T Q

Thursday, August 30, 2018

2:57 PM

* + Go through the **variable selection**process in python
  + Benefits:
  + It enables the machine learning algorithm to **train faster**.
  + It reduces the complexity of a model and makes it **easier to interpre**t.
  + It **improves the accuracy** of a model if the right subset is chosen.
  + It **reduces Overfitting**.

* + Method 1: **filter** method for preprocessing
  + **Independent** of any machine learning algorithm.
  + Features give **rank** on the basis of statistical scores which tend to determine the **features**' **correlation** with the **outcome** variable.
  + Example: use SelectKBest to select k highest scoring features

test = **SelectKBest**(score\_func=chi2, k=4)

fit = test.fit(**X, Y**)

* + Method 2: Forward Selection/Backward Elimination
  + Needs one machine learning **algorithm** and uses its **performance** as evaluation criteria. This method searches for a feature which is **best-suited** for the machine learning algorithm and aims to **improve** the mining performance. To evaluate the features, the predictive accuracy used for classification tasks and goodness of cluster is evaluated using clustering.
  + **Forward Selection**: The procedure starts with an **empty set** of features [reduced set]. The **best** of the original features is determined and added to the reduced set. At each subsequent iteration, the **best of the remaining** original attributes is added to the set.
  + **Backward Elimination**: The procedure starts with the **full set** of attributes. At each step, it removes the **worst attribute remaining** in the set
  + **Combination** of above two: at each step, the procedure **selects the best** attribute and **removes the worst** from among the remaining attributes.

* + Method 3: **Regularization methods** are the most commonly used **embedded** methods which **penalize** a feature given a coefficient **threshold**.
  + introduce **additional constraints** into the optimization of a predictive algorithm (such as a regression algorithm) that bias the model **toward lower complexity** (fewer coefficients).
  + **Examples** of regularization algorithms are the LASSO, Ridge Regression (key difference is in how they assign penalty to the coefficients)
  + **Ridge Regression:**
    - Performs **L2** regularization, i.e. adds penalty equivalent to **square of the magnitude** of coefficients
    - Minimization objective = LS Obj + α \* (sum of **square** of coefficients)
  + **Lasso Regression:**
    - Performs **L1** regularization, i.e. adds penalty equivalent to **absolute value of the magnitude** of coefficients
    - Minimization objective = LS Obj + α \* (sum of **absolute value** of coefficients)

#Beginner's Guide to Feature Selection in Python

<https://www.datacamp.com/community/tutorials/feature-selection-python>

import pandas as pd

import numpy as np

# original data: have invalid numbers like 0 for some

data = pd.read\_csv("h:/StatisticalLearning\_DataCamp\_python/diabetes.csv")

print(data.shape)

data.head()

# use a processed data instead

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

data\_processed = pd.read\_csv("h:/StatisticalLearning\_DataCamp\_python/pima-indians-diabetes.data.csv", names=names)

# convert the DataFrame object to a NumPy array to achieve faster computation

array = data\_processed.values

X = array[:,0:8]

Y = array[:,8]

# Method 1: implement a Chi-Squared statistical test for non-negative features to select 4 of the best features from the dataset

# Import the necessary libraries first

from sklearn.feature\_selection import SelectKBest

from sklearn.feature\_selection import chi2

# Feature extraction: SelectKBest removes all but the k highest scoring features

#For regression: f\_regression, mutual\_info\_regression

#For classification: chi2, f\_classif, mutual\_info\_classif

test = SelectKBest(score\_func=chi2, k=4)

fit = test.fit(X, Y)

# Summarize scores

np.set\_printoptions(precision=3)

print(fit.scores\_)

# You can see the scores for each attribute and the 4 attributes chosen (those with the highest scores): plas, test, mass, and age.

# This scores will help you further in determining the best features for training your model.

features = fit.transform(X)

# Summarize selected features

print(features[0:5,:])

# Method 2: Use Recursive Feature Elimination (RFE) to do feature selection

# Import your necessary dependencies

from sklearn.feature\_selection import RFE

from sklearn.linear\_model import LogisticRegression

# You will use RFE with the Logistic Regression classifier to select the top 3 features.

# The choice of algorithm does not matter too much as long as it is skillful and consistent.

# Feature extraction

lr = LogisticRegression()

rfe = RFE(lr, 3)

rfe\_fit = rfe.fit(X, Y)

print("Num Features: %s" % (rfe\_fit.n\_features\_))

print("Selected Features: %s" % (rfe\_fit.support\_))

print("Feature Ranking: %s" % (rfe\_fit.ranking\_))

# Method 3: use Ridge regression which is basically a regularization technique and an embedded feature selection techniques as well.

