FINAL PROJECT- CMPS 142 - Machine_Learning - "Titanic"

Project Members:

Ekaterina Tcareva (etcareva@ucsc.edu), Sneha Das(sndas@ucsc.edu)

Aim:
Our goal is to build a prediction on surviving Titanic passengers.

We have some information about passengers:

1. **PassengerId** -- A numerical id assigned to each passenger.
2. **Survived** -- Whether the passenger survived (1), or didn't (0). We'll be making predictions for this column.
3. **Pclass** -- The class the passenger was in -- first class (1), second class (2), or third class (3).
4. **Name** -- the name of the passenger.
5. **Sex** -- The gender of the passenger -- male or female.
6. **Age** -- The age of the passenger. Fractional.
7. **SibSp** -- The number of siblings and spouses the passenger had on board.
8. **Parch** -- The number of parents and children the passenger had on board.
9. **Ticket** -- The ticket number of the passenger.
10. **Fare** -- How much the passenger paid for the ticker.
11. **Cabin** -- Which cabin the passenger was in.
12. **Embarked** -- Where the passenger boarded the Titanic.

We also have some domain knowledge. We know that women and children had more chances of surviving than men. Also people travelling in first class were more likely to survive than people travelling in third class. Thus we consider **Pclass, Age,** and **Sex** as important features. We do not need **Fare** since it provides almost the same information as Pclass and Pclass is chosen since it is more related with location of the cabins on the Titanic (Also chances are that people from cabins on the upper level had more chances to survive.)

We also found information that there is a relation between the family size and surviving. People who traveled in group of 2-4 were more likely to survive. Thus, our main idea is to create a new feature: familySize = SibSp + Parch.

Our plan

- To read our csv file using **read_csv** function from PANDAS lib.
- To fix all NAs in the dataset using **fillna** function from PANDAS lib. We will use median for Age and familySize; the most common value for Sex and Pclass
- Replace the strings with integers in the Sex column using **loc** function from PANDAS lib
- To add a new feature familySize
    1. Then we can try different algorithms using cross validation. We will use **cross_validation** module from SKLEARN. We create 3 folders (we will train our algorithms on two folders 1 + 2, 1+ 3, 2 + 3 and test on one folder 3, 2, 1).

We will run some algorithms from SKLEARN (**Linear regression** and **random forest** are two models being considered now.)
2. We will choose the algorithm with will gives us the best result and train it on the whole set.
3. We will try to use boosting to make our prediction better.

Our work:

1. **First of all we fill NA and change strings in the "Sex" and "Embarked" to floats using "pandas":**

```
titanic['Sex'] = titanic['Sex'].fillna('male')

titanic.loc[titanic["Sex"] == "male", "Sex"] = 0

titanic.loc[titanic["Sex"] == "female", "Sex"] = 1
```

2. **Now we try different models and check which one is the best fit.**

A)We started with **Logistic regression** using cross validation (3 folders) from "sklearn":

```
#we are using almost all features except "name", "cabin", and "ticket" since we have too much NA for "cabin" and we do not know what the numbers of "ticket" and letters mean

predictors = ["Pclass", "Sex", "Age", "SibSp", "Parch", "Fare", "Embarked"]
```

This approach gives us **0.787878787879** on the training set with cross validation (3 folders)
and **0.799102132435** when we train our algorithm on whole dataset without cross-validation.

Since 0.799 is bigger than 0.788 we suggest that we have some overfitting. Thus, we can try to use few features.

Next we wanted to explore which of the features are more correlated with the labels. We used Logistic regression for only one feature with cross validation.

We got:

predictors = ["Pclass"]
0.679012345679

predictors = ["Sex"]
0.786756453423

predictors = ["Age"]
0.616161616162

predictors = ["Embarked"]
0.594837261504

predictors = ["Fare"]
0.672278338945

predictors = ["SibSp"]
0.616161616162

predictors = ["Parch"]
0.607182940516

We can see that logistic regression with only “Sex” feature gives us almost the same accuracy as with all features!

“Pclass” and “Fare” gave us almost the same prediction and hence we can suggest that they are very closely related to each other. Thus, we decided to check their relation. We tried to predict “Fare” using logistic regression and “Pclass” feature. We got 0.693573836191. It was surprising, but they do not have a strong correlation .Hence we could use them both.

To avoid overfitting we run logistic regression using only three features: “Sex”, “Pclass”, “Fare”:
This approach gave as **0.780022446689** on the training set with cross validation (3 folders)
and **0.785634118967** when we train our algorithm on the whole dataset without cross-validation.
It is important to note that the results with and without cross validation is almost same. Thus, now we avoid overfitting using few features.

