use tableau to visualize data from first observation | <https://public.tableau.com/profile/kenny2174#!/vizhome/Kaggle_titanic_15746524123480/agebin> |
| Feature engineering plan | List to-do | Train  7 Fill in missing Embark  Test  12 Fill in missing Fare  Both  1 Remove passengerid  2 Extract name info (Mr. Miss.)  3 Sex convert to numeric value (0 male, 1 female)  4 Fill in missing age  5 Remove tickets  6 Remove Carbin  8 Embark convert to numeric value (0S 1C 2Q)  9 Group fare (bin size )  10 Group age (bin size )  11 FamilySize/IsAlone  13 nameLen |
| Address issues in to-do list | Address issues in to-do list | 2 get Title first using Regex then convert it to ordinal value  title\_mapping = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Rare": 5}  4 More accurate way of guessing missing values is to use other correlated features. In our case we note correlation among Age, Gender, and Pclass. Guess Age values using median values for Age across sets of Pclass and Gender feature combinations. So, median Age for Pclass=1 and Gender=0, Pclass=1 and Gender=1, and so on…  10 put age into 5 bins in the range of min/max, then convert age to ordinal value based on age bins  9 and 12 Used median then convert fare to ordinal value (hard-coded 7.91/14.454/31)  7 only 2 missing Embarked in train\_df, simply replace them with the most frequent one  11 FamilySize = SibSp + Parch; IsAlone is based on FamilySize, choose to keep SibSp/Parch |
| Implement feature engineering |  |  |
| Model | a classification and regression problem |  |
|  |  |  |

Starter: <https://www.kaggle.com/startupsci/titanic-data-science-solutions>

Stacking: <https://www.kaggle.com/arthurtok/introduction-to-ensembling-stacking-in-python>

Feature engineering: <https://www.kaggle.com/gunesevitan/advanced-feature-engineering-tutorial-with-titanic>

Ultimate: <https://www.kaggle.com/ldfreeman3/a-data-science-framework-to-achieve-99-accuracy>

Advanced feature engineering lacks one-hot vector

Terms:

Ordinal: 1, 2, 3

Nominal: name, text

Quantitative: age and fare variable are continuous

Categorical: sex

|  |  |
| --- | --- |
| Understand data | Head  Tail  Info  get\_null\_percentage(df):  describe  display\_missingness(dfs): // better than get\_null\_percentage(df) |
| See correlation | Tableau  Line chart for trend  Pearson\_Correlation\_of\_Features for cross-variable correlationcorrelation(df, col, abs): *# find absolute value of correlation between given col and the rest of columns, missing values are ignored, only for numerical, suitable for finding correlation in order to decide how to fill in missing-ness*  Data online should also be used  concat\_df(df\_train, df\_test): **stacked df** of training and test set on axis 0 (vertically) # Advanced\_feature\_engineering\_refactored.py |
| Feature Engineering (combine=[train, test] array is passed by reference, it will update array itself, train, test)  (train/test is passed by copy, no update) | Add cols  Drop cols  Categorize into several big groups  categorical\_to\_ordinal (map non-numerical to numerical)  Fill missing data (use closely correlated vars to get the mean/median)  Group continuous data into bins, then use bin to convert to ordinal  caregorize titles to Rare/Miss/Mrs  Manipulate data in df, train/test model in ndarray |
| Stacking |  |
|  |  |
|  |  |
| Post analysis | feature\_importance |

Dfs: []

|  |  |
| --- | --- |
| Advanced feature engineering | Basic feature engineering |
| display\_missingness | get\_null\_percentage |
| add\_title\_col | add\_title\_col |
| caregorize\_title | categorize\_title |
| categorical\_to\_ordinal | categorical\_to\_ordinal |
| create\_FamilySize | create\_FamilySize |
| create\_ticketFreq |  |
| create\_IsAlone | create\_IsAlone |
| convert\_age\_to\_ordinal\_based\_on\_bins | convert\_age\_to\_ordinal\_based\_on\_age\_bins |
| convert\_fare\_to\_ordinal\_based\_on\_bins | convert\_fare\_to\_ordinal\_based\_on\_fare\_bins |
| add\_name\_length | add\_name\_length |
|  |  |
| correlation |  |
| fill\_missing\_age | fill\_missing\_age |
| fill\_missing\_embarked | fill\_missing\_embarked |
| fill\_missing\_fare | fill\_missing\_fare |
| create\_deck |  |
| fill\_missing\_embarked | fill\_missing\_embarked |
|  | group\_age\_into\_bins |
|  | group\_fare\_into\_bins |