# more: <https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-ridge-lasso-regression-python/#three>

from sklearn.linear\_model import Ridge

ridge = Ridge(alpha=1.0)

ridge.fit(X,Y)

Ridge(alpha=1.0, copy\_X=True, fit\_intercept=True, max\_iter=None,

normalize=False, random\_state=None, solver='auto', tol=0.001)

#In order to better understand the results of Ridge regression, you will implement a little helper function

#that will help you to print the results in a better so that you can interpret them easily.

# A helper method for pretty-printing the coefficients

def pretty\_print\_coefs(coefs, names = None, sort = False):

if names == None:

names = ["X%s" % x for x in range(len(coefs))]

lst = zip(coefs, names)

if sort:

lst = sorted(lst, key = lambda x:-np.abs(x[0]))

return " + ".join("%s \* %s" % (round(coef, 3), name)

for coef, name in lst)

print ("Ridge model:", pretty\_print\_coefs(ridge.coef\_))

**Overfitting/Underfitting**: Use **decision tree** as example

* + Goals of Supervised Learning: Find a model that best approximates f : f(head) ≈ f; f(head) should acheive a **low predictive error** on **unseen** datasets
  + Evaluation: Use **cross-validation** (K-fold CV)
    - split whole data into **training/test** data; then use the **training** data to cross validate;
    - compare **CV** error (average of k fold sample fitting error), **training** error (whole training set fitting error), and **test** error(test set fitting error)

* + **Overfitting**:
    - f(head) fits training set noise, has low predictive power on the unseen datasets (test data)
    - High **variance** (variance: how much f(head) is **inconsistent over different training** sets.)
    - Detect: CV error > training error
    - Solution: **Decrease** **model complexity** (For Decision tree: decrease max depth, or increase min sample per leaf); gather more samples
  + **Underfitting**:
    - f(head) us not flexible enough to approximate f.
    - High **bias** (bias: error term that tells you on average how much **f(head) =! f** )
    - Detect: CV error close to training error >> desired error
    - Solution: **Increase model complexity** (For Decision tree: increase max depth, or decrease min sample per leaf); gather more relevant feature

Simple Decision tree (CART: classification and regression tree):

* + Good:
    - Flexibility: can be used for both classification/regression; can describe non-linear relationship (the decision boundary is not linear)
    - easy to use: only two main parameters (max depth/min sample per leaf); no need to some preprocessing (standardize or normalize features)
  + Bad:
    - For classification, can only produce orthogonal decision boundaries (rectangle)
    - Sensitive to small variations in the training datasets
    - Unconstrained decision tree may overfit easily (high variance)
  + Solution: **ensemble learning**

Ensemble learning:

* + Train **different models** on the same dataset. Let each model make its **predictions**, and **aggregates** predictions of individual models.
  + Make final prediction using some **Classifier** (like Voting); Final prediction will be more robust and less prone to errors.

* + **Voting** Classifier: same training set, **different algorithms**;
    - Example: Classification: can use logistic regression/KNN/DecisionTreeRegressor fit the data respectively, and make the final prediction by vote (e.p. predict the results using the results from two or more models)

* + **Bagging (Bootstrap** Aggregation**)**: same algorithm, different subsets of the **training set (using bootstrap)**, estimators use **all features** for training and prediction
    - Uses a technique known as the **Bootstrap**.
    - Good: **Reduces variance** of individual models in the ensemble
    - Bagging Classification: Aggregates predictions by **majority** voting. **BaggingClassifier** in scikit-learn
    - Bagging Regression: Aggregates predictions through **averaging**. **BaggingRegressor** in scikit-learn
    - Python example:

base\_estimator is the **algorithm** (can be Decision Tree, Logistic Regression, Neural Net) used to fit the training data;

n\_estimators is the **number of estimators** will be generated; trained on a distinct bootstrap sample of the training set

**oob\_score = True,** calculate the **hold-out accuracy** for each tree

# Instantiate dt

dt = DecisionTreeClassifier(min\_samples\_leaf=8, random\_state=1)

# Instantiate bc

bc = BaggingClassifier(**base\_estimator**=dt,

**n\_estimators**=50,

**oob\_score**=True,

random\_state=1)