At the beginning of our work we decided to create **a new feature "famillySize".** Thus. we are creating it:

titanic["FamilySize"] = titanic["SibSp"] + titanic["Parch"]

But the logistic regression with only one feature "FamilySize" gives us: 0.612794612795. So, this new feature is not very related with our labels. We tried logistic regression with "FamilySize" instead of "SibSp"and "Parch" and it gives us the same result as before.

B)Next we tried **Naive Bayes** since we can suggest that most of the features in the data set are independent (we did not use "Fare" since we will be using "Pclass"):

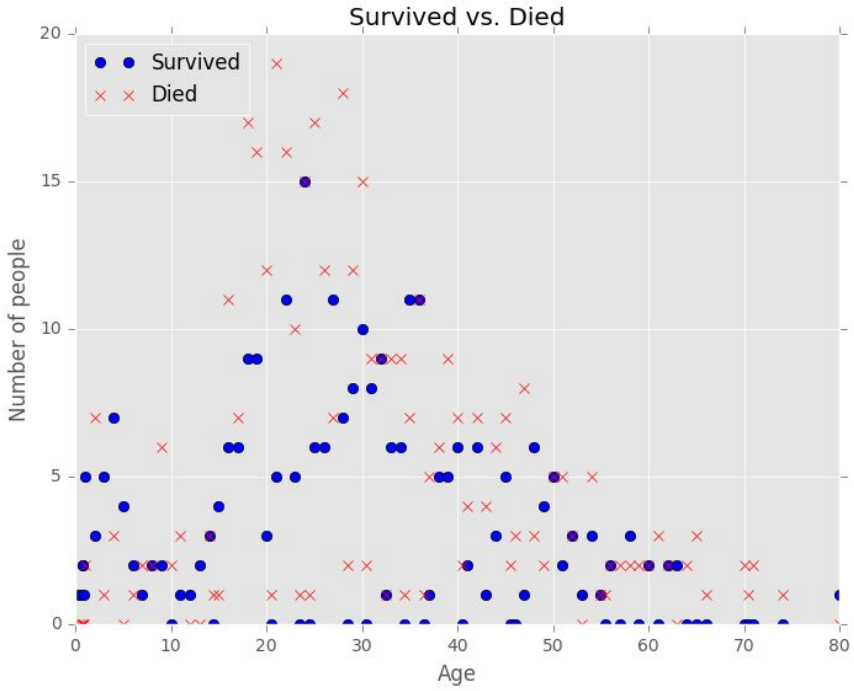
For the predictors = ["Pclass", "Sex", "Age", "SibSp", "Parch", "Embarked"]
we got **0.765432098765** using cross validation

For the predictors = ["Pclass", "Sex", "Age"]
we got **0.777777777778** using cross validation
The result is worser than the result we got using Logistic Regression. So, we decided to not explore this model deeper.

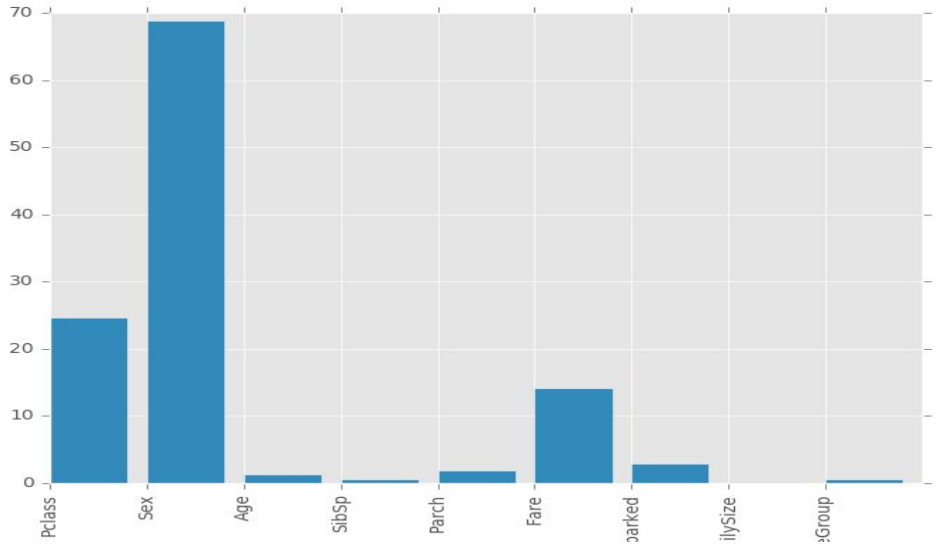
C)Next we tried the **Random Forest** model .We started by using maximum number of features because it looks like it could be useful to use a lot of features since the algorithm will create a lot of different trees (150 trees gives us the best result).