df

|  |  |
| --- | --- |
| advanced |  |
| drop\_col | drop\_col |

kernal\_achieve\_99\_accuracy\_clean.py

|  |
| --- |
| save\_df\_to\_csv(df, outFileName): |
| unique\_values\_in\_column(dfs, ceiling): *a helper function for categorical\_to\_ordinal* |
| categorical\_to\_ordinal(data\_cleaner): built-in method, don’t need manual mapping |
| data1\_dummy = pd.get\_dummies(data1[data1\_x]): *# having one-hot effect for categorical data, keep numeric unchanged,* will skip the non-appearing values. Eg, a/b/c/e without d |
| multi\_classifier\_model\_compare(): use table to compare an array of classifiers, get accuracy on train, get predications with all classifiers and average out to get final prediction  MLA\_compare.loc[row\_index, **'MLA\_Test\_Accuracy\_3\*STD'**] = cv\_results[**'test\_score'**].std() \* 3: within +/-3 std, how much more precision can be improved |
| cross\_validation\_split\_2(): in competition, labelled data should not fully be used as train, we should *split training dataset into a:b:c subsets and return index of those subset, run model x times with a/b split intentionally leaving out c%* |
| tune\_model = model\_selection.GridSearchCV: for a given classifier, list all reasonable values for different params, e*xhaustive search over these values for an the best set of values for this* classifier |
| feature\_select: *# for a given classifier, more features do not necessarily make a better model, but the right features do.* |
| multi\_classifier\_voting\_predication: similar to average out different models predictions, this lets models vote |
| *# support is the number of samples of the true response that lie in that class. # macro average (averaging the unweighted mean per label)* print(**'f1-score on myTree'**) print(metrics.classification\_report(data1[**'Survived'**], Tree\_Predict)) *# Build a text report showing the main classification metrics* |

Plan:

1. Use save\_df\_to\_csv(df, outFileName) for intermediate df saving
2. Use advanced engineering
3. Try one-hot result from last step
4. Stacking
   1. one-hot effect is almost the same, slightly worse
   2. 3rd stacking
5. Re-learn all models in multi\_classifier\_model\_compare, list all reasonable values for different params for these different classifiers,
   1. then select best param value combo for these classifiers (model\_selection.GridSearchCV),
   2. (here maybe insert feature\_select),
   3. then use these tuned models to vote for final prediction (multi\_classifier\_voting\_predication)

Generic Algorithm

<https://www.youtube.com/watch?v=RxTfc4JLYKs>

<https://www.kaggle.com/guesejustin/91-genetic-algorithms-explained-using-geap>

Based on Darwin’s theory of biological evolution, there are three points:

1) Heredity — There must be a process in place by which children receive the property of their parent;

2) Variation — There must be a variety of traits present in the population or a means with which to introduce a variation;

3) Selection — There must be a mechanism by which some members of the population can be parents and pass down their genetic information and some do not (survival for the fittest)

Situation: our goal is to use random string to produce the word “theory”.

Step 1: We start with say 4 (N) words of length 6, namely,

theoaa – 4 (40%)

abcdef – 1 (10%)

cddorx – 0 (0%)

heortk – 5 (50%)

Step 2: We have a fitness function to assess how fit an individual is, here we use letter match, as the number shown above.

Step 3 (N times for N new children):

1. Pick 2 parents, such as theoaa and herotk (based on letter match) or pick 2 based on the possibility, say we choose theoaa and herotk here;
2. Crossover: Create a child from the parent, such as the+otk or th+rotk, say we get the+otk=theotk here;
3. Mutation: we can have a 1% chance of mutation rate here, such as 1% chance to change the first letter t to some random letter, this is for the case that original population does not have the right letter/gene that we hope for (y in this case);
4. Add the new child to a new population

Step 4: replace the old population with the new (the new population should be fitter, meaning closer to our end goal “theory”)

A screenshot of a cell phone

Description automatically generated