* + **Random Forest:** Use **decision tree** as algorithm; Each estimator is trained on a different **bootstrap** sample having the **same size** as the training set
    - RF introduces further **randomization** in the **training** of individual trees; **d features** are sampled at each node without replacement ( d < total number of features )
    - RF Classification: Aggregates predictions by majority voting; RandomForestClassifier in scikit-learn
    - RF Regression: Aggregates predictions through averaging; RandomForestRegressor in scikit-learn
    - Feature **Importance**:
      * Tree-based methods enable measuring the **importance of each feature** in prediction;
      * how much the **tree nodes** use a particular feature (weighted average) to reduce impurity
      * In sklearn, accessed using the attribute **feature\_importance\_**

* + Basics of stats for Qi (p value/ANOVA/regression)
    - P value:

The **p-value** is the level of marginal significance for a hypothesis test representing the probability of a given event happens.

In a hypothesis testing, the calculated p-value is used to provide the smallest level of significance at which the null hypothesis would be rejected. That means, a smaller p-value means that there is stronger evidence in rejecting the null hypothesis.

For example, when we fit the coefficients in regression, the null hypothesis is that the coefficients is equal to zero (no effect). So if we choose the significant level threshold as 5% and the calculated p value is less than 5%, we can conclude that the null hypothesis should be rejected which means that the coefficient is significant. Therefore, you can use the coefficient's p value to determine variable to keep in the regression model.

* + Type 1/2 Error; Precision; Recall:
    - **Type 1 error**: predict positive when it is actually negative (Kill the innocent guy);

**Type 2 error**: predict negative when it is actually positive(Let go the guilty guy)

* + **Accuracy** = (TP+TN)/(TP+FP+TN+FN); Accuracy does **not perform well** with **imbalanced** data sets. For example, if you have 95 negative and 5 positive samples, **classifying all as negative** gives 0.95 accuracy score.

**Balanced Accuracy** = (TPR+TNR)/2 = [(TP/(TP+FN)+TN/(TN+FP)]/2; overcomes the problem of imbalanced data, by **normalizing** true positive and true negative predictions by the **number of positive and negative samples**, respectively, and divides their sum into two. Regarding the previous example (95 negative and 5 positive samples), classifying all as negative gives 0.5 balanced accuracy score out of the maximum one, which is equivalent to the expected value of a **random guess** of a **balanced** data. Since most of the real data sets are imbalanced, Balanced Accuracy metric is suggested instead of Accuracy metric.

* + **Precision** = TP / (TP+FP); **precision** is "how **useful** the search results are", the accuracy rate of all **predict positive**

**Recall** = TP / (TP+FN); **recall** is "how **complete** the results are", the accuracy rate of all **actual positive**

There is an **inverse relationship** between precision and recall, where it is possible to **increase** one at the cost of **reducing** the other; frequently plotted **against each other** as **ROC curves** and provide a principled mechanism to explore operating point **tradeoffs**.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **True condition** |  |  |  |  |
|  | [Total population](https://en.wikipedia.org/wiki/Statistical_population) | Condition positive | Condition negative | [Prevalence](https://en.wikipedia.org/wiki/Prevalence) = Σ Condition positive/Σ Total population | [Accuracy](https://en.wikipedia.org/wiki/Accuracy_and_precision) (ACC) = Σ True positive + Σ True negative/Σ Total population |  |
| **Predicted**  **condition** | Predicted condition  positive | [**True positive**](https://en.wikipedia.org/wiki/True_positive) | [**False positive**](https://en.wikipedia.org/wiki/False_positive),  [Type I error](https://en.wikipedia.org/wiki/Type_I_error) | [Positive predictive value](https://en.wikipedia.org/wiki/Positive_predictive_value) (PPV), [Precision](https://en.wikipedia.org/wiki/Precision_(information_retrieval)) = Σ True positive/Σ Predicted condition positive | [False discovery rate](https://en.wikipedia.org/wiki/False_discovery_rate) (FDR) = Σ False positive/Σ Predicted condition positive |  |
|  | Predicted condition  negative | [**False negative**](https://en.wikipedia.org/wiki/False_negative),  [Type II error](https://en.wikipedia.org/wiki/Type_II_error) | [**True negative**](https://en.wikipedia.org/wiki/True_negative) | [False omission rate](https://en.wikipedia.org/wiki/False_omission_rate) (FOR) = Σ False negative/Σ Predicted condition negative | [Negative predictive value](https://en.wikipedia.org/wiki/Negative_predictive_value) (NPV) = Σ True negative/Σ Predicted condition negative |  |
|  |  | [True positive rate](https://en.wikipedia.org/wiki/True_positive_rate) (TPR), [Recall](https://en.wikipedia.org/wiki/Recall_(information_retrieval)), [Sensitivity](https://en.wikipedia.org/wiki/Sensitivity_(tests)), probability of detection, [Power](https://en.wikipedia.org/wiki/Statistical_power)= Σ True positive/Σ Condition positive | [False positive rate](https://en.wikipedia.org/wiki/False_positive_rate) (FPR), [Fall-out](https://en.wikipedia.org/wiki/Information_retrieval), probability of false alarm = Σ False positive/Σ Condition negative | [Positive likelihood ratio](https://en.wikipedia.org/wiki/Positive_likelihood_ratio) (LR+) = TPR/FPR | [Diagnostic odds ratio](https://en.wikipedia.org/wiki/Diagnostic_odds_ratio)(DOR) = LR+/LR− | [F1 score](https://en.wikipedia.org/wiki/F1_score) = 2 · Precision · Recall/Precision + Recall |
|  |  | [False negative rate](https://en.wikipedia.org/wiki/False_negative_rate) (FNR), Miss rate = Σ False negative/Σ Condition positive | [Specificity](https://en.wikipedia.org/wiki/Specificity_(tests)) (SPC), Selectivity, [True negative rate](https://en.wikipedia.org/wiki/True_negative_rate) (TNR) = Σ True negative/Σ Condition negative | [Negative likelihood ratio](https://en.wikipedia.org/wiki/Negative_likelihood_ratio) (LR−) = FNR/TNR |  |  |

* + ROC Curve:
    - An **ROC** curve (receiver operating characteristic curve) is a graph showing the **performance of a classification** model at all classification **thresholds**. This curve plots two parameters:

True Positive Rate (Recall; TP/(TP+FN) )

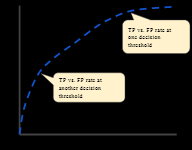
False Positive Rate (FP/(FP+TN))

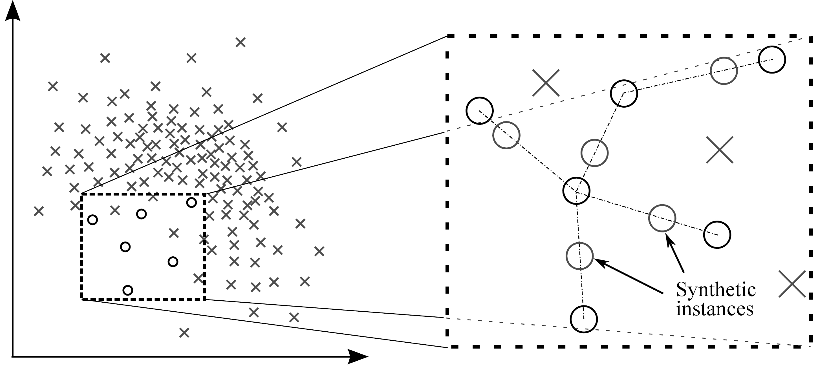
An ROC curve plots **TPR vs. FPR** at different classification **thresholds**. **Lowering** the classification threshold classifies more items as **positive**, thus increasing both False Positives and True Positives. The following figure shows a typical ROC curve.

X axis: FPR; Y axis: TPR

Right top point: use 0 as threshold, predict all to positive; TPR=1, FPR=1;

Left bottom point: use 1 as threshold, predict all to negative; TPR=0, FPR=0;

* + 
  + AUC: Area Under the ROC Curve:
    - AUC stands for "**Area under the ROC Curve**." That is, AUC measures the entire **two-dimensional area** underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1).
    - AUC provides an **aggregate measure of performance** across all possible classification **thresholds**.
    - AUC ranges in value **from 0 to 1**. A model whose predictions are **100% wrong** has an AUC of **0.0**; one whose predictions are **100% correct** has an AUC of **1.0**.
    - Good: AUC is classification-threshold-invariant. It measures the quality of the model's predictions **irrespective** of what classification **threshold** is chosen.
    - Bad: Classification-threshold invariance is **not always desirable**. In cases where there are **wide disparities** in the cost of false negatives vs. false positives, it may be critical to **minimize one type of classification error**.
      * For example, when doing **email spam** detection, you likely want to prioritize **minimizing false positives** (even if that results in a significant increase of false negatives). AUC isn't a useful metric for this type of optimization.
      * For example, when predict **default** rate, want to prioritize **minimizing false negatives** (not detect real default)

* + Imbalanced data: classifiers can have **good** accuracy on the **majority** class but very **poor** accuracy on the **minority** class(es) due to the influence that the larger majority class.
    - Change the performance **metric**:
      * **Accuracy** is **not** the best metric to use when evaluating imbalanced datasets as it can be very **misleading**.
      * Metrics that can provide better insight include:
        + **Confusion Matrix**: a table showing **correct** predictions and types of **incorrect** predictions; **Precision** + **Recall**
        + **F1 score** = 2 \* (precision \* recall) / (precision + recall); weighted average of the **precision** and **recall**, where an F1 score reaches its **best value at 1** and worst score at 0.
    - **Resampling** techniques: resample the original training dataset to **decrease** the overall level of class **imbalance**.
      * **Oversample** **minority** class:
        + adding **more copies** of the minority class. Oversampling can be a good choice when you **don’t have a ton of data** to work with.
        + Good: **No information is lost** in oversampling as all original instances of the minority and the majority classes are **retained** in the oversampled dataset.
        + Bad: random oversampling can lead to **overfitting** since it **replicates** the **minority** class events.
        + Always **split** into test and train sets **BEFORE** trying **oversampling** techniques! Oversampling before splitting the data can allow the **exact same observations** to be present in **both the test and train sets**. This can allow our model to simply memorize specific data points and **cause overfitting** and poor generalization to the test data.
      * **Undersample** **majority** class:
        + **removing** some observations of the majority class. Undersampling can be a good choice when you have **a ton of data** -think millions of rows.
        + Good: improve **run time and storage** problems by reducing the number of training data samples when the training data set is huge.
        + Bad: we may **remove information** that may be valuable. This could lead to **underfitting** and poor generalization to the test set.
      * **SMOTE** (Synthetic Minority Over-sampling Technique )
        + designed to **generate new samples** that are coherent with the **minor class distribution**. The main idea is to consider the relationships that exist between samples and **create new synthetic points** along the segments connecting a group of **neighbors**.
        + 
    - Try different **models**:
      * **Decision trees** frequently perform well on imbalanced data (usually better than logistic regression). They work by learning a hierarchy of **if/else questions** and this can **force both classes** to be addressed.

* + High-dimensional data:

* + Data treatments:
  + **Missing** value:
    - Missing reason:
    - Dropping **samples** (rows)
      * (1) Missing completely **at Random**:
        + A certain value is missing has nothing to do with its hypothetical value and with the values of other variables.
        + In this case it is **safe to remove the data** with missing values depending upon their occurrences: mydata.**dropna**(**inplace**=True)
      * (2) Missing **not at Random:** Two possible reasons:
        + The missing value depends on the **hypothetical value**: People with **high salaries** generally do **not want to reveal** their incomes in surveys, missing value in **salaries** variable is impacted by its salary level
        + The missing value is **dependent** on some **other variable’s value**: **females** generally don’t want to reveal their **ages,** missing value in **age** variable is impacted by **gender** variable.
        + In this case removing observations with missing values can **produce a bias** in the model. So we have to be **really careful** before removing observations. Note that **imputation** does **not necessarily give better** results.
        + Imputation using data from **other sources**:

Example: FICO, make no sense using median/average; use score from **other sources**, may need **transformation** because of the different ranges

For Risk Model: conservative (never underestimate the risk); if missing FICO, assign the **riskiest** bucket

* + Imputation with **Mean, Median and Mode**:
    - Good: very fast; Bad: reduce the **variance** of the datasets
    - from sklearn.preprocessing import Imputer

imputer = **Imputer**(missing\_values=’NaN’, strategy=’**mean**’)

transformed\_values = imputer.**fit\_transform**(values)

# strategy can be changed to "**median**" and “**most\_frequent**”

* + Imputation using **linear regression**: for very important variables that you don't want to drop or simply replaced by mean
    - several **predictors** of the variable with missing values are identified using a **correlation** matrix. The best predictors are selected and used as **independent variables** in a **regression** equation. The variable with missing data is used as the **dependent** variable.
    - Good: "theoretically” provides **good estimates** for missing values; Bad: tend to **fit together “too well”** and so standard error is deflated. Must also assume that there is a **linear** relationship
  + Imputation using **KNN** (K Nearest Neighbors):
    - The method requires the selection of the **number of nearest neighbors**, and a **distance** metric. KNN can predict both **discrete** attributes (the **most frequent** value among the k nearest neighbors) and **continuous** attributes (the **mean** among the k nearest neighbors)
    - Good: simple to **understand** and easy to **implement**; Bad: **time-consuming** when analyzing **large** datasets because it searches for similar instances through the entire dataset.
  + Dropping **variables** (columns)
    - it is always **better to keep** data than to discard it. Sometimes you can drop variables if the data is **missing for more than 60% observations** but only if that variable is insignificant. Having said that, **imputation** is always a **preferred** choice over dropping variables
    - mydata.**drop**('column\_name', axis=1, inplace=True)

* + **Outlier** detection:
    - Independent variable
    - Dependent variable:
    - For continuous outputs, can trimmed the samples to 99th percentile to remove outliers
    - Can use influence statistics
  + Data **transformation**:
    - Use **Box-Cox** transformation: get a distribution like a normal distribution:   
      A way to transform **non-normal dependent variables** into a **normal** shape. Normality is an important **assumption** for many statistical techniques;
    - Other transformation: log/change/difference/square root/binning/

ML method summary:

* + **Supervised Learning**: Finds patterns (and develops predictive models) using both, **input** data and **output** data.

All Supervised Learning techniques area form of either Classification or Regression.

* + **Classification**: used for predicting **discrete** responses.

For example:

Whether India will WIN or LOSE a Cricket match? Whether an email is SPAM or GENUINE?

WIN, LOSE, SPAM, GENUINE are the predefined classes. And output has to fall among these depending on the input.

* + **Regression**: used for predicting **continuous** responses.

For example:

Trend in stock market prices, Weather forecast, etc.

* + **Unsupervised Learning:**Finds patterns based **only on input** data. This technique is useful when you’re not quite sure what to look for. Often used for **exploratory Analysis** of raw data.

Most Unsupervised Learning techniques are a form of **Cluster** Analysis.

* + **Cluster Analysis:**

In Cluster Analysis, you group data items that have some **measure of similarity** based on characteristic values.

At the end what you will have is a set of different groups (Let’s assume A — Z such groups). A Data Item(d1) in one group(A) is very much similar to other Data Items(d2 — dx) in the same group(A), but d1 is significantly different from Data Items belonging to different groups (B — Z).

Experience with **Tableau**:

Learned in school. It is quite similar to excel pivot table. You drag the fields to the x or y axis, and choose the type of chart you want. And it will generate very nice plots.

Data visualization:

Key risk driver plots:

We will monitor some of the key risk drivers for different products. Take the loan feature above as example. We use **bar chart or pie chart** to show the distribution of these key risk drivers.

When the distribution of the key driver change a lot, we will take it very seriously since it may means that some of the assumption of previous model has change. We then will work with business to decide whether we need to recalibrate the model.

Sensitivity analysis:

We use **waterfall plot** to show the sensitivity analysis of some important macro-economic variables. Business team want us to show them the contribution of each variables to the final results. For example, after we build a regression model, although we can tell the rough effect of each variable by checking the coefficients of the regression model, but that is hard to **quantify the contribution**. So we will plus or minus one variable by its standard deviation and fix all the other variables, and we re-run the model to see the updated results. We do this for all variables and create the waterfall plot to show the contribution of each variable.

Use **Tableau**:

We usually use python or SAS to create plots as python; Many package can be used in python like Matplotlib/ggplot/seaborn;

But if I want to analyze your data **geographically**, I will plot the data on a map in Tableau. For example, I want to show the loss rates forecast for different regions, **a map view** is the best visualization tool for the data. The audience can see very clearly about the which state has higher loss rates, which state has more customers. For those states with low loss rates but fewer customer, meaning it may has huge potential for that region, we will recommend to the business departments asking them to target more customers there.

Model:

* + Additional drawn before default

DS:

How to **impute** missing values:

<https://towardsdatascience.com/how-to-handle-missing-data-8646b18db0d4>

<https://towardsdatascience.com/6-different-ways-to-compensate-for-missing-values-data-imputation-with-examples-6022d9ca0779>

PCA:

**Principal Component Analysis (PCA)** is a **dimensionality-reduction** technique that is often used to transform a **high**-dimensional dataset into a **smaller**-dimensional subspace **prior to running a machine learning algorithm** on the data.

<http://www.lauradhamilton.com/introduction-to-principal-component-analysis-pca>