Our predictors = ["Pclass", "Sex", "Age", "SibSp", "Parch", "Fare", "Embarked", "FamilySize"]

We got **0.918069584736** with training on whole training set without cross validation. But only **0.817059483726** with cross validation. It is better than using logistic regression, but it is clear that we have overfitting again. Thus, we need to use fewer features. Also we think that we need to split the "Age" feature into some groups. For example: 0 -13, 13-17, 18-25, 26-50, 51-80. To find the best split we use matplotlib to plot the data:



Looking at this graph we decided to split the "Age" into three groups: <15, 16-30, >31
Then we created a new feature "AgeGroup": 0 for persons < 16, 1 for persons between 16 and 31, and 2 for persons older than 31.
Also, we have found a tool **SelectKBes**t in sklearn and tried to use it to find the best features:



From the above graph we see that Sex, Pclass, and Fare are the most important features![1] And we realised that "AgeGroup" is less important than age.

Using this information we tried some combinations of features and got the best result using:
predictors = ["Pclass", "Sex", "Age", "Parch", "Fare", "Embarked"]
We got **0.82379349046** using cross validation.

Next, to improve our model, we used GradientBoostingClassifier (we used the random forest with the same set of features).
We got 0.824915824916 with cross validation.

The result is slightly better than the result we got by just using Random Forest with the same parameters.

We submitted this result on Kaggle and it gave us **2392nd/3616** and about 0.777 points. We realized we had overfitting again! Hence we decided to try the same algorithms with few features:
predictors = ["Pclass", "Sex", "Fare", "Embarked"]
We got 0.806958473625 with cross validation and decided to submit this script. We got **891st/3661 (0.79904 accuracy)**

https://www.kaggle.com/etcareva/results

Thus, the algorithm which was less accurate on the training data (we used 3 folder cross validation to test it), gave us a better result on the test data. The conclusion is that our first algorithm was overfitting.

But when we tried to reduce the number of predictors further:
predictors = ["Pclass", "Sex", "Fare"]
and we submitted a new script, which gave us the same result as predictors = ["Pclass", "Sex", "Fare", "Embarked"] on the training data, the result was worse.

We also tried **SVM and KNN.** But we got only 0.68125701459 using cross validation for SVM and 0.703703703704 using cross validation for KNN. Thus, we decided not to explore these models deeper.

[1] Unfortunately, we did not find how this tool calculate the height of the bar on the graph. But it shows "best features from the data" (https://www.dataquest.io/mission/75/improving-your-submission)

Conclusion:

We got the best result using Random Forest algorithm with GradientBoostingClassifier using predictors = ["Pclass", "Sex", "Fare", "Embarked"]. We got 0.806958473625 with cross validation, we decided to submit this script and we we got **891st/3661 (0.79904 accuracy).**

We think, it is possible to create a better algorithm. A big improvement could be using **one hot encoding** for "Sex", "Pclass", and "Embarked" features. Also, it is possible to use "Cabin" feature.We could create a feature "isCabinKnown" .It could be a relation between "Survived" and "isCabinKnown" since we are more likely to know the cabins of survived passengers.

Model	**Features**	**Without cross validation**	**With 3 folders cross validation**	**Score of submission**
Logistic Regression	["Pclass", "Sex", "Age", "SibSp", "Parch", "Fare", "Embarked"]	**0.799102132435**	**0.787878787879**	
Naive Bayes	["Pclass", "Sex", "Age", "SibSp", "Parch", "Embarked"]		0.765432098765	
Naive Bayes	["Pclass", "Sex", "Age"]		0.777777777778	
Random Forest	["Pclass", "Sex", "Age", "SibSp", "Parch", "Fare", "Embarked", "FamilySize"]	**0.918069584736**	**0.817059483726**	

Gradient Boosting Classifier with random forest	["Pclass", "Sex", “Age”, "SibSp", "Parch", "Fare", "Embarked", "FamilySize"]		0.824915824916	0.77777
Gradient Boosting Classifier with random forest	["Pclass", "Sex", "Fare", "Embarked"		0.806958473625	**0.79904**
SVM	["Pclass", "Sex", "Age", "SibSp", "Parch", "Fare", "Embarked", "FamilySize", "AgeGroup"]		0.681257014459	
KNN	["Pclass", "Sex", "Age", "SibSp", "Parch", "Fare", "Embarked", "FamilySize", "AgeGroup"]		0.703703703